




## Extensible 3D (X3D)

# ISO/IEC 19775-1:20xx



This document is Edition 4 of ISO/IEC 19775-1, Extensible 3D (X3D). The full title of the International Standard is: *Information technology — Computer graphics, image processing and environmental data representation — Extensible 3D (X3D)*.

Parts	Description
 <a href="#">Part 1: Architecture and base components</a>	Part 1 contains the abstract functional specification for the X3D framework, and definitions of the standardized components and profiles.






## Extensible 3D (X3D)

# ISO/IEC 19775-1:20xx



This document is Edition 4 of ISO/IEC 19775-1, Extensible 3D (X3D). The full title of the International Standard is: *Information technology — Computer graphics, image processing and environmental data representation — Extensible 3D (X3D)*.

Parts	Description
 <a href="#">Part 1: Architecture and base components</a>	Part 1 contains the abstract functional specification for the X3D framework, and definitions of the standardized components and profiles.






































## Extensible 3D (X3D) Part 1: Architecture and base components

ISO/IEC 19775-1:2013 202x



This document is Edition 3 4 of ISO/IEC 19775-1, Extensible 3D (X3D). The full title of this part of the International Standard is: *Information technology — ~~Computer graphics and image processing~~ Computer graphics, image processing and environmental data representation — Extensible 3D (X3D) — Part 1: Architecture and base components.*

Background	Clauses		Annexes
<a href="#">Foreword</a>	1 <a href="#">Scope</a>	22 <a href="#">Environmental sensor component</a>	A <a href="#">Core profile</a>
<a href="#">Introduction</a>	2 <a href="#">Normative references</a>	23 <a href="#">Navigation component</a>	B <a href="#">Interchange profile</a>
	3 <a href="#">Definitions, acronyms, and abbreviations</a>	24 <a href="#">Environmental effects component</a>	C <a href="#">Interactive profile</a>
	4 <a href="#">Concepts</a>	25 <a href="#">Geospatial component</a>	D <a href="#">MPEG-4 interactive profile</a>
	5 <a href="#">Field type reference</a>	26 <a href="#">Humanoid Animation (HAnim) component</a> (H-Anim)	E <a href="#">Immersive profile</a>
	6 <a href="#">Conformance</a>	27 <a href="#">NURBS component</a>	F <a href="#">Full profile</a>
	7 <a href="#">Core component</a>	28 <a href="#">Distributed interactive simulation (DIS) component</a>	G <a href="#">Recommended navigation behaviours</a>
	8 <a href="#">Time component</a>	29 <a href="#">Scripting component</a>	H <a href="#">CADInterchange profile</a>
	9 <a href="#">Networking component</a>	30 <a href="#">Event utilities component</a>	I <a href="#">OpenGL shading language (GLSL) binding</a>

	 10 <a href="#">Grouping component</a>	 31 <a href="#">Programmable shaders component</a>	 J <a href="#">Microsoft high level shading language (HLSL) binding</a>
	 11 <a href="#">Rendering component</a>	 32 <a href="#">CAD geometry component</a>	 K <a href="#">nVidia Cg shading language binding</a>
	 12 <a href="#">Shape component</a>	 33 <a href="#">Texturing3D component</a>	 L <a href="#">MedicalInterchange profile</a>
	 13 <a href="#">Geometry3D component</a>	 34 <a href="#">Cube map environmental texturing component</a>	 Z <a href="#">Version content</a>
	 14 <a href="#">Geometry2D component</a>	 35 <a href="#">Layering component</a>	 <a href="#">Bibliography</a>
	 15 <a href="#">Text component</a>	 36 <a href="#">Layout component</a>	 <a href="#">Component index</a>
	 16 <a href="#">Sound component</a>	 37 <a href="#">Rigid body physics component</a>	 <a href="#">Profile index</a>
	 17 <a href="#">Lighting component</a>	 38 <a href="#">Picking sensor component</a>	 <a href="#">Node index</a> <a href="#">Node, abstract node type, and abstract interface index</a>
	 18 <a href="#">Texturing component</a>	 39 <a href="#">Followers component</a>	
	 19 <a href="#">Interpolation component</a>	 40 <a href="#">Particle systems component</a>	
	 20 <a href="#">Pointing device sensor component</a>	 41 <a href="#">Volume rendering component</a>	
	 21 <a href="#">Key device sensor component</a>	 42 <a href="#">Projective texture mapping component</a>	
		 43 <a href="#">Annotation component</a>	

The **Foreword** provides background on the standards process for X3D. The **Introduction** describes the purpose, design criteria, and functional characteristics of X3D. The following clauses define Part 1 of ISO/IEC 19775:

1. **Scope** defines the problem area that X3D addresses.
2. **Normative references** lists the normative standards referenced in this part of

ISO/IEC 19775. (editorial updates to latest versions)

3. **Definitions, acronyms, and abbreviations** contains the glossary of terminology used in this part of ISO/IEC 19775.
4. **Concepts** describes the workings of the X3D runtime system.
5. **Field type reference** describes the fundamental data types in X3D (expected addition of HTML5 event-model and DEF/id relationships).
6. **Conformance** describes the conformance requirements for X3D implementations.
7. **Core component** provides a detailed specification of the Core component of X3D.
8. **Time component** provides a detailed specification of the Time component of X3D.
9. **Networking component** provides a detailed specification of the Networking component of X3D (proposed changes to Inline content, security precautions).
10. **Grouping component** provides a detailed specification of the Grouping component of X3D.
11. **Rendering component** provides a detailed specification of the Rendering component of X3D.
12. **Shape component** provides a detailed specification of the Shape component of X3D (proposed node PointProperties, expected node ExternalShape, Material extensions for textures and their mapping, PhysicalMaterial, UnlitMaterial).
13. **Geometry3D component** provides a detailed specification of the Geometry3D component of X3D.
14. **Geometry2D component** provides a detailed specification of the Geometry2D component of X3D.
15. **Text** provides a detailed specification of the Text component of X3D.
16. **Sound component** provides a detailed specification of ~~the Time component of X3D~~ audio generation, spatialized sound, and acoustic rendering.
17. **Lighting component** provides a detailed specification of the Lighting component of X3D (lighting model rewritten, to account for Phong, physical and unlit models, and to clarify texture sampling and Gouraud shading).
18. **Texturing component** provides a detailed specification of the Texturing component of X3D (expected addition of ImageTextureAtlas, X3DSingleXxx abstract types and mapping fields).
19. **Interpolation component** provides a detailed specification of the Interpolation component of X3D.
20. **Pointing device sensor component** provides a detailed specification of the Pointing device sensor component of X3D.
21. **Key device sensor component** provides a detailed specification of the Key device sensor component of X3D.
22. **Environmental sensor component** provides a detailed specification of the Environmental sensor component of X3D.
23. **Navigation component** provides a detailed specification of the Navigation component of X3D.
24. **Environmental effects component** provides a detailed specification of the Environmental effects component of X3D.
25. **Geospatial component** provides a detailed specification of the Geospatial component of X3D.
26. **Humanoid animation (H-Anim HAnim) component** provides a detailed

- specification of **the Humanoid animation (H-Anim) component of X3D, Humanoid structure and motion animation.**
27. **NURBS component** provides a detailed specification of the NURBS component of X3D.
  28. **Distributed interactive simulation (DIS) component** provides a detailed specification of the DIS component of X3D.
  29. **Scripting component** provides a detailed specification of the Scripting component of X3D.
  30. **Event utilities component** provides a detailed specification of the Event utilities component of X3D.
  31. **Shader component** provides a detailed specification of the Shader component of X3D.
  32. **CAD geometry component** provides a detailed specification of the CAD geometry component of X3D.
  33. **Texturing3D component** provides a detailed specification of the 3D texturing component of X3D.
  34. **Environmental texturing component** provides a detailed specification of the environmental texturing component of X3D.
  35. **Layering component** provides a detailed specification for organizing the content of worlds into independent, overlapping layers.
  36. **Layout component** provides a detailed specification for arranging content to appear in specific regions of the display surface.
  37. **Rigid body physics component** provides a detailed specification for applying rigid body physics properties to content.
  38. **Picking sensor component** provides a detailed specification for selecting items in the content by user interaction.
  39. **Followers component** provides a detailed specification for using follower transitions.
  40. **Particle systems component** provides a detailed specification for specifying and using particle systems in X3D worlds.
  41. **Volume rendering component** provides a detailed specification for the rendering of volumetric data sets as part of X3D worlds.
  42. **Projective texture mapping component** provides a detailed specification for projecting textures onto geometry.
  43. **Annotation component** provides a detailed specification on how to present information that always faces the viewer (incomplete, not accepted).

There are several annexes included in the specification:

- A. **Core profile** defines a minimal subset of X3D functionality that constitutes the Core profile.
- B. **Interchange profile** defines the proper subset of X3D functionality that constitutes the Interchange profile.
- C. **Interactive profile** defines the proper subset of X3D functionality that constitutes the Interactive profile.
- D. **MPEG-4 interactive profile** defines the proper subset of X3D functionality that constitutes the MPEG-4 interactive profile.

- E. **Immersive profile** defines the proper subset of X3D functionality that corresponds to the base profile defined in ISO/IEC 14772-1.
- F. **Full profile** defines the proper subset of X3D functionality that constitutes the Full profile.
- G. **Recommended navigation behaviours** specifies some recommended behaviours that may be adopted by browser implementers.
- H. **CADInterchange profile** defines the proper subset of X3D functionality that constitutes the CADInterchange profile.
- I. **OpenGL shading language (GLSL) binding** provides a mapping of Programmable shader component functionality to the GLSL shading language.
- J. **Microsoft DirectX shading language (HLSL) binding** provides a mapping of Programmable shader component functionality to the HLSL shading language.
- K. **nVidia CG shading language binding** provides a mapping of Programmable shader component functionality to the Cg shading language.
- L. **MedicalInterchange profile** defines the proper subset of X3D functionality that constitutes the MedicalInterchange profile.
- Z. **Version content** specifies which X3D functionality is in which version.

**Bibliography** lists the informative, non-standard topics referenced in this part of ISO/IEC 19775.

**Component index** lists the available components defined in this part of ISO/IEC 19775 in alphabetical order with hyperlinks to their respective definitions.

**Profile index** lists the profiles defined in this part of ISO/IEC 19775 in alphabetical order with hyperlinks to their respective definitions.

**Node index** **Node, abstract node type, and abstract interface index** lists the nodes defined in this part of ISO/IEC 19775 in alphabetical order with hyperlinks to their respective definitions.







## Extensible 3D (X3D) Part 1: Architecture and base components

### Foreword



ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO and IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see [www.iso.org/directives](http://www.iso.org/directives)).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see [www.iso.org/patents](http://www.iso.org/patents)).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see the following URL: [www.iso.org/iso/foreword.html](http://www.iso.org/iso/foreword.html).

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology, Subcommittee 24, Computer graphics, image processing and environmental data representation*, in collaboration with The Web3D Consortium, Inc. (<http://www.web3d.org>).

This fourth edition cancels and replaces the third edition (ISO 19775-1:2013), which has been technically revised.



A list of all parts in the ISO 19775 series can be found on the ISO website.




draft



# Information technology — Computer graphics, image processing and environmental **data** representation — Extensible 3D (X3D) — Part 1: Architecture and base components

## 1 Scope

---



ISO/IEC 19775 **Extensible 3D (X3D)** defines a software system that integrates network-enabled 3D graphics and multimedia. Conceptually, each X3D application is a 3D time-based space that contains graphic and aural objects that can be dynamically modified through a variety of mechanisms. This part of ISO/IEC 19775 defines the architecture and base components of X3D.

The semantics of X3D describe an abstract functional behaviour of time-based, interactive 3D, multimedia information. This part of ISO/IEC 19775 does not define physical devices or any other implementation-dependent concepts (*e.g.*, screen resolution and input devices). This part of ISO/IEC 19775 is intended for a wide variety of devices and applications, and provides wide latitude in interpretation and implementation of the functionality. For example, this part of ISO/IEC 19775 does not assume the existence of a mouse or 2D display device.

Each X3D application:

- a. implicitly establishes a world coordinate space for all objects defined, as well as all objects included by the application;
- b. explicitly defines and composes a set of 3D and multimedia objects;
- c. can specify hyperlinks to other files and applications;
- d. can define programmatic or data-driven object behaviours;
- e. can connect to external modules or applications via programming and scripting languages;
- f. explicitly declares its functional requirements by specifying a profile;
- g. can declare additional functional requirements by specifying components.





## Extensible 3D (X3D)

### Part 1: Architecture and base components

# 22 Environmental sensor component



## 22.1 Introduction

### 22.1.1 Name

The name of this component is "EnvironmentalSensor". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

### 22.1.2 Overview

This clause describes the Environmental Sensor component of this part of ISO/IEC 19775. [Table 22.1](#) provides links to the major topics in this clause.

**Table 22.1 — Topics**

- [22.1 Introduction](#)
  - [22.1.1 Name](#)
  - [22.1.2 Overview](#)
- [22.2 Concepts](#)
- [22.3 Abstract types](#)
  - [22.3.1 X3DEnvironmentalSensorNode](#)
- [22.4 Node reference](#)
  - [22.4.1 ProximitySensor](#)
  - [22.4.2 TransformSensor](#)
  - [22.4.3 VisibilitySensor](#)
- [22.5 Support levels](#)
- [Table 22.1 — Topics](#)
- [Table 22.2 — Environmental sensor component support levels](#)

## 22.2 Concepts



Environment sensors are nodes which emit events based on some event which occurs within the environment, usually an interaction between two elements within the world. Most environment sensors events occur because of an interaction between the viewer and the world. However, an environment sensor event may also occur because of an interaction between a non-manipulable piece of hardware (e.g., a clock) and the world, between two objects in the world, or an event over the network.

Environmental sensors include:

- [Collision](#)
- [ProximitySensor](#)
- [TransformSensor](#)
- [VisibilitySensor](#)

The [Collision](#) grouping node detects when the user collides with objects in the virtual world. Proximity, collision, and visibility sensors are each processed independently of whether others exist or overlap. See [23 Navigation component](#) for more information.

The [ProximitySensor](#) detects when the user navigates into a specified region in the world.

The [TransformSensor](#) detects when for the target object specified enters, exits, or is transformed within a specified rectangular parallelepiped.

The [VisibilitySensor](#) detects when a specific part of the world becomes visible to the user.

When environmental sensors are inserted into the transformation hierarchy and before the presentation is updated (*i.e.*, read from file or created by a script), they shall generate events indicating any conditions which the sensor is intended to detect. The conditions for individual sensor types to generate these initial events are defined in the individual node specifications in [22.4 Node reference](#).

## 22.3 Abstract types

### 22.3.1 X3DEnvironmentalSensorNode

```
X3DEnvironmentalSensorNode : X3DSensorNode {
  SFVec3f/d [in,out] center 0 0 0 (-∞,∞)
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFVec3f [in,out] size 0 0 0 (-∞,∞)
  SFTime [out] enterTime
  SFTime [out] exitTime
  SFBool [out] isActive
}
```

The `X3DEnvironmentalSensorNode` **abstract node type** is the base type for the environmental sensor nodes [ProximitySensor](#) and [VisibilitySensor](#).

## 22.4 Node reference

### 22.4.1 ProximitySensor

```

ProximitySensor : X3DEnvironmentalSensorNode {
  SFVec3f  [in,out] center      0 0 0 (-∞,∞)
  SFBool   [in,out] enabled    TRUE
  SFNode   [in,out] metadata   NULL [X3DMetadataObject]
  SFVec3f  [in,out] size       0 0 0 [0,∞)
  SFTime   [out]   enterTime
  SFTime   [out]   exitTime
  SFVec3f  [out]   centerOfRotation_changed
  SFBool   [out]   isActive
  SFRotation [out] orientation_changed
  SFVec3f  [out]   position_changed
}

```

The `ProximitySensor` node generates events when the viewer enters, exits, and moves within a region in space (defined by a box). A proximity sensor is enabled or disabled by sending it an *enabled* event with a value of `TRUE` or `FALSE`. A disabled sensor does not send events.

A `ProximitySensor` node generates *isActive* `TRUE/FALSE` events as the viewer enters and exits the rectangular box defined by its *center* and *size* fields. Browsers shall interpolate viewer positions and timestamp the *isActive* events with the exact time the viewer first intersected the proximity region. The *center* field defines the centre point of the proximity region in object space. The *size* field specifies a vector which defines the width (x), height (y), and depth (z) of the box bounding the region. The components of the *size* field shall be greater than or equal to zero. `ProximitySensor` nodes are affected by the hierarchical transformations of their parents.

The *enterTime* event is generated whenever the *isActive* `TRUE` event is generated (user enters the box), and *exitTime* events are generated whenever an *isActive* `FALSE` event is generated (user exits the box).

The *centerOfRotation\_changed* field sends events whenever the user is contained within the proximity region and the center of rotation of the viewer for `EXAMINE` mode changes with respect to the `ProximitySensor` node's coordinate system. This may result when the bound [Viewpoint](#) nodes's center of rotation changes, when a new viewpoint is bound, when the user changes the center of rotation through the browser's user interface, or from changes to the `ProximitySensor` node's coordinate system. *centerOfRotation\_changed* events are only generated when the currently bound [NavigationInfo](#) node includes `LOOKAT` navigation. For more information, see [23.3.1 X3DViewpointNode](#) and [23.4.4. NavigationInfo](#).

The *position\_changed* and *orientation\_changed* fields send events whenever the user is contained within the proximity region and the position and orientation of the viewer changes with respect to the `ProximitySensor` node's coordinate system including enter and exit times. The viewer movement may be a result of a variety of circumstances resulting from browser navigation, `ProximitySensor` node's coordinate system changes, or bound `Viewpoint` node's position or orientation changes.

Each `ProximitySensor` node behaves independently of all other `ProximitySensor` nodes. Every enabled `ProximitySensor` node that is affected by the viewer's movement receives and sends events, possibly resulting in multiple `ProximitySensor` nodes receiving and sending events simultaneously. Unlike [TouchSensor](#) nodes, there is no notion of a `ProximitySensor` node lower in the scene graph "grabbing" events.

Instanced (DEF/USE) `ProximitySensor` nodes use the union of all the boxes to check for enter and exit. A multiply instanced `ProximitySensor` node will detect enter and exit for all instances of the box and send enter/exit events appropriately. For non-overlapping

bounding boxes, *position\_changed* and *orientation\_changed* events are calculated relative to the coordinate system associated with the bounding box in which the proximity was detected. However, the results are undefined if the any of the boxes of a multiply instanced ProximitySensor node overlap.

A ProximitySensor node that surrounds the entire world has an *enterTime* equal to the time that the world was entered and can be used to start up animations or behaviours as soon as a world is loaded. A ProximitySensor node with a box containing zero volume (*i.e.*, any *size* field element of 0.0) cannot generate events. This is equivalent to setting the *enabled* field to `FALSE`.

A ProximitySensor read from an X3D file shall generate *isActive* `TRUE`, *position\_changed*, *orientation\_changed* and *enterTime* events if the sensor is enabled and the viewer is inside the proximity region or as soon as the ProximitySensor is enabled. A ProximitySensor inserted into the transformation hierarchy shall generate *isActive* `TRUE`, *position\_changed*, *orientation\_changed* and *enterTime* events if the sensor is enabled and the viewer is inside the proximity region. A ProximitySensor removed from the transformation hierarchy shall generate *isActive* `FALSE`, *position\_changed*, *orientation\_changed* and *exitTime* events if the sensor is enabled and the viewer is inside the proximity region.

## 22.4.2 TransformSensor

```

TransformSensor : X3DEnvironmentalSensorNode {
  SFVec3f [in,out] center      0 0 0 (-∞,∞)
  SFBool  [in,out] enabled    TRUE
  SFNode  [in,out] metadata   NULL [X3DMetadataObject]
  SFVec3f [in,out] size       0 0 0 [0,∞)
  SFNode  [in,out] targetObject NULL [X3DGroupingNode|X3DShapeNode]
  SFTime  [out]  enterTime
  SFTime  [out]  exitTime
  SFBool  [out]  isActive
  SFRotation [out] orientation_changed
  SFVec3f [out]  position_changed
}

```

The TransformSensor node generates events when its target object enters, exits, and moves within a region in space (defined by a box). The target object can be any valid [X3DShapeNode](#) or [X3DGroupingNode](#) node. A TransformSensor is enabled or disabled by sending it an *enabled* event with a value of `TRUE` or `FALSE`. A disabled sensor does not send events.

A TransformSensor node generates *isActive* `TRUE/FALSE` events as the target object enters and exits the rectangular box defined by its *center* and *size* fields. Browsers shall timestamp the *isActive* events with the exact time the target object first intersected the proximity region. The *center* field defines the centre point of the proximity region in object space. The *size* field specifies a vector that defines the width (x), height (y), and depth (z) of the box bounding the region. The components of the *size* field shall be greater than or equal to zero. TransformSensor nodes are affected by the hierarchical transformations of their parents.

The *enterTime* event is generated whenever the *isActive* `TRUE` event is generated (target object enters the box), and *exitTime* events are generated whenever an *isActive* `FALSE` event is generated (target object exits the box).

The *position\_changed* and *orientation\_changed* fields send events whenever the target object is contained within the proximity region and the position and orientation of the

target object changes with respect to the TransformSensor node's coordinate system including enter and exit times. The object movement may be a result of a variety of circumstances resulting from the TransformSensor node's coordinate system changes, changes to the target object's position or orientation, or changes to the coordinate system of any of the ancestors or the target object.

Each TransformSensor node behaves independently of all other TransformSensor nodes. Every enabled TransformSensor node that is affected by the target object's movement receives and sends events, possibly resulting in multiple TransformSensor nodes receiving and sending events simultaneously. Unlike TouchSensor nodes, there is no notion of a TransformSensor node lower in the scene graph "grabbing" events.

Instanced (DEF/USE) TransformSensor nodes use the union of all the boxes to check for enter and exit. A multiply instanced TransformSensor node will detect enter and exit for all instances of the box and send enter/exit events appropriately. For non-overlapping bounding boxes, *position\_changed* and *orientation\_changed* events are calculated relative to the coordinate system associated with the bounding box in which the proximity was detected. However, the results are undefined if the any of the boxes of a multiply instanced TransformSensor node overlap.

A TransformSensor node with a box containing zero volume (*i.e.*, any *size* field element of 0.0) cannot generate events. This is equivalent to setting the *enabled* field to `FALSE`.

A TransformSensor read from an X3D file shall generate *isActive* `TRUE`, *position\_changed*, *orientation\_changed* and *enterTime* events if the sensor is enabled and the target object is inside the proximity region. A TransformSensor inserted into the transformation hierarchy shall generate *isActive* `TRUE`, *position\_changed*, *orientation\_changed* and *enterTime* events if the sensor is enabled and the target object is inside the proximity region. A TransformSensor removed from the transformation hierarchy shall generate *isActive* `FALSE`, *position\_changed*, *orientation\_changed* and *exitTime* events if the sensor is enabled and the target object is inside the proximity region.

### 22.4.3 VisibilitySensor

```
VisibilitySensor : X3DEnvironmentalSensorNode {
  SFVec3f [in,out] center 0 0 0 (-∞,∞)
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFVec3f [in,out] size 0 0 0 [0,∞)
  SFTime [out] enterTime
  SFTime [out] exitTime
  SFBool [out] isActive
}
```

The VisibilitySensor node detects visibility changes of a rectangular box as the user navigates the world. VisibilitySensor is typically used to detect when the user can see a specific object or region in the scene in order to activate or deactivate some behaviour or animation. The purpose is often to attract the attention of the user or to improve performance. Intermediate occluding geometry between the current viewpoint and the sensed volume has no effect on the behavior of the VisibilitySensor.

The *enabled* field enables and disables the VisibilitySensor node. If *enabled* is set to `FALSE`, the VisibilitySensor node does not send events. If *enabled* is `TRUE`, the VisibilitySensor node detects changes to the visibility status of the box specified and sends events through the *isActive* field. A `TRUE` event is output to *isActive* when any



portion of the box impacts the rendered view. A `FALSE` event is sent when the box has no effect on the view. Browsers shall guarantee that, if `isActive` is `FALSE`, the box has absolutely no effect on the rendered view. Browsers may err liberally when `isActive` is `TRUE`. For example, the box may affect the rendering.

The fields `center` and `size` specify the object space location of the box centre and the extents of the box (*i.e.*, width, height, and depth). The `VisibilitySensor` node's box is affected by hierarchical transformations of its parents. The components of the `size` field shall be greater than or equal to zero.

The `enterTime` event is generated whenever the `isActive TRUE` event is generated, and `exitTime` events are generated whenever `isActive FALSE` events are generated. A `VisibilitySensor` read from an X3D file shall generate `isActive TRUE` and `enterTime` events if the sensor is enabled and the visibility box is visible. A `VisibilitySensor` inserted into the transformation hierarchy shall generate `isActive TRUE` and `enterTime` events if the sensor is enabled and the visibility box is visible. A `VisibilitySensor` removed from the transformation hierarchy shall generate `isActive FALSE` and `exitTime` events if the sensor is enabled and the visibility box is visible.

Each `VisibilitySensor` node behaves independently of all other `VisibilitySensor` nodes. Every enabled `VisibilitySensor` node that is affected by the user's movement receives and sends events, possibly resulting in multiple `VisibilitySensor` nodes receiving and sending events simultaneously. Unlike `TouchSensor` nodes, there is no notion of a `VisibilitySensor` node lower in the scene graph "grabbing" events. Multiply instanced `VisibilitySensor` nodes (*i.e.*, DEF/USE) use the union of all the boxes defined by their instances. An instanced `VisibilitySensor` node shall detect visibility changes for all instances of the box and send events appropriately.

## 22.5 Support levels

The Environmental Sensor component provides two levels of support as specified in [Table 22.2](#). Level 1 is intended to enable automatic animations by supporting a simplified `ProximitySensor` node. Level 2 provides full environment sensing support.

**Table 22.2 — Environmental sensor component support levels**

Level	Prerequisites	Nodes	Support
1	Core 1 Time 1 Grouping 1 Navigation 1		
		<code>X3DEnvironmentSensorNode</code> (abstract)	n/a
		<code>ProximitySensor</code>	<code>position_changed</code> optionally supported. <code>orientation_changed</code> optionally supported.
	Core 1		

2	Time 1 Grouping 1 Navigation 1		
		All Level 1 Environmental Sensor nodes	All fields as supported in Level 1.
		ProximitySensor	All fields fully supported.
		VisibilitySensor	All fields fully supported.
3	Core 1 Time 1 Grouping 1 Navigation 1		
		All Level 2 Environmental Sensor nodes	All fields as supported in Level 2.
		TransformSensor	All fields fully supported.





# Extensible 3D (X3D) Part 1: Architecture and base components

## Annex A (normative) Core profile

---



### A.1 General

This annex defines the X3D components which comprise the Core profile. This includes not only the nodes which shall be supported but also which fields in the supported nodes may be ignored.


This profile is targeted towards:

- Absolute minimal file definitions required by X3D,
- Building minimally defined scenes by explicitly specifying the component and levels required, and
- Allowing a broader range of implementations by eliminating some complexity of a complete X3D implementation.

### A.2 Topics

[Table A.1](#) provides links to the major topics in this annex.

**Table A.1 — Topics**

- |  |
|--|
| <ul style="list-style-type: none"><li><a href="#">A.1 General</a></li><li><a href="#">A.2 Topics</a></li><li><a href="#">A.3 Component support</a></li><li><a href="#">A.4 Conformance criteria</a></li><li><a href="#">A.5 Node set</a></li><li><a href="#">A.6 Other limitations</a></li><br/><li><a href="#">Table A.1 — Topics</a></li><li><a href="#">Table A.2 — Components and levels</a></li></ul> |
|--|
- 

- [Table A.3 — Nodes for conforming to the Core profile](#)
- [Table A.4 — Other limitations](#)

## A.3 Component support

[Table A.2](#) lists the components and their levels which shall be supported in the Core profile. Tables A.2 and A.3 describe limitations on required support for nodes and fields contained within these components.

**Table A.2 — Components and levels**

Component	Level	Reference
Core	1	<a href="#">7.5 Support levels</a>

## A.4 Conformance criteria

Conformance to this profile shall include conformance criteria defined by the specifications for those components and levels listed in [Table A.2](#).

In [Table A.3](#) and [Table A.4](#), the first column defines the item for which conformance is being defined. In some cases, general limits are defined but are later overridden in specific cases by more restrictive limits. The second column defines the requirements for a X3D file conforming to the Core profile; if a X3D file contains any items that exceed these limits, it may not be possible for a X3D browser conforming to the Core profile to successfully parse that X3D file. The third column defines the minimum complexity for a X3D scene that a X3D browser conforming to the Core profile shall be able to present to the user. Fields flagged as "not supported" may be supported by browsers which conform to the Core profile. The word "ignore" in the minimum browser support column refers only to the display of the item; in particular, *set\_events* to ignored *inputOutput* fields shall still generate corresponding *\_changed* events.

## A.5 Node set

[Table A.3](#) lists the nodes which shall be supported in the Core profile and specifies any fields in these nodes for which this profile requires less than full support.

**Table A.3 — Nodes for conforming to the Core profile**

Item	X3D File Limit	Minimum Browser Support
MetadataBoolean	No restrictions.	Full support.
MetadataDouble	No restrictions.	Full support.
MetadataFloat	No restrictions.	Full support.

MetadataInteger	No restrictions.	Full support.
MetadataSet	No restrictions.	Full support.
MetadataString	No restrictions.	Full support.

## A.6 Other limitations

[Table A.4](#) specifies other aspects of X3D functionality which are supported by this profile. Note that general items refer only to those specific nodes listed in [Table A.3](#).

**Table A.4 — Other limitations**

Item	X3D File Limit	Minimum Browser Support
All groups	500 children.	500 children. <i>bboxCenter</i> and <i>bboxSize</i> optionally supported.
All interpolators	1000 key-value pairs.	1000 key-value pairs.
All lights	8 simultaneous lights.	8 simultaneous lights.
Names for DEF/field	50 utf8 octets.	50 utf8 octets.
All <i>url</i> fields	10 URLs.	10 URLs. URN's optionally supported. Support `http`, `file`, and `ftp` protocols. Support relative URLs where relevant.
SFBool	No restrictions.	Full support.
SFColor	No restrictions.	Full support.
SFColorRGBA	No restrictions.	Full support.
SFDouble	Mp restrictions.	Full support. Range $\pm 1e\pm 12$ . Precision $1e-7$ .
SFFloat	No restrictions.	Full support.
SFImage	256 width. 256 height.	256 width. 256 height.
SFInt32	No restrictions.	Full support.
SFNode	No restrictions.	Full support.
SFRotation	No restrictions.	Full support.
SFString	30,000 utf8 octets.	30,000 utf8 octets.

SFTime	No restrictions.	Full support.
SFVec2d	15,000 values.	15,000 values.
SFVec2f	15,000 values.	15,000 values.
SFVec3d	15,000 values.	15,000 values.
SFVec3f	15,000 values.	15,000 values.
MFCColor	15,000 values.	15,000 values.
MFCColorRGBA	15,000 values.	15,000 values.
MFDouble	1000 values.	1000 values.
MFFloat	1,000 values.	1,000 values.
MFInt32	20,000 values.	20,000 values.
MFNode	500 values.	500 values.
MFRotation	1,000 values.	1,000 values.
MFString	30,000 utf8 octets per string, 10 strings.	30,000 utf8 octets per string, 10 strings.
MFTime	1,000 values.	1,000 values.
MFVec2d	15,000 values.	15,000 values.
MFVec2f	15,000 values.	15,000 values.
MFVec3d	15,000 values.	15,000 values.
MFVec3f	15,000 values.	15,000 values.





## Extensible 3D (X3D) Part 1: Architecture and base components

### Introduction

#### General

*Extensible 3D (X3D)* is a software standard for defining interactive web- and broadcast-based 3D content integrated with multimedia. X3D is intended for use on a variety of hardware devices and in a broad range of application areas such as engineering and scientific visualization, multimedia presentations, entertainment and educational titles, web pages, and shared virtual worlds. X3D is also intended to be a universal interchange format for integrated 3D graphics and multimedia. X3D is the successor to the Virtual Reality Modeling Language (VRML), the original ISO standard for web-based 3D graphics ([ISO/IEC 14772](#)). X3D improves upon VRML with new features, advanced application programmer interfaces, additional data encoding formats, stricter conformance, and a componentized architecture that allows for a modular approach to supporting the standard.

This section ~~provides a background to~~ describes the design objectives behind the development of X3D, and provides an overview of the features of X3D ~~and a description of the X3D specification process.~~

#### Design objectives

X3D has been developed to meet a specific set of market and technical requirements. To meet these requirements, X3D has adopted the following design objectives:

- Separate the runtime architecture from the data encoding
- Support a variety of encoding formats, including the Extensible Markup Language (XML)
- Add new graphical, ~~behavioral~~ behavioural and interactive objects
- Provide alternative application programmer interfaces (APIs) into the 3D scene
- Define subsets of the specification ("Profiles") that meet different market needs
- Allow for the specification to be implemented at varying levels of service
- Eliminate, where possible, unspecified or underspecified ~~behaviors~~ behaviours

#### X3D features



X3D has a rich set of features to support applications such as engineering and scientific visualization, multimedia presentations, entertainment and educational titles, web pages, and shared virtual worlds. The X3D feature set includes:

- **3D graphics** - Polygonal geometry, parametric geometry, hierarchical transformations, ~~lighting, materials and Physically Based Rendering (PBR)~~, **advanced materials and lighting for Physically Based Rendering (PBR)**, multi-pass/multi-stage texture mapping
- **2D graphics** - Text, 2D vector and planar shapes displayed within the 3D transformation hierarchy
- **Animation** - Timers and interpolator to drive continuous animations; humanoid animation and morphing
- **Humanoid Animation** - full-fidelity representations of human skeleton with motion animation
- **Metadata** - comprehensive inclusion of typed metadata sets
- **Spatialized audio and video** - **Audio generation and rendering**, audiovisual sources mapped onto geometry in the scene
- **User interaction** - Mouse-based picking and dragging; keyboard input
- **Navigation** - Cameras; user movement within the 3D scene; collision, proximity and visibility detection
- **User-defined objects** - Ability to extend built-in browser functionality by creating user-defined data types
- **Scripting** - Ability to dynamically change the scene via programming and scripting languages
- **Networking** - Ability to compose a single X3D scene out of assets located on a network; hyperlinking of objects to other scenes or assets located on the World Wide Web; **improved control of loading, refresh rates and security**
- **Physical simulation** - Humanoid animation; geospatial datasets; integration with Distributed Interactive Simulation (DIS) protocols
- **Geospatial positioning** - Ability to accurately position X3D scene objects geospatially.
- **CAD geometry** – ability to represent CAD models mapped from CAD systems.
- **Layering** – Ability to organize X3D scenes into rendering groups so that objects in each layer can overlay objects in underlying layers.
- **Support for programmable shaders** – Ability to replace the X3D lighting model with custom shader programs.
- **Particle systems** – Ability to generate systems of particles that can represent fire, smoke, and other such effects.
- **Volume rendering** – Ability to specify and render volumetric data sets, as used within medical imaging, for example.

For a complete list of X3D features, consult the component descriptions in clauses 7 through ~~40~~ **42** of this part of ISO/IEC 19775.







## Extensible 3D (X3D) Part 1: Architecture and base components

### 2 Normative references



The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

The [Bibliography](#) contains a list of informative documents and technology.

\* \* , version ,

Identifier	Reference
<b>I639</b>	<a href="#">ISO 639</a> , <i>Codes for the representation of names of languages:</i> ISO 639-1:2002, <i>Part 1: Alpha-2 code</i> ISO 639-2:1998, <i>Part 2: Alpha-3 code.</i>
<b>I3166</b>	<a href="#">ISO 3166</a> , <i>Codes for the representation of names of countries and their subdivisions:</i> ISO 3166-1, <i>Part 1: Country codes</i> ISO 3166-2, <i>Part 2: Country subdivision code</i> ISO 3166-3, <i>Part 3: Code for formerly used names of countries.</i>
<b>I8632</b>	<a href="#">ISO/IEC 8632</a> , <i>Information technology — Computer graphics — Metafile for the storage and transfer of picture description information:</i> ISO/IEC 8632-1:1999, <i>Part 1: Functional specification</i> ISO/IEC 8632-3:1999, <i>Part 3: Binary encoding</i> ISO/IEC 8632-4:1999, <i>Part 4: Clear text encoding.</i>
<b>I8859-1</b>	<a href="#">ISO/IEC 8859-1:1998</a> , <i>Information technology — 8-bit single-byte coded graphic character sets — Part 1: Latin alphabet No. 1.</i>
<b>I9899</b>	<a href="#">ISO/IEC 9899:1999</a> , <i>Programming languages — C.</i>
<b>I9973</b>	<a href="#">ISO/IEC 9973:2006</a> , <i>Information technology — Computer graphics, image processing and environmental representation — Procedures for registration of items.</i>
<b>I10641</b>	<a href="#">ISO/IEC 10641:1993</a> , <i>Information technology — Computer graphics and image processing — Conformance testing of implementations of graphics standards.</i>

<b>I10646</b>	<a href="#">ISO/IEC 10646:2003</a> , <i>Information technology — Universal Multiple-Octet Coded Character Set (UCS)</i> .
<b>I11172-1</b>	<a href="#">ISO/IEC 11172-1:1993</a> , <i>Information technology — Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s — Part 1: Systems</i> .
<b>I14496-1</b>	<a href="#">ISO/IEC 14496-1:2001</a> , <i>Coding of audio-visual objects — Part 1: Systems</i> .
<b>I14772-1</b>	<a href="#">ISO/IEC 14772-1:1997</a> , <i>Information technology — Computer graphics and image processing — The Virtual reality modeling language (VRML) — Part 1: Functional specification and UTF-8 encoding</i> .
<b>I15948</b>	<a href="#">ISO/IEC 15948:2004</a> , <i>Information technology — Computer graphics — PNG (Portable Network Graphics): Functional specification</i> .
<b>I16262</b>	<a href="#">ISO/IEC 16262:2002</a> , <i>Information technology — ECMAScript language specification</i> .
<b>I18026</b>	<a href="#">ISO/IEC 18026:2006</a> , <i>Information technology — Spatial Reference Model (SRM)</i> .
<b>I19774</b>	<a href="#">ISO/IEC 19774:2019</a> <del>19774:2006</del> , <i>Information technology — Computer graphics and image processing — Humanoid Animation (H-Anim) Parts 1 and 2</i> .
<b>I19775-2</b>	<a href="#">ISO/IEC 19775-2</a> , <del>Information technology — Computer graphics and image processing — Part 2: Scene access interface</del> <i>Information technology — Computer graphics, image processing and environmental data representation — Extensible 3D (X3D) — Part 2: Scene access interface (SAI)</i> .
<b>I19776</b>	<a href="#">ISO/IEC 19776</a> , <i>Information technology — <del>Computer graphics and image processing</del> Computer graphics, image processing and environmental data representation — Extensible 3D (X3D) encodings</i> . ISO/IEC 19776-1, <i>Part 1: Extensible Markup Language (XML) encoding</i> ISO/IEC 19776-2, <i>Part 2: Classic VRML encoding</i> ISO/IEC 19776-3, <i>Part 3: Compressed binary encoding</i>
<b>I19777</b>	<a href="#">ISO/IEC 19777</a> , <i>Information technology — Computer graphics and image processing — Extensible 3D (X3D) language bindings</i> ISO/IEC 19777-1, <i>Part 1: ECMAScript</i> ISO/IEC 19777-2, <i>Part 2: Java</i>
<b>I80000</b>	<a href="#">ISO 80000</a> , <i>Quantities and Units</i> ISO 80000-1:2009, <i>Part 1: General</i> ISO 80000-2:2009, <i>Part 2: Mathematical signs and symbols to be used in the natural sciences and technology</i> ISO 80000-3:2006, <i>Part 3: Space and time</i> ISO 80000-4:2006, <i>Part 4: Mechanics</i> ISO 80000-5:2007, <i>Part 5: Thermodynamics</i> ISO 80000-6:2008, <i>Part 6: Electromagnetism</i> ISO 80000-7:2008, <i>Part 7: Light</i> ISO 80000-8:2007, <i>Part 8: Acoustics</i> ISO 80000-9:2009, <i>Part 9: Physical chemistry and molecular physics</i>

	<p>ISO 80000-10:2009, <i>Part 10: Atomic and nuclear physics</i>  ISO 80000-11:2008, <i>Part 11: Characteristic numbers</i>  ISO 80000-12:2009, <i>Part 12: Solid state physics</i>  ISO 80000-13:2008, <i>Part 13: Information science and technology</i>  ISO 80000-14:2008, <i>Part 14: Telebiometrics related to human physiology</i></p>
<b>IEEE1278</b>	<p><a href="#">IEEE Standard 1278.1-1995</a>, <i>Standard for Distributed Interactive Simulation — Application Protocols, 1995.</i>  <a href="#">IEEE Standard 1278.1a-1998</a>, <i>Supplement to Standard for Distributed Interactive Simulation — Application Protocols, 1998.</i>  <a href="#">IEEE Standard 1278.2-1995</a>, <i>Supplement to Standard for Distributed Interactive Simulation — Communication Services and Profiles, 1995.</i>  <a href="#">IEEE Standard 1278.3-1996</a>, <i>Recommended Practice for Distributed Interactive Simulation — Exercise Management and Feedback, 1996.</i>  <a href="#">IEEE Standard 1278.4-1997</a>, <i>Trial-Use Recommended Practice for Distributed Interactive Simulation — Verification, Validation, and Accreditation, 1997.</i></p>
<b>DICOM</b>	<p><i>The DICOM Standard, Digital Imaging and Communications in Medicine</i>, Rosslyn, VA, 2003.  <a href="https://www.nema.org">https://www.nema.org</a></p>
<b>JAVA</b>	<p><i>The Java Language Specification, Third Edition</i> by James Gosling, Bill Joy, Guy Steele and Gilad Bracha, Addison Wesley, Reading Massachusetts, 2005, ISBN 0-321-24678-0.  <i>The Java™ Virtual Machine Specification, Second Edition</i> by Tim Lindhold and Frank Yellin, Addison Wesley, Reading Massachusetts, 1999, ISBN 0-201-43294-3.</p>
<b>JPEG</b>	<p><i>JPEG File Interchange Format, JFIF, Version 1.02</i>, 1992.  <a href="http://www.w3.org/Graphics/JPEG/jfif.txt">http://www.w3.org/Graphics/JPEG/jfif.txt</a>  <a href="#">ISO/IEC 10918-1:1994</a>, <i>Information technology — Digital compression and coding of continuous-tone still images: Requirements and guidelines.</i></p>
<b>GLTF</b>	<p><i>GL Transmission Format (glTF) Specification</i>, The Khronos Group, Version 2.0, 2017.  <a href="https://github.com/KhronosGroup/glTF/tree/master/specification/2.0">https://github.com/KhronosGroup/glTF/tree/master/specification/2.0</a></p>
<b>MIDI</b>	<p><i>Complete MIDI 1.0 Detailed Specification v96.1 (second edition)</i>, MIDI Manufacturers Association, P.O. Box 3173, La Habra, CA 90632-3173 USA, 2001.  <a href="http://www.midi.org">http://www.midi.org</a></p>
<b>REG</b>	<p><i>ISO International Register of Graphical Items</i>, Maintenance agencies and registration authorities.  <a href="http://www.iso.org/iso/standards_development/maintenance_agencies.htm">http://www.iso.org/iso/standards_development/maintenance_agencies.htm</a></p>
<b>RFC1738</b>	<p><a href="#">IETF RFC 1738</a>, <i>Uniform Resource Locators (URL)</i>.</p>
<b>RFC1766</b>	<p><a href="#">IETF RFC 1766</a>, <i>Tags for the Identification of Languages, Internet standards track protocol.</i></p>
<b>RFC1808</b>	<p><a href="#">IETF RFC 1808</a>, <i>Relative Uniform Resource Locators</i>.</p>
<b>RFC1889</b>	<p><a href="#">IETF RFC 1889</a>, <i>RTP: A Transport Protocol for Real-Time Applications.</i></p>

<b>RFC2077</b>	<a href="#">IETF RFC 2077</a> , <i>The Model Primary Content Type for Multipurpose Internet Mail Extensions.</i>
<b>RFC2141</b>	<a href="#">IETF RFC 2141</a> , <i>URN Syntax.</i>
<b>RFC2397</b>	<a href="#">IETF RFC 2397</a> , <i>The "data" URL scheme.</i>
<b>RFC3066</b>	<a href="#">IETF RFC 3066</a> , <i>Tags for the Identification of Languages.</i>
<b>RFC3541</b>	<a href="#">IETF RFC 3541</a> , <i>A Uniform Resource Name (URN) Namespace for the Web3D Consortium (Web3D).</i>
<b>RFC7231</b>	<a href="#">IETF RFC 7231</a> , <i>Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content.</i>
<b>W3C-HTML5</b>	<a href="#">Hypertext Markup Language (HTML) 5.2</a> , <i>World Wide Web Consortium (W3C) Recommendation, 14 December 2017.</i>
<b>W3C-WebAudio</b>	<a href="#">Web Audio API</a> , <i>World Wide Web Consortium (W3C) Candidate Recommendation, 11 June 2020.</i>





## Extensible 3D (X3D) Part 1: Architecture and base components

### 23 Navigation component

---



#### 23.1 Introduction

##### 23.1.1 Name

The name of this component is "Navigation". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 23.1.2 Overview

This clause describes the Navigation component of this part of ISO/IEC 19775. [Table 23.1](#) provides links to the major topics in this clause.

**Table 23.1 — Topics**

- [23.1 Introduction](#)
  - [23.1.1 Name](#)
  - [23.1.2 Overview](#)
- [23.2 Concepts](#)
  - [23.2.1 An overview of navigation](#)
  - [23.2.2 Navigation paradigms](#)
  - [23.2.3 Viewing model](#)
  - [23.2.4 Collision detection and terrain following](#)
  - [23.2.5 Viewpoint list](#)
- [23.3 Abstract types](#)
  - [23.3.1 X3DViewpointNode](#)
- [23.4 Node reference](#)
  - [23.4.1 Billboard](#)
  - [23.4.2 Collision](#)
  - [23.4.3 LOD](#)
  - [23.4.4 NavigationInfo](#)
  - [23.4.5 OrthoViewpoint](#)
  - [23.4.6 Viewpoint](#)
  - [23.4.7 ViewpointGroup](#)



- [23.5 Support levels](#)
- [Table 23.1 — Topics](#)
- [Table 23.2 — Navigation component support levels](#)

## 23.2 Concepts

### 23.2.1 An overview of navigation

Navigation is the capability of users to interact with the X3D browser using one or more input devices to affect the view it presents. Navigation support is not required for all profiles.

Every X3D scene can be thought of as containing a *viewpoint* from which the objects in the scene are presented to the viewer. Navigation permits the user to change the position and orientation of the viewpoint with respect to the remainder of the scene thereby enabling the user to move through the scene and examine objects in the scene.

The [NavigationInfo](#) node (see [23.4.4 NavigationInfo](#)) specifies the characteristics of the desired navigation behaviour, but the exact user interface is browser-dependent. Nodes derived from [X3DViewpointNode](#) (see [23.3.1 X3DViewpointNode](#)) specify key locations and orientations in the world to which the user may be moved via SAI services or browser-specific user interfaces.

### 23.2.2 Navigation paradigms

The browser may allow the user to modify the location and orientation of the viewer in the virtual world using a navigation paradigm. Many different navigation paradigms are possible, depending on the nature of the virtual world and the task the user wishes to perform. For instance, a walking paradigm would be appropriate in an architectural walkthrough application, while a flying paradigm might be better in an application exploring interstellar space. Examination is another common use for X3D, where the scene contains one or more objects which the user wishes to view from many angles and distances.

The [NavigationInfo](#) node has a *type* field that specifies the navigation paradigm for this world. The actual user interface provided to accomplish this navigation is browser-dependent. See [23.4.4 NavigationInfo](#), for details.

### 23.2.3 Viewing model

The browser controls the location and orientation of the viewer in the world, based on input from the user (using the browser-provided navigation paradigm) and the motion of the currently bound [X3DViewpointNode](#) node (and its coordinate system). The X3D author can place any number of viewpoints in the world at important places from which the user might wish to view the world. Each viewpoint is described by an [X3DViewpointNode](#) node. Viewpoint nodes exist in their parent's coordinate system, and both the viewpoint and the coordinate system may be changed to affect the view of the world presented by the browser. Only one viewpoint is bound at a time. A detailed

description of how *X3DViewpointNode* nodes operate is described in [7.2.2 Bindable children nodes](#) and [23.3.1 X3DViewpointNode](#).

Navigation is performed relative to the viewpoint's location and does not affect the location and orientation values of an *X3DViewpointNode* node. The location of the viewer may be determined with a [ProximitySensor](#) node (see [22.4.1 ProximitySensor](#)).

This part of ISO/IEC 19775 specifies two node types derived from *X3DViewpointNode*. The [Viewpoint](#) node specifies a perspective viewpoint while the [OrthoViewpoint](#) node specifies an orthographic viewpoint.

### 23.2.4 Collision detection and terrain following

In profiles in which collision detection is required, the [NavigationInfo](#) types of `WALK`, `FLY`, and `NONE` shall strictly support collision detection between the user's avatar and other objects in the scene by prohibiting navigation and/or adjusting the position of the viewpoint. However, the [NavigationInfo](#) types `ANY` and `EXAMINE` may temporarily disable collision detection during navigation, but shall not disable collision detection during the normal execution of the world. See [23.4.4 NavigationInfo](#), for details on the various navigation types.

Collision nodes can be used to generate events when viewer and objects collide, and can be used to designate that certain objects should be treated as not being subject to collision detection and should not be recognized as terrain for navigation modes that require terrain following to be supported. Browser support for inter-object collision is not specified.

[NavigationInfo](#) nodes can be used to specify certain parameters often used by browser navigation paradigms. The size and shape of the viewer's avatar determines how close the avatar may be to an object before a collision is considered to take place. These parameters can also be used to implement *terrain following* by keeping the avatar a certain distance above the ground. They can additionally be used to determine how short an object must be for the viewer to automatically step up onto it instead of colliding with it.

### 23.2.5 Viewpoint list

The viewpoint list is an optional browser-provided feature that lists currently available viewpoints for user information and selection.

Viewpoints are listed in the order corresponding to the extended scene graph. Thus viewpoints contained in [Inline](#) nodes and nodes that are instances of prototypes are loaded in the order defined by the scene, even if load time delays are different from scene-specified order. This has no effect on specification-defined eligibility for first bound viewpoint. Viewpoints that are removed from the scene are no longer eligible for the viewpoint list.

Selecting a viewpoint from a viewpoint list will first unbind the current viewpoint before binding the selected viewpoint. When *retainUserOffsets* is `FALSE`, the viewer is returned to the originally defined viewpoint position/orientation after local navigation. Such a return to the defined viewpoint can occur either by reselection of current viewpoint from

the viewpoint list, or else by using the PgUp key (as defined in [Annex G.2 Select from multiple viewpoints](#)).

## 23.3 Abstract types

### 23.3.1 X3DViewpointNode

```
X3DViewpointNode : X3DBindableNode {
  SFBool [in] set_bind
  SFVec3f/d [in,out] centerOfRotation 0 0 0 (-∞,∞)
  SFString [in,out] description ""
  SFFloat [in,out] farClippingPlane -1 -1 or (0,∞)
  SFBool [in,out] jump TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFNode [in,out] navigationInfo NULL [NavigationInfo]
  SFFloat [in,out] nearClippingPlane -1 -1 or (0,∞)
  SFRotation [in,out] orientation 0 0 1 0 (-∞,∞)
  SFVec3f/d [in,out] position 0 0 1 0 (-∞,∞)
  SFBool [in,out] retainUserOffsets FALSE
  SFBool [in,out] viewAll FALSE
  SFTime [out] bindTime
  SFBool [out] isBound
}
```

A node of node type *X3DViewpointNode* defines a specific location in the local coordinate system from which the user may view the scene. *X3DViewpointNode* nodes are bindable children nodes (see [7.2.2 Bindable children nodes](#)) and thus there exists an *X3DViewpointNode* stack in the browser in which the top-most *X3DViewpointNode* node on the stack is the currently active *X3DViewpointNode* node. If a `TRUE` value is sent to the `set_bind` field of an *X3DViewpointNode* node, it is moved to the top of the *X3DViewpointNode* node stack and activated. When an *X3DViewpointNode* node is at the top of the stack, the user's view is conceptually re-parented as a child of the *X3DViewpointNode* node. All subsequent changes to the *X3DViewpointNode* node's coordinate system change the user's view (e.g., changes to any ancestor transformation nodes or to the *X3DViewpointNode* node's `position` or `orientation` fields). Sending a `set_bind FALSE` event removes the *X3DViewpointNode* node from the stack and produces `isBound FALSE` and `bindTime` events. If the popped *X3DViewpointNode* node is at the top of the *X3DViewpointNode* stack, the user's view is re-parented to the next entry in the stack. More details on binding stacks can be found in [7.2.2 Bindable children nodes](#). When an *X3DViewpointNode* node is moved to the top of the stack, the existing top of stack *X3DViewpointNode* node sends an `isBound FALSE` event and is pushed down the stack.

An author can automatically move the user's view through the world by binding the user to either an *X3DViewpointNode* node and then animating either the *X3DViewpointNode* node or the transformations above it. Browsers shall allow the user view to be navigated relative to the coordinate system defined by the *X3DViewpointNode* node (and the transformations above it) even if the *X3DViewpointNode* node or its ancestors' transformations are being animated.

The `bindTime` field sends the time at which the *X3DViewpointNode* node is bound or unbound. This can happen:

- during loading;
- when a `set_bind` event is sent to the *X3DViewpointNode* node;
- when the browser binds to the *X3DViewpointNode* node through its user interface described below.

The *position* and *orientation* fields of the *X3DViewpointNode* node specify relative locations in the local coordinate system. *Position* is relative to the coordinate system's origin (0,0,0), while *orientation* specifies a rotation relative to the default orientation. In the default position and orientation, the viewer is on the Z-axis looking down the  $-Z$ -axis toward the origin with  $+X$  to the right and  $+Y$  straight up. *X3DViewpointNode* nodes are affected by the transformation hierarchy.

Navigation types (see [23.4.4 NavigationInfo](#)) that require a definition of a down vector (e.g., terrain following) shall use the negative Y-axis of the coordinate system of the currently bound *X3DViewpointNode* node. Likewise, navigation types that require a definition of an up vector shall use the positive Y-axis of the coordinate system of the currently bound *X3DViewpointNode* node. The *orientation* field of the *X3DViewpointNode* node does not affect the definition of the down or up vectors. This allows the author to separate the viewing direction from the gravity direction.

The *jump* field specifies whether the user's view "jumps" to the position and orientation of a bound *X3DViewpointNode* node or remains unchanged. This jump is instantaneous and discontinuous in that no collisions are performed and no [ProximitySensor](#) nodes are checked in between the starting and ending jump points. If the user's position before the jump is inside a [ProximitySensor](#) the *exitTime* of that sensor shall send the same timestamp as the bind field. Similarly, if the user's position after the jump is inside a [ProximitySensor](#) the *enterTime* of that sensor shall send the same timestamp as the bind field. Regardless of the value of *jump* at bind time, the relative viewing transformation between the user's view and the current *X3DViewpointNode* node shall be stored with the current *X3DViewpointNode* node for later use when *un-jumping* (i.e., popping the *X3DViewpointNode* binding stack from an *X3DViewpointNode* node with *jump* `TRUE`). The following summarizes the bind stack rules (see [7.2.2 Bindable children nodes](#)) with additional rules regarding *X3DViewpointNode* nodes (displayed in boldface type):

- d. During read, the first encountered *X3DViewpointNode* node is bound by pushing it to the top of the *X3DViewpointNode* node stack. If an *X3DViewpointNode* node name is specified in the URL that is being read, this named *X3DViewpointNode* node is considered to be the first encountered *X3DViewpointNode* node. Nodes contained within [Inline](#) nodes (see [9.4.2 Inline](#)), within the strings passed to the `Browser.createX3DFromString()` method, or within files passed to the `Browser.createX3DFromURL()` method (see [2.\[19775-2\]](#)) are not candidates for the first encountered *X3DViewpointNode* node. The first node within a prototype instance is a valid candidate for the first encountered *X3DViewpointNode* node. The first encountered *X3DViewpointNode* node sends an `isBound` `TRUE` event.
- e. When a `set_bind` `TRUE` event is received by an *X3DViewpointNode* node,
  1. If it is not on the top of the stack: **The relative transformation from the current top of stack *X3DViewpointNode* node to the user's view is stored with the current top of stack *X3DViewpointNode* node.** The current top of stack node sends an `isBound` `FALSE` event. The new node is moved to the top of the stack and becomes the currently bound *X3DViewpointNode* node. The new *X3DViewpointNode* node (top of stack) sends an `isBound` `TRUE` event. **If *jump* is `TRUE` for the new *X3DViewpointNode* node, the user's view is instantaneously "jumped" to match the values in the *position* and *orientation* fields of the new *X3DViewpointNode* node.**



2. If the node is already at the top of the stack, this event has no affect.
- f. When a *set\_bind* `FALSE` event is received by an *X3DViewpointNode* node in the stack, it is removed from the stack. If it was on the top of the stack,
  1. it sends an *isBound* `FALSE` event,
  2. the next node in the stack becomes the currently bound *X3DViewpointNode* node (*i.e.*, *pop*) and issues an *isBound* `TRUE` event,
  3. **if its *jump* field value is `TRUE`, the user's view is instantaneously "jumped" to the *position* and *orientation* of the next *X3DViewpointNode* node in the stack with the stored relative transformation of this next *X3DViewpointNode* node applied.**
- g. If a *set\_bind* `FALSE` event is received by a node not in the stack, the event is ignored and *isBound* events are not sent.
- h. When a node replaces another node at the top of the stack, the *isBound* `TRUE` and `FALSE` events from the two nodes are sent simultaneously (*i.e.*, with identical timestamps).
- i. If a bound node is deleted, it behaves as if it received a *set\_bind* `FALSE` event (see c. above).

The *jump* field may change after an *X3DViewpointNode* node is bound. The rules described above still apply. If *jump* was `TRUE` when the *X3DViewpointNode* node is bound, but changed to `FALSE` before the *set\_bind* `FALSE` is sent, the *X3DViewpointNode* node does not *un-jump* during unbind. If *jump* was `FALSE` when the *X3DViewpointNode* node is bound, but changed to `TRUE` before the *set\_bind* `FALSE` is sent, the *X3DViewpointNode* node does perform the *un-jump* during unbind.

Note that there are two other mechanisms that result in the binding of a new *X3DViewpointNode*:

- j. An [Anchor](#) node's *url* field specifies a "#X3DViewpointNodeName".
- k. A script invokes the `loadURL()` method and the URL argument specifies a "#X3DViewpointNodeName".

Both of these mechanisms override the *jump* field value of the specified *X3DViewpointNode* node (#X3DViewpointNodeName) and assume that *jump* is `TRUE` when binding to the new *X3DViewpointNode*. The behaviour of the viewer transition to the newly bound *X3DViewpointNode* depends on the currently bound [NavigationInfo](#) node's *type* field value (see [23.4.4 NavigationInfo](#)).

The *fieldOfView* field specifies a preferred minimum viewing angle from this *X3DViewpointNode* in angle base units. A small field of view roughly corresponds to a telephoto lens; a large field of view roughly corresponds to a wide-angle lens. The field of view shall be greater than zero and smaller than  $\pi$ . The value of *fieldOfView* represents the minimum viewing angle in any direction axis perpendicular to the view. For example, a browser with a rectangular viewing projection shall have the following relationship:

$$\frac{\text{display width}}{\text{display height}} = \frac{\tan(\text{FOV}_{\text{horizontal}}/2)}{\tan(\text{FOV}_{\text{vertical}}/2)}$$

where the smaller of display width or display height determines which angle equals the *fieldOfView* (the larger angle is computed using the relationship described above). The larger angle shall not exceed  $\pi$  and may force the smaller angle to be less than *fieldOfView* in order to sustain the aspect ratio.

The *description* field specifies a textual description of the *X3DViewpointNode* node. This may be used by browser-specific user interfaces. If an *X3DViewpointNode*'s *description* field is empty it is recommended that the browser not present this *X3DViewpointNode* in its browser-specific user interface.

The *centerOfRotation* field specifies a center about which to rotate the user's eyepoint when in `EXAMINE` mode. If the browser does not provide the ability to spin around the object in `EXAMINE` mode, or `LOOKAT` is not in the list of allowed navigation modes, this field shall be ignored. For additional information, see [23.4.4 NavigationInfo](#) and [22.4.1 ProximitySensor](#).

The URL syntax ".../scene.wrl#X3DViewpointNodeName" specifies the user's initial view when loading "scene.wrl" to be the first *X3DViewpointNode* node in the X3D file that appears as `DEF X3DViewpointNodeName X3DViewpointNode {...}`. This overrides the first *X3DViewpointNode* node in the X3D file as the initial user view, and a `set_bind TRUE` message is sent to the *X3DViewpointNode* node named "X3DViewpointNodeName". If the *X3DViewpointNode* node named "X3DViewpointNodeName" is not found, the browser shall use the first *X3DViewpointNode* node in the X3D file (*i.e.*, the normal default behaviour). The URL syntax "#X3DViewpointNodeName" (*i.e.*, no file name) specifies an *X3DViewpointNode* within the existing X3D file. If this URL is loaded (*e.g.*, Anchor node's *url* field or `loadURL()` method is invoked by a Script node), the *X3DViewpointNode* node named "X3DViewpointNodeName" is bound (a `set_bind TRUE` event is sent to this *X3DViewpointNode* node).

The *retainUserOffsets* field indicates whether a viewpoint needs to retain (`TRUE`) or reset to zero (`FALSE`) any prior user navigation offsets from defined viewpoint position, orientation. When an node of type *X3DViewpointNode* is bound, user navigation offsets are reinitialized if the associated *retainUserOffsets* is `TRUE`.

The *navigationInfo* field defines a dedicated *NavigationInfo* node for this *X3DViewpointNode*. The specified *NavigationInfo* node receives a `set_bind TRUE` event at the time when the parent node is bound and receives a `set_bind FALSE` at the time when the parent node is unbound.

If specified and positive, the values specified for *nearClippingPlane* and *farClippingPlane* define the near and far clipping plane distances when the *X3DViewpointNode* is bound. Otherwise these values are defined by the bound *NavigationInfo* node, including when the *X3DViewpointNode* is unbound.

If *nearClippingPlane* is defined, it shall be less than the defined *farClippingPlane* (if provided) or the corresponding *visibilityLimit* value defined by *NavigationInfo*. If *farClippingPlane* is defined, it shall be greater than the defined *nearClippingPlane* (if provided) or the corresponding value defined by *NavigationInfo*.

A default value of -1 for *nearClippingPlane* or *farClippingPlane* means that the field has no effect on currently active view-frustum boundaries.

Each type of viewpoint defines the specific actions associated with the *viewAll* field.

## 23.4 Node reference

### 23.4.1 Billboard

```

Billboard : X3DGroupingNode {
  MFNode [in]  addChildren      [X3DChildNode]
  MFNode [in]  removeChildren  [X3DChildNode]
  SFVec3f [in,out] axisOfRotation 0 1 0 (-∞,∞)
  MFNode [in,out] children      [] [X3DChildNode]
  SFBool [in out] bboxDisplay  FALSE
  SFNode [in,out] metadata      NULL [X3DMetadataObject]
  SFBool [in out] visible       TRUE
  SFVec3f []  bboxCenter        0 0 0 (-∞,∞)
  SFVec3f []  bboxSize          -1 -1 -1 [0,∞) or -1 -1 -1
}

```

The Billboard node is a grouping node that transforms the coordinate system of its children so that the local Z-axis of the children turns to point at the viewer within the limitations of its rotational axis.

The *axisOfRotation* field specifies which axis to use to perform the rotation. This axis is defined in the local coordinate system.

When the *axisOfRotation* field is not (0, 0, 0), the following steps describe how to rotate the billboard to face the viewer:

- Compute the vector from the Billboard node's origin to the viewer's position. This vector is called the *billboard-to-viewer* vector.
- Compute the plane defined by the *axisOfRotation* and the billboard-to-viewer vector.
- Rotate the local Z-axis of the billboard into the plane from b., pivoting around the *axisOfRotation*.

When the *axisOfRotation* field is set to (0, 0, 0), the special case of *viewer-alignment* is indicated. In this case, the object rotates to keep the billboard's local Y-axis parallel with the Y-axis of the viewer. This special case is distinguished by setting the *axisOfRotation* to (0, 0, 0). The following steps describe how to align the billboard's Y-axis to the Y-axis of the viewer:

- Compute the billboard-to-viewer vector.
- Rotate the Z-axis of the billboard to be collinear with the billboard-to-viewer vector and pointing towards the viewer's position.
- Rotate the Y-axis of the billboard to be parallel and oriented in the same direction as the Y-axis of the viewer.

If the *axisOfRotation* and the billboard-to-viewer line are coincident, the plane cannot be established and the resulting rotation of the billboard is undefined. For example, if the *axisOfRotation* is set to (0,1,0) (Y-axis) and the viewer flies over the billboard and peers directly down the Y-axis, the results are undefined.

Multiple instances of Billboard nodes (DEF/USE) operate as expected: each instance rotates in its unique coordinate system to face the viewer.

[10.2.1 Grouping and children node types](#) provides a description of the *children*,



*addChildren*, and *removeChildren* fields.

The *bboxCenter* and *bboxSize* fields specify a bounding box that encloses the Billboard node's children. This is a hint that may be used for optimization purposes. The results are undefined if the specified bounding box is smaller than the actual bounding box of the children at any time. A default *bboxSize* value, (-1, -1, -1), implies that the bounding box is not specified and if needed shall be calculated by the browser. A description of the *bboxCenter* and *bboxSize* fields is contained in [10.2.2 Bounding boxes](#).

## 23.4.2 Collision

```
Collision : X3DGroupingNode, X3DSensorNode {
  MFNode [in]  addChildren    [X3DChildNode]
  MFNode [in]  removeChildren [X3DChildNode]
  MFNode [in,out] children    [] [X3DChildNode]
  SFFBool [in,out] bboxDisplay FALSE
  SFFBool [in,out] enabled    TRUE
  SFNode [in,out] metadata    NULL [X3DMetadataObject]
  SFFBool [in,out] visible    TRUE
  SFTime [out]  collideTime
  SFFBool [out] isActive
  SFVec3f []   bboxCenter    0 0 0 (-∞,∞)
  SFVec3f []   bboxSize      -1 -1 -1 [0,∞) or -1 -1 -1
  SFNode []    proxy         NULL [X3DChildNode]
}
```

The Collision node is a grouping node that specifies the collision detection properties for its children (and their descendants), specifies surrogate objects that replace its children during collision detection, and sends events signalling that a collision has occurred between the avatar and the Collision node's geometry or surrogate. By default, all geometric nodes in the scene are collidable with the viewer except [IndexedLineSet](#) and [PointSet](#). Browsers shall detect geometric collisions between the avatar (see [23.3.4 NavigationInfo](#)) and the scene's geometry and prevent the avatar from "entering" the geometry. See [23.2.4 Collision detection and terrain following](#) for general information on collision detection.

If there are no Collision nodes specified in a X3D file, browsers shall detect collisions between the avatar and all objects during navigation.

[10.2.1 Grouping and children node types](#) contains a description of the *children*, *addChildren*, and *removeChildren* fields.

The Collision node's *enabled* field enables and disables collision detection as well as terrain following when the navigation type requires it. If *enabled* is set to `FALSE`, the children and all descendants of the Collision node shall not be checked for collision or terrain, even though they are drawn. This includes any descendent Collision nodes that have *enabled* set to `TRUE` (*i.e.*, setting *enabled* to `FALSE` turns collision off for every child node below it).

The value of the *isActive* field indicates the current state of the Collision node. An *isActive* `TRUE` event is generated when a collision occurs. An *isActive* `FALSE` event is generated when a collision no longer occurs.

Collision nodes with the *enabled* field set to `TRUE` detect the nearest collision with their descendent geometry (or proxies). When the nearest collision is detected, the collided Collision node sends the time of the collision through its *collideTime* field. If a Collision node contains a child, descendant, or proxy (see below) that is a Collision node, and

both Collision nodes detect that a collision has occurred, both send a *collideTime* event at the same time. A *collideTime* event shall be generated if the avatar is colliding with collidable geometry when the Collision node is read from a X3D file or inserted into the transformation hierarchy.

The *bboxCenter* and *bboxSize* fields specify a bounding box that encloses the Collision node's children. This is a hint that may be used for optimization purposes. The results are undefined if the specified bounding box is smaller than the actual bounding box of the children at any time. A default *bboxSize* value, (-1, -1, -1), implies that the bounding box is not specified and if needed shall be calculated by the browser. More details on the *bboxCenter* and *bboxSize* fields can be found in [10.2.2 Bounding boxes](#).

The collision proxy, defined in the *proxy* field, is any legal children node as described in [10.2.1 Grouping and children node types](#) that is used as a substitute for the Collision node's children during collision detection. The proxy is used strictly for collision detection; it is not drawn.

If the value of the *enabled* field is `TRUE` and the *proxy* field is non-NULL, the *proxy* field defines the scene on which collision detection is performed. If the *proxy* value is `NULL`, collision detection is performed against the *children* of the Collision node.

If *proxy* is specified, any descendent children of the Collision node are ignored during collision detection. If *children* is empty, *enabled* is `TRUE`, and *proxy* is specified, collision detection is performed against the proxy but nothing is displayed. In this manner, invisible collision objects may be supported.

The *collideTime* field generates an event specifying the time when the avatar (see [23.3.4 NavigationInfo](#)) makes contact with the collidable children or proxy of the Collision node. An ideal implementation computes the exact time of collision. Implementations may approximate the ideal by sampling the positions of collidable objects and the user. The [NavigationInfo](#) node contains additional information for parameters that control the avatar size.

### 23.4.3 LOD

```

LOD : X3DGroupingNode {
  MFNode [in]  addChildren      [X3DChildNode]
  MFNode [in]  removeChildren  [X3DChildNode]
  MFNode [in,out] children      [] [X3DChildNode]
  SFBool [in,out] bboxDisplay  FALSE
  SFNode [in,out] metadata     NULL [X3DMetadataObject]
  SFBool [in,out] visible      TRUE
  SFInt32 [out] level_changed
  SFVec3f []  bboxCenter      0 0 0 (-∞,∞)
  SFVec3f []  bboxSize        -1 -1 -1 [0,∞) or -1 -1 -1
  SFVec3f []  center           0 0 0 (-∞,∞)
  SFBool []   forceTransitions FALSE
  MFFloat []  range            [] [0,∞) or -1
}

```

The LOD node specifies various levels of detail or complexity for a given object, and provides hints allowing browsers to automatically choose the appropriate version of the object based on the distance from the user. The *children* field contains a list of nodes that represent the same object or objects at varying levels of detail, ordered from highest level of detail to the lowest level of detail.

The *range* field specifies the ideal distances at which to switch between the levels. The *forceTransitions* field specifies whether browsers are allowed to disregard level

distances in order to provide better performance. A *forceTransitions* value of `TRUE` specifies that every transition should be performed regardless of any internal optimizations that might be available. A *forceTransitions* value of `FALSE` specifies that browsers are allowed to disregard level distances in order to provide better performance.

[10.2.1 Grouping and children node types](#) contains details on the types of nodes that are legal values for *children*.

The *center* field is a translation offset in the local coordinate system that specifies the centre of the LOD node for distance calculations.

The number of nodes in the *children* field shall exceed the number of values in the *range* field by one (*i.e.*,  $N+1$  *children* nodes for  $N$  *range* values). The *range* field contains monotonic increasing values that shall be greater than zero. In order to calculate which level to display, first the distance is calculated from the viewer's location, transformed into the local coordinate system of the LOD node (including any scaling transformations), to the *center* point of the LOD node. Then, the LOD node evaluates the step function  $L(d)$  to choose a level for a given value of  $d$  (where  $d$  is the distance from the viewer position to the centre of the LOD node).

Let  $n$  ranges,  $R_0, R_1, R_2, \dots, R_{n-1}$ , partition the domain  $(0, +infinity)$  into  $n+1$  subintervals given by  $(0, R_0), [R_0, R_1), \dots, [R_{n-1}, +infinity)$ . Also, let  $n$  levels  $L_0, L_1, L_2, \dots, L_{n-1}$  be the values of the step function  $L(d)$ . The level,  $L(d)$ , for a given distance  $d$  is defined as follows:

$$\begin{aligned} L(d) &= L_0, & \text{if } d < R_0, \\ &= L_{i+1}, & \text{if } R_i \leq d < R_{i+1}, \text{ for } -1 < i < n-1, \\ &= L_{n-1}, & \text{if } d \geq R_{n-1}. \end{aligned}$$

The  $L(d)^{th}$  node of the *children* field is that which is displayed. The  $L(d)^{th}$  node of the *children* field (denoted by  $L_i$  in the equation above) is that which is displayed. When  $L(d)$  is activated for display, the LOD node generates a *level\_changed* event with value  $i$  where the value of  $i$  identifies which value of  $L$  was activated for display.

Specifying too few levels will result in the last level being used repeatedly for the lowest levels of detail. If more levels than ranges are specified, the extra levels are ignored. An empty range field is an exception to this rule. This case is a hint to the browser that it may choose a level automatically to maintain a constant display rate. Each value in the *range* field shall be greater than the previous value.

LOD nodes are evaluated top-down in the scene graph. Only the descendants of the currently selected *children* node are rendered. All nodes under an LOD node continue to receive and send events regardless of which LOD node's *level* is active.

**EXAMPLE** If an active [TimeSensor](#) node is contained within an inactive level of an LOD node, the TimeSensor node sends events regardless of the LOD node's state.

The *bboxCenter* and *bboxSize* fields specify a bounding box that encloses the LOD node's children. This is a hint that may be used for optimization purposes. The results are undefined if the specified bounding box is smaller than the actual bounding box of

the child with the largest bounding box at any time. A default *bboxSize* value,  $(-1, -1, -1)$ , implies that the bounding box is not specified and, if needed, is calculated by the browser. A description of the *bboxCenter* and *bboxSize* fields is contained in [10.2.2 Bounding boxes](#).

## 23.4.4 NavigationInfo

```
NavigationInfo : X3DBindableNode {
  SFFBool [in] set_bind
  MFFloat [in,out] avatarSize [0.25 1.6 0.75] [0,∞)
  SFFBool [in,out] headlight TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFFloat [in,out] speed 1.0 [0,∞)
  SFTime [in,out] transitionTime 1.0 [0, ∞)
  MFString [in,out] transitionType ["LINEAR"] ["TELEPORT","LINEAR",
  "ANIMATE",...]
  MFString [in,out] type ["EXAMINE" "ANY"] ["ANY","WALK","EXAMINE","FLY",
  "LOOKAT","NONE","EXPLORE",...]
  SFFloat [in,out] visibilityLimit 0.0 [0,∞)
  SFTime [out] bindTime
  SFFBool [out] isBound
  SFFBool [out] transitionComplete
}
```

The *NavigationInfo* node contains information describing the physical characteristics of the viewer's avatar and viewing model. *NavigationInfo* node is a bindable node (see [7.2.2 Bindable children nodes](#)). Whenever the current *X3DViewpointNode* node changes, the current *NavigationInfo* node shall be re-parented to it by the browser. Whenever the current *NavigationInfo* node changes, the new *NavigationInfo* node shall be re-parented to the current *Viewpoint* node by the browser.

If a `TRUE` value is sent to the *set\_bind* field of a *NavigationInfo* node, the node is pushed onto the top of the *NavigationInfo* node stack. When a *NavigationInfo* node is bound, the browser uses the fields of the *NavigationInfo* node to set the navigation controls of its user interface and the *NavigationInfo* node is conceptually re-parented under the currently bound *X3DViewpointNode* node. All subsequent scaling changes to the current *X3DViewpointNode* node's coordinate system automatically change aspects (see below) of the *NavigationInfo* node values used in the browser (*e.g.*, scale changes to any ancestors' transformations). A `FALSE` value sent to *set\_bind* pops the *NavigationInfo* node from the stack, results in an *isBound FALSE* event, and pops to the next entry in the stack which shall be re-parented to the current *X3DViewpointNode* node. [7.2.2 Bindable children nodes](#) has more details on binding stacks.

The *type* field specifies an ordered list of navigation paradigms that specify a combination of navigation types and the initial navigation type. The navigation type of the currently bound *NavigationInfo* node determines the user interface capabilities of the browser. For example, if the currently bound *NavigationInfo* node's *type* is "WALK", the browser shall present a "WALK" navigation user interface paradigm (see below for description of WALK). Browsers shall recognize at least the following navigation types: "ANY", "WALK", "EXAMINE", "FLY", "LOOKAT", and "NONE" with support as specified in [Table 23.2](#).

If "ANY" does not appear in the *type* field list of the currently bound *NavigationInfo*, the browser's navigation user interface shall be restricted to the recognized navigation types specified in the list. In this case, browsers shall not present a user interface that allows the navigation type to be changed to a type not specified in the list. However, if any one of the values in the *type* field are "ANY", the browser may provide any type of navigation interface, and allow the user to change the navigation type dynamically. Furthermore, the first recognized type in the list shall be the initial navigation type presented by the browser's user interface.

"ANY" navigation specifies that the browser may choose the navigation paradigm that best suits the content and provide a user interface to allow the user to change the navigation paradigm dynamically. The results are undefined if the currently bound `NavigationInfo`'s *type* value is "ANY" and Viewpoint transitions (see [23.3.5 Viewpoint](#)) are triggered by the Anchor node (see [9.4.1 Anchor](#)) or the `loadURL()` scripting method (see [Part 2 of ISO/IEC 19775](#)).

"WALK" navigation is used for exploring a virtual world on foot or in a vehicle that rests on or hovers above the ground. It is strongly recommended that `WALK` navigation define the up vector in the +Y direction and provide some form of terrain following and gravity in order to produce a walking or driving experience. If the bound `NavigationInfo`'s *type* is "WALK", the browser shall strictly support collision detection (see [23.3.2 Collision](#)).

"FLY" navigation is similar to `WALK` except that terrain following and gravity may be disabled or ignored. There shall still be some notion of "up" however. If the bound `NavigationInfo`'s *type* is "FLY", the browser shall strictly support collision detection (see [23.3.2 Collision](#)).

"LOOKAT" navigation is used to explore a scene by navigating to a particular object. Selecting an object with "LOOKAT":

- a. Moves the viewpoint directly to some convenient viewing distance from the bounding box center of the selected object, with the viewpoint orientation set to aim the view at the approximate centre of the object;
- b. Sets the center of rotation in the currently bound Viewpoint node to the approximate centre of the selected object.

"EXAMINE" navigation is used for viewing individual objects. "EXAMINE" shall provide the ability to orbit or spin the user's eyepoint about the center of rotation in response to user actions. The center of rotation for moving the viewpoint around the object and determining the viewpoint orientation is specified in the currently bound `X3DViewpointNode` node (see [23.3.1 X3DViewpointNode](#)). The browser shall strictly support collision detection (see [23.4.2 Collision](#)) and shall trigger exit and enter events throughout `EXAMINE` operations.

"LOOKAT" navigation in combination with "EXAMINE" is used to explore a scene by navigating to a particular object, then being able to conveniently navigate in order to examine the object from different orientations. If content specifies both "LOOKAT" and "EXAMINE" types, any "LOOKAT" operations shall change the center of rotation for subsequent "EXAMINE" operations.

"NONE" navigation disables and removes all browser-specific navigation user interface forcing the user to navigate using only mechanisms provided in the scene, such as Anchor nodes or scripts that include `loadURL()`. "NONE" has an effect only when it is the first supported navigation type. If "NONE" is not the first supported navigation type, it has no effect.

"EXPLORE" navigation is used to provide consistent keystroke navigation for both geospatial and Cartesian modes. When "EXPLORE" mode is active:

- a. Dragging left and right while holding the left button down causes viewpoint



- rotation about a vertical axis that passes through the point of rotation. This vertical axis is always perpendicular to the viewpoint vector. Motion in the left direction rotates the viewpoint clockwise (as viewed from the top) about the vertical axis. Rotation is tied to the motion of the pointing device; there is no damping or delay.
- b. Dragging the up and down while holding the left button down causes rotation about a horizontal axis that passes through the point of rotation. Motion in the up direction rotates the viewpoint clockwise (as viewed from the right) about the horizontal axis. Rotation is tied to the motion of the pointing device; there is no damping or delay.
  - c. Holding the Ctrl key (or other key that may be user-selectable) down modifies the left button down drag movement such that up and down (Y-axis) movement causes the viewpoint to zoom toward and from the point of rotation. Left and right motion while Ctrl is held down has no effect. Shift and Ctrl (or other keys that may be user-selectable) held at the same time also enables zoom but disables TouchSensors.
  - d. Holding the Alt key (or other key that may be user-selectable) modifies the movement such that motion of the pointing device while the left button is held down is translated into a pan of the viewpoint in a plane passing through the viewpoint perpendicular to the vector pointing to the point of rotation. Shift and Alt (or other keys that may be user-selectable) held at the same time also enables pan but disables TouchSensors.
  - e. The point of rotation can be set by holding the Shift key (or other key that may be user-selectable) while pointing at an object and clicking the left button. To provide feedback that the point has been selected, the viewpoint shall zoom about twenty percent of the distance toward that point.
  - f. If the pointer is positioned over a TouchSensor, the pointer icon shall change its appearance to indicate that a left click will activate the TouchSensor.
  - g. Holding the Shift key (or other key that may be user-selectable) overrides any TouchSensor that the pointer may be over and forces the pointing device to function as the viewpoint navigation tool; *i.e.*, drag operations cause rotation, click operations cause center of rotation point selection.

Whether user-selectable alternatives to the Shift, Ctrl, and/or Alt are provided is browser-dependent. If provided, the method by which such alternatives are specified is also browser-dependent.

If the NavigationInfo type is "WALK", "FLY", "EXAMINE", or "NONE" or a combination of these types (*i.e.*, "ANY" is not in the list), *X3DViewpointNode* transitions (see [23.3.1 X3DViewpointNode](#)) triggered by the [Anchor](#) node (see [9.4.1 Anchor](#)) or the `loadURL()` scripting method (see [Part 2 of ISO/IEC 19775](#)) shall be implemented as a jump from the old *X3DViewpointNode* to the new *X3DViewpointNode* with transition effects that shall not trigger events besides the exit and enter events caused by the jump.

Browsers may create browser-specific navigation type extensions. It is recommended that extended *type* names include a unique suffix (*e.g.*, HELICOPTER\_mydomain.com) to prevent conflicts. *X3DViewpointNode* transitions (see [23.3.5 Viewpoint](#)) triggered by the [Anchor](#) node (see [9.4.1 Anchor](#)) or the `loadURL()` scripting method (see [Part 2 of ISO/IEC 19775](#)) are undefined for extended navigation types. If none of the types are recognized by the browser, the default "ANY" is used. These strings values are case

sensitive ("any" is not equal to "ANY").

The *transitionType* field specifies an ordered list of paradigms that determine the manner in which the browser moves the viewer when a new Viewpoint node is bound. Browsers shall recognize and support at least the following transition types: "TELEPORT", "LINEAR", and "ANIMATE". For value "TELEPORT", the transition shall be immediate without any intervening positions. For value "LINEAR", the browser shall perform a linear interpolation of the position and orientation values. For value "ANIMATE", the browser shall perform a browser-specific animation effect. If all values are unrecognized or the field is empty, the default value of "LINEAR" shall be used. This field applies to any transitions between positions and orientations including Viewpoint bindings and "LOOKAT" navigation type.

The *transitionTime* field specifies the duration of any viewpoint transition. The transition starts when the next Viewpoint node is bound. The duration of the transition depends on the value of the *transitionType* field. If *transitionType* is "TELEPORT", the transition is instantaneous and completes at the same time it starts. A transition type of "LINEAR" indicates that the transition lasts the number of seconds specified by the first value in the *transitionTime* field. If *transitionType* is "ANIMATE", *transitionTime* provides browser-dependent parameters to the browsers viewpoint animation engine. When a transition completes, a *transitionComplete* event is signaled.

The *speed* field specifies the rate at which the viewer travels through a scene in speed base units. Since browsers may provide mechanisms to travel faster or slower, this field specifies the default, average speed of the viewer when the NavigationInfo node is bound. If the NavigationInfo *type* is "EXAMINE", *speed* shall not affect the viewer's rotational speed. Scaling in the transformation hierarchy of the currently bound Viewpoint node (see above) scales the *speed*; parent translation and rotation transformations have no effect on *speed*. Speed shall be non-negative. Zero speed indicates that the avatar's position is stationary, but its orientation and field of view may still change. If the navigation *type* is "NONE", the *speed* field has no effect.

The *avatarSize* field specifies the user's physical dimensions in the world for the purpose of collision detection and terrain following. It is a multi-value field allowing several dimensions to be specified. The first value shall be the allowable distance between the user's position and any collision geometry (as specified by a [Collision](#) node) before a collision is detected. The second shall be the height above the terrain at which the browser shall maintain the viewer. The third shall be the height of the tallest object over which the viewer can move. This allows staircases to be built with dimensions that can be ascended by viewers in all browsers. The transformation hierarchy of the currently bound Viewpoint node scales the *avatarSize*. Translations and rotations have no effect on *avatarSize*.

For purposes of terrain following, the browser maintains a notion of the *down* direction (down vector), since gravity is applied in the direction of the down vector. This down vector shall be along the negative Y-axis in the local coordinate system of the currently bound *X3DViewpointNode* node (*i.e.*, the accumulation of the *X3DViewpointNode* node's ancestors' transformations, not including the *X3DViewpointNode* node's *orientation* field).

Geometry beyond the *visibilityLimit* may not be rendered. A value of 0.0 indicates an infinite *visibilityLimit*. The *visibilityLimit* field is restricted to be greater than or equal to

zero.

The *speed*, *avatarSize* and *visibilityLimit* values are all scaled by the transformation being applied to the currently bound *X3DViewpointNode* node. If there is no currently bound *X3DViewpointNode* node, the values are interpreted in the world coordinate system. This allows these values to be automatically adjusted when binding to a *X3DViewpointNode* node that has a scaling transformation applied to it without requiring a new *NavigationInfo* node to be bound as well. The results are undefined if the scale applied to the *X3DViewpointNode* node is non-uniform.

The *headlight* field specifies whether a browser shall turn on a headlight. A headlight is a directional light that always points in the direction the user is looking. Setting this field to `TRUE` allows the browser to provide a headlight, possibly with user interface controls to turn it on and off. Scenes that enlist precomputed lighting (EXAMPLE radiosity solutions) can turn the headlight off. The headlight shall have *intensity* = 1, *color* = (1 1 1), *ambientIntensity* = 0.0, and *direction* = (0 0 -1).

It is recommended that the near clipping plane be set to one-half of the collision radius as specified in the *avatarSize* field (setting the near plane to this value prevents excessive clipping of objects just above the collision volume, and also provides a region inside the collision volume for content authors to include geometry intended to remain fixed relative to the viewer). Such geometry shall not be occluded by geometry outside of the collision volume.

## 23.4.5 OrthoViewpoint

```
OrthoViewpoint : X3DViewpointNode {
  SFBool [in] set_bind
  SFVec3f [in,out] centerOfRotation 0 0 0 (-∞,∞)
  SFString [in,out] description ""
  SFFloat [in,out] farClippingPlane -1 -1 or (0,∞)
  MFFloat [in,out] fieldOfView -1, -1, 1, 1 (-∞,∞)
  SFBool [in,out] jump TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFNode [in,out] navigationInfo NULL [NavigationInfo]
  SFFloat [in,out] nearClippingPlane -1 -1 or (0,∞)
  SFRotation [in,out] orientation 0 0 1 0 [-1,1],(-∞,∞)
  SFVec3f [in,out] position 0 0 10 (-∞,∞)
  SFBool [in,out] retainUserOffsets FALSE
  SFBool [in,out] viewAll FALSE
  SFTime [out] bindTime
  SFBool [out] isBound
}
```

The *OrthoViewpoint* node defines a viewpoint that provides an orthographic view of the scene. An orthographic view is one in which all projectors are parallel to the projector from *centerOfRotation* to *position*.

The *fieldOfView* field specifies minimum and maximum extents of the view in units of the local coordinate system. A small field of view roughly corresponds to a telephoto lens; a large field of view roughly corresponds to a wide-angle lens. The minimum and maximum values in each direction of the field of view shall have the relationship  $\text{minimum} < \text{maximum}$ . The value of *fieldOfView* represents the minimum viewing extent in any direction axis perpendicular to the view.

A browser with a rectangular viewing projection has the following relationship:

$$\frac{\text{display width}}{\text{display height}} = \frac{(\text{maximum}_x - \text{minimum}_x)}{(\text{maximum}_y - \text{minimum}_y)}$$



When the *viewAll* field is set to `TRUE` or a viewpoint is bound with *viewAll* field `TRUE`, the current view is modified to change the *centerOfRotation* field to match center of bounding box for entire visible scene, and the *orientation* field is modified to aim at that point. Zoom in or out until outside the bounding box for all models. Finally, the *fieldOfView* field is modified to encompass the visibility of all geometry in the bounding box for the entire scene. If the current view is within a model, any intervening geometry does not block the change in position. No collision detection or proximity sensing occurs when zooming. If needed, near and far clipping planes shall be adjusted to allow viewing the entire scene. When the value of the *viewAll* field is changed from `TRUE` to `FALSE`, no change in the current view occurs.

## 23.4.6 Viewpoint

```
Viewpoint : X3DViewpointNode {
  SFBool [in] set_bind
  SFVec3f [in,out] centerOfRotation 0 0 0 (-∞,∞)
  SFString [in,out] description ""
  SFFloat [in,out] farClippingPlane -1 -1 or (0,∞)
  SFFloat [in,out] fieldOfView π/4 (0,π)
  SFBool [in,out] jump TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFNode [in,out] navigationInfo NULL [NavigationInfo]
  SFFloat [in,out] nearClippingPlane -1 -1 or (0,∞)
  SFRotation [in,out] orientation 0 0 1 0 [-1,1],(-∞,∞)
  SFVec3f [in,out] position 0 0 10 (-∞,∞)
  SFBool [in,out] retainUserOffsets FALSE
  SFBool [in,out] viewAll FALSE
  SFTime [out] bindTime
  SFBool [out] isBound
}
```

The Viewpoint node defines a viewpoint that provides a perspective view of the scene. A perspective view is one in which all projectors coalesce at *position*.

The *fieldOfView* field specifies a preferred minimum viewing angle from this viewpoint in angle base units. A small field of view roughly corresponds to a telephoto lens; a large field of view roughly corresponds to a wide-angle lens. The field of view shall be greater than zero and smaller than  $\pi$ . The value of *fieldOfView* represents the minimum viewing angle in any direction axis perpendicular to the view.

A browser with a rectangular viewing projection has the following relationship:

$$\frac{\text{display width}}{\text{display height}} = \frac{\tan(\text{FOV}_{\text{horizontal}}/2)}{\tan(\text{FOV}_{\text{vertical}}/2)}$$

where the smaller of display width or display height determines which angle equals the *fieldOfView* (the larger angle is computed using the relationship described above). The larger angle shall not exceed  $\pi$  and may force the smaller angle to be less than *fieldOfView* in order to sustain the aspect ratio.

When the *viewAll* field is set to `TRUE` or a viewpoint is bound with *viewAll* field `TRUE`, the current view is modified to change the *centerOfRotation* field to match center of bounding box for entire visible scene, and the *orientation* field is modified to aim at that point. Finally, zoom position in or out until the bounding box containing the entire scene is fully within the current viewing window. If the current view is within a model, any intervening geometry does not block the change in position. No collision detection or proximity sensing occurs when zooming. If needed, near and far clipping planes shall be adjusted to allow viewing the entire scene. When the value of the *viewAll* field is changed from `TRUE` to `FALSE`, no change in the current view occurs.

## 23.4.7 ViewpointGroup

```
ViewpointGroup : X3DChildNode {
  SFVec3f [in,out] center      0 0 0 (-∞,∞)
  MFNode [in,out] children    NULL [X3DViewpointNode | ViewpointGroup]
  SFString [in,out] description ""
  SFBool [in,out] displayed   TRUE
  SFNode [in,out] metadata    NULL [X3DMetadataObject]
  SFBool [in,out] retainUserOffsets FALSE
  SFVec3f [in,out] size       0 0 0 (-∞,∞)
}
```

The `ViewpointGroup` node is used to control display of viewpoints on the viewpoint list. Use of `ViewpointGroup` allows a viewpoint list to have a tree structure, similar to a bookmark list.

The *children* field is a sequence of nodes of type [X3DViewpointNode](#).

The *description* field provides a simple description or navigation hint to be displayed for this `ViewpointGroup`.

The *displayed* field determines whether this `ViewpointGroup` is displayed in the current viewpoint list.

The *center* and *size* fields are defined identically as the corresponding [ProximitySensor](#) definitions. The *center* field provides a position offset from origin of local coordinate system. The *size* field provides the size of a proximity box within which the `ViewpointGroup` is usable and displayed on the viewpoint list. A *size* field of 0 0 0 specifies that the `ViewpointGroup` is always usable and displayable.

The *retainUserOffsets* field specifies whether the user is returned to the originally defined viewpoint position/orientation after local navigation (see [23.2.5 Viewpoint list](#)).

## 23.5 Support levels

The Navigation component provides two levels of support as specified in [Table 23.2](#).

**Table 23.2— Navigation component support levels**

Level	Prerequisites	Nodes	Support
1	Core 1		
		<i>X3DViewpointNode</i>	n/a
		NavigationInfo	avatarSize optionally supported. speed optionally supported. type support for at least "ANY", "FLY", "EXAMINE", and "NONE". visibilityLimit optionally supported.
			fieldOfView optionally supported. description optionally

		Viewpoint	supported. <i>retainUserOffsets</i> optionally supported. All other fields fully supported.
<b>2</b>	Core 1 Grouping 1 Shape 1 Environmental sensor 2		
		All Level 1 Navigation nodes	All fields fully supported.
		NavigationInfo	<i>type</i> support for at least "ANY", "FLY", "EXAMINE", "WALK", "LOOKAT", and "NONE". All other fields fully supported.
		Billboard	All fields fully supported.
		Collision	All fields fully supported.
		LOD	All fields fully supported.
<b>3</b>	Core 1 Grouping 1 Shape 1 Environmental sensor 2		
		All Level 2 Navigation nodes	All fields fully supported.
		OrthoViewpoint	All fields fully supported.
		ViewpointGroup	All fields fully supported.





# Extensible 3D (X3D) Part 1: Architecture and base components

## Annex B

(normative)

### Interchange profile

---



#### B.1 General

This annex defines the X3D components that comprise the Interchange profile. This includes not only the nodes that shall be supported but also which fields in the supported nodes may be ignored.

This profile is targeted towards:

- Exchange of geometry and animations between authoring systems,
- Possible implementation in a low-footprint engine requiring no interaction (EXAMPLE an applet or small browser plug-in),
- Addressing the limitations of software renders not capable of dealing with all details of the full X3D lighting model, and
- Allowing a broader range of implementations by eliminating some complexity of a complete X3D implementation.

#### B.2 Topics

[Table B.1](#) provides links to the major topics in this annex.

**Table B.1 — Topics**

- |  |
|--|
| <ul style="list-style-type: none"><li>• <a href="#">B.1 General</a></li><li>• <a href="#">B.2 Topics in this annex</a></li><li>• <a href="#">B.3 Component support</a></li><li>• <a href="#">B.4 Conformance criteria</a></li><li>• <a href="#">B.5 Node set</a></li><li>• <a href="#">B.6 Other limitations</a></li></ul> |
|--|

- [Table B.1 — Topics](#)
- [Table B.2 — Components and levels](#)
- [Table B.3 — Nodes for conforming to the Interchange profile](#)
- [Table B.4 — Other limitations](#)

## B.3 Component support

[Table B.2](#) lists the components and their levels which shall be supported in the Interchange profile. Tables B.2 and B.3 describe limitations on required support for nodes and fields contained within these components.

**Table B.2 — Components and levels**

Component	Level	Reference
Core	1	<a href="#">7.5 Support levels</a>
Time	1	<a href="#">8.5 Support levels</a>
Networking	1	<a href="#">9.5 Support levels</a>
Grouping	1	<a href="#">10.5 Support levels</a>
Rendering	3	<a href="#">11.5 Support levels</a>
Shape	1	<a href="#">12.5 Support levels</a>
Geometry3D	2	<a href="#">13.4 Support levels</a>
Lighting	1	<a href="#">17.5 Support levels</a>
Texturing	2	<a href="#">18.5 Support levels</a>
Interpolation	2	<a href="#">19.5 Support levels</a>
Navigation	1	<a href="#">23.4 Support levels</a>
Environmental effects	1	<a href="#">24.5 Support levels</a>

## B.4 Conformance criteria

Conformance to this profile shall include conformance criteria defined by the specifications for those components and levels listed in [Table B.2](#).

In Tables B.3 and B.4, the first column defines the item for which conformance is being defined. In some cases, general limits are defined but are later overridden in specific cases by more restrictive limits. The second column defines the requirements for a X3D file conforming to the Interchange profile; if a X3D file contains any items that exceed these limits, it may not be possible for a X3D browser conforming to the Interchange

profile to successfully parse that X3D file. The third column defines the minimum complexity for a X3D scene that a X3D browser conforming to the Interchange profile shall be able to present to the user. Fields flagged as "not supported" may be supported by browsers which conform to the Interchange profile. The word "ignore" in the minimum browser support column refers only to the display of the item; in particular, *set\_* events to ignored inputOutput fields shall still generate corresponding *\_changed* events.

## B.5 Node set

[Table B.3](#) lists the nodes which shall be supported in the Interchange profile and specifies any fields in these nodes for which this profile requires less than full support.

**Table B.3 — Nodes for conforming to the Interchange profile**

Item	X3D File Limit	Minimum Browser Support
Appearance	No restrictions.	<i>textureTransform</i> optionally supported. <i>lineProperties</i> not supported. <i>fillProperties</i> not supported.
Background	No restrictions.	<i>groundAngle</i> and <i>groundColor</i> optionally supported. <i>backURL</i> , <i>frontURL</i> , <i>leftURL</i> , <i>rightURL</i> , <i>topURL</i> optionally supported. <i>skyAngle</i> optionally supported. One <i>skyColor</i> .
Box	No restrictions.	Full support.
Color	15,000 colours.	15,000 colours.
ColorInterpolator	Restrictions as for all interpolators.	Full support except as for all interpolators.
ColorRGBA	15,000 colours.	15,000 colours. Alpha component optionally supported.
Cone	No restrictions.	Full support.
Coordinate	65,535 points	65,535 points.
CoordinateInterpolator	15,000 coordinates per <i>keyValue</i> .	15,000 coordinates per <i>keyValue</i> .

	Restrictions as for all interpolators.	Support as for all interpolators.
Cylinder	No restrictions.	Full support.
DirectionalLight	No restrictions.	Not scoped by parent Group or Transform.
Group	Restrictions as for all groups.	<i>addChildren</i> optionally supported. <i>removeChildren</i> optionally supported. Otherwise as for all groups.
ImageTexture	JPEG ( <a href="#">2. [JPEG]</a> ) and PNG ( <a href="#">2. [115948]</a> ) format.	JPEG ( <a href="#">2. [JPEG]</a> ) and PNG ( <a href="#">2. [115948]</a> ) format.
IndexedFaceSet	10 vertices per face. 5000 faces. Less than 65,535 indices.	<i>ccw</i> optionally supported. <i>set_colorIndex</i> optionally supported. <i>set_normalIndex</i> optionally supported. <i>normal</i> optionally supported. Only convex indexed face sets supported. Hence, <i>convex</i> optionally supported. For <i>creaseAngle</i> , only 0 and $\pi$ radians supported (or the equivalent if a different angle base unit has been specified). 10 vertices per face. 5000 faces. 65,535 indices in any index field.  Face list shall be well-defined as follows: <ol style="list-style-type: none"> <li>1. Each face is terminated with -1, including the last face in the array.</li> <li>2. Each face contains at least three non-coincident vertices.</li> <li>3. A given <i>coordIndex</i> is not repeated in a face.</li> <li>4. The vertices of a face shall define a planar polygon.</li> <li>5. The vertices of a face shall not define a self-intersecting polygon.</li> </ol>
	15,000 total vertices.	

IndexedLineSet	15,000 indices in any index field.	15,000 total vertices. 15,000 indices in any index field.
IndexedTriangleFanSet	5,000 total faces. 15,000 indices in any index field.	5,000 total faces. 15,000 indices in any index field.
IndexedTriangleSet	5,000 total faces. 15,000 indices in any index field.	5,000 total faces. 15,000 indices in any index field.
IndexedTriangleStripSet	5,000 total faces. 15,000 indices in any index field.	5,000 total faces. 15,000 indices in any index field.
LineSet	15,000 total vertices.	15,000 total vertices.
Material	No restrictions.	<i>ambientIntensity</i> optionally supported. <i>shininess</i> optionally supported. <i>specularColor</i> optionally supported. A Material with <i>emissiveColor</i> not equal to (0,0,0), <i>diffuseColor</i> equal to (0,0,0) is an unlit material. One-bit transparency; transparency values $\geq 0.5$ transparent.
MetadataBoolean	No restrictions.	Full support.
MetadataDouble	No restrictions.	Full support.
MetadataFloat	No restrictions.	Full support.
MetadataInteger	No restrictions.	Full support.
MetadataSet	No restrictions.	Full support.
MetadataString	No	Full support.



	restrictions.	
MultiTexture	No restrictions.	At least one texture displayed per node with any number specified. Full support.
MultiTextureCoordinate	15,000 coordinates.	15,000 coordinates.
MultiTextureTransform	No restrictions.	At least one texture displayed per node with any number specified. Full support.
NavigationInfo	No restrictions.	<i>avatarSize</i> optionally supported. <i>speed</i> optionally supported. <i>type</i> optionally supported. <i>visibilityLimit</i> optionally supported.
Normal	15,000 normals	15,000 normals.
NormalInterpolator	15,000 normals	15,000 normals.
OrientationInterpolator	Restrictions as for all interpolators.	Full support except as for all interpolators.
PixelTexture	512 width. 512 height.	512 width. 512 height. Display fully transparent and fully opaque pixels.
PointSet	5,000 points.	5000 points.
PositionInterpolator	Restrictions as for all interpolators.	Full support except as for all interpolators.
ScalarInterpolator	Restrictions as for all interpolators.	Full support except as for all interpolators.
Shape	No restrictions.	Full support.
Sphere	No restrictions.	Full support.
TextureCoordinate	65,535 coordinates.	65,535 coordinates.
	No	

TextureCoordinateGenerator	restrictions.	Full support.
TextureTransform	No restrictions.	Full support.
TimeSensor	No restrictions.	<i>pause</i> , optionally supported. <i>isPaused</i> , optionally supported. <i>resumeTime</i> , optionally supported.
Transform	Restrictions as for all groups.	<i>addChildren</i> optionally supported. <i>removeChildren</i> optionally supported. Otherwise, full support except as for all groups.
TriangleFanSet	5,000 triangles per fan. 15,000 total triangles.	5,000 triangles per fan. 15,000 total triangles.
TriangleSet	15,000 triangles	15,000 triangles.
TriangleStripSet	5,000 triangles per strip. 15,000 total triangles	5,000 triangles per strip. 15,000 total triangles.
Viewpoint	No restrictions.	<i>fieldOfView</i> optionally supported. <i>description</i> optionally supported.
WorldInfo	No restrictions.	<i>info</i> , <i>title</i> Ignored.

## B.6 Other limitations

[Table B.4](#) specifies other aspects of X3D functionality which are supported by this profile. Note that general items refer only to those specific nodes listed in [Table B.3](#).

**Table B.4 — Other limitations**

Item	X3D File Limit	Minimum Browser Support
All groups	500 children.	500 children. Ignore <i>bboxCenter</i> and <i>bboxSize</i> .
All interpolators	1000 key-value pairs.	1000 key-value pairs.
All lights	8 simultaneous lights.	8 simultaneous lights.

Names for DEF/field	50 utf8 octets.	50 utf8 octets.
All <i>url</i> fields	10 URLs.	10 URLs. URN's ignored. Support `http`, `file`, and `ftp` protocols. Support relative URLs where relevant.
SFBool	No restrictions.	Full support.
SFColor	No restrictions.	Full support.
SFColorRGBA	No restrictions.	Full support.
SFDouble	Mp restrictions.	Full support. Range $\pm 1e\pm 12$ . Precision $1e-7$ .
SFFloat	No restrictions.	Full support.
SFImage	512 width. 512 height.	512 width. 512 height.
SFInt32	No restrictions.	Full support.
SFNode	No restrictions.	Full support.
SFRotation	No restrictions.	Full support.
SFString	30,000 utf8 octets.	30,000 utf8 octets.
SFTime	No restrictions.	Full support.
SFVec2d	15,000 values.	15,000 values.
SFVec2f	15,000 values.	15,000 values.
SFVec3d	15,000 values.	15,000 values.
SFVec3f	15,000 values.	15,000 values.
MFCColor	15,000 values.	15,000 values.
MFCColorRGBA	15,000 values.	15,000 values.
MFDouble	1000 values.	1000 values.
MFFloat	1,000 values.	1,000 values.
MFInt32	20,000 values.	20,000 values.
MFNode	500 values.	500 values.
MFRotation	1,000 values.	1,000 values.

MFString	30,000 utf8 octets per string, 10 strings.	30,000 utf8 octets per string, 10 strings.
MFTime	1,000 values.	1,000 values.
MFVec2d	15,000 values.	15,000 values.
MFVec2f	15,000 values.	15,000 values.
MFVec3d	15,000 values.	15,000 values.
MFVec3f	15,000 values.	15,000 values.





## Extensible 3D (X3D) Part 1: Architecture and base components

### 3 Definitions, acronyms, and abbreviations



#### 3.1 Definitions

For the purposes of this part of ISO/IEC 19775, the following definitions apply.

##### 3.1.1 **activate**

cause a [sensor node](#) to generate an "isActive" [event](#)

##### 3.1.2 **ancestor**

[node](#) which is an antecedent of another node in the [transformation hierarchy](#)

##### 3.1.3 **author**

person or agent that creates [X3D files](#)

##### 3.1.4 **authoring tool**

See [generator](#).

##### 3.1.5 **avatar**

abstract representation of the [user](#) in an X3D [world](#)

##### 3.1.6 **bearing**

straight line passing through the [pointer](#) location in the direction of the pointer

##### 3.1.7 **bindable node**

[node](#) that may have many [instances](#) in a [scene graph](#) but only one instance may be active at any instant of [time](#)

### 3.1.8

#### **browser**

computer program that interprets [X3D files](#), presents their content to a [user](#) on a [display device](#), and allows the user to interact with [worlds](#) defined by X3D files by means of a user interface

### 3.1.9

#### **browser extension**

[nodes](#) defined using the prototyping mechanism that are understood only by certain [browsers](#)

### 3.1.10

#### **built-in node**

[node](#) of a [type](#) explicitly defined in this part of ISO/IEC 19775

### 3.1.11

#### **callback**

function defined in a [scripting language](#) to which [events](#) are passed

### 3.1.12

#### **child**

instance of a [children node](#)

### 3.1.13

#### **children node**

one of a set of [node types](#), instances of which can be collected in a group to share specific properties dependent on the type of the [grouping node](#)

### 3.1.14

#### **client system**

computer system, attached to a [network](#), that relies on another computer (the server) for essential processing functions

### 3.1.15

#### **collision proxy**

[node](#) used as a substitute for all of a Collision node's children during collision detection

### 3.1.16

#### **colour model**

characterization of a colour space in terms of explicit parameters

### 3.1.17

#### **culling**

process of identifying [objects](#) or parts of objects which do not need to be processed

further by the [browser](#) in order to produce the desired view of a [world](#)

### 3.1.18

#### **descendant**

[node](#) which descends from another node in the [transformation hierarchy](#) (a [children node](#))

### 3.1.19

#### **display device**

graphics device on which X3D [worlds](#) may be rendered

### 3.1.20

#### **drag sensor**

[pointing device sensor](#) that causes [events](#) to be generated in response to sensor-dependent pointer motions

### 3.1.21

#### **environmental sensor**

sensor [node](#) that generates [events](#) based on the location of the viewpoint in the [world](#) or in relation to [objects](#) in the world

### 3.1.22

#### **event**

message sent from one [node](#) to another as defined by a [route](#)

### 3.1.23

#### **event cascade**

sequence of [events](#) initiated by a script or sensor event and propagated from [node](#) to node along one or more [routes](#) all of which are considered to have occurred simultaneously

### 3.1.24

#### **execution model**

rules governing how [events](#) are processed by [browsers](#) and scripts

### 3.1.25

#### **external prototype**

[prototype](#) defined in an external file and referenced by a [URL](#)

### 3.1.26

#### **field**

property or attribute of a [node](#)

### 3.1.27

#### **field name**

identifier of a [field](#)

### 3.1.28

#### **frame**

single rendering of a [world](#) on a [display device](#) or a single time-step in a simulation

### 3.1.29

#### **generator**

computer program which creates [X3D files](#)

### 3.1.30

#### **geometric property node**

[node](#) defining the properties of a specific geometry node

### 3.1.31

#### **geometry node**

[node](#) containing mathematical descriptions of points, lines, surfaces, text strings and solids

### 3.1.32

#### **grab**

receive [events](#) from activated [pointing devices](#)

### 3.1.33

#### **grouping node**

one of a set of [node types](#) which include a list of nodes, referred to as its [children nodes](#)

### 3.1.34

#### **host application**

client application with which the [browser](#) communicates using the SAI

### 3.1.35

#### **image**

two-dimensional (2D) rectangular array of pixel values

### 3.1.36

#### **immersive**

creating the illusion of being inside a computer-generated scene

### 3.1.37

#### **in-lining**

mechanism by which one [X3D file](#) is hierarchically included in another

### 3.1.38



## **instance**

the [node](#) created by an instantiation

### **3.1.39**

## **instantiation**

the creation of a [node](#) based on its [node type](#)

### **3.1.40**

## **interpolator node**

[node](#) that defines a piece-wise linear interpolation

### **3.1.41**

## **intranet**

private [network](#) that uses the same protocols and standards as the Internet

### **3.1.42**

## **level of detail**

amount of detail or complexity which is displayed at any particular [time](#) for any particular [object](#)

### **3.1.43**

## **line terminator**

linefeed character (0x0A) and/or carriage return character (0x0D)

### **3.1.44**

## **loop**

sequence of [events](#) which would result in a specific event generator sending more than one event with the same [timestamp](#)

### **3.1.45**

## **multimedia**

integrated presentation, typically on a computer, of content of various types, such as computer graphics, audio, and video

### **3.1.46**

## **network**

set of interconnected computers

### **3.1.47**

## **node**

fundamental component of a [scene graph](#)

### **3.1.48**

## **node type**

characteristic of each [node](#) that describes, in general, its particular semantics

### 3.1.49

#### **object**

collection of data and procedures, packaged according to the rules and syntax defined in this part of ISO/IEC 19775

Note: This term is usually synonymous with [node](#).

### 3.1.50

#### **order of preference**

order (specified by the user) in which a list of [field](#) values is processed by the [browser](#)

### 3.1.51

#### **panorama**

background texture that is placed behind all geometry in the scene and in front of the ground and sky

### 3.1.52

#### **parent**

[node](#) which is an instance of a [grouping node](#)

### 3.1.53

#### **pixel**

one element of an [image](#) specified as a matrix of colour elements

### 3.1.54

#### **pointer**

location and direction in the [virtual world](#) defined by the [pointing device](#) with which the [user](#) is currently interacting with the virtual world

### 3.1.55

#### **pointing device**

hardware device connected to the [user's](#) computer by which the user directly controls the location and direction of the [pointer](#)

### 3.1.56

#### **pointing device sensor**

sensor [node](#) that generates [events](#) based on [user](#) actions, such as [pointing device](#) motions or button activations

### 3.1.57

#### **polyline**

piecewise linear curve

### 3.1.58

#### **profile**

named collection of criteria for functionality and conformance that defines an implementable subset of a standard

### 3.1.59

#### **prototype**

definition of a new [node type](#) in terms of the [nodes](#) defined in this part of ISO/IEC 19775

### 3.1.60

#### **prototyping**

mechanism for extending the set of [node types](#) from within a [X3D file](#)

### 3.1.61

#### **route**

connection between a [node](#) generating an [event](#) and a node receiving the event

### 3.1.62

#### **scene graph**

ordered collection of [grouping nodes](#) and other nodes

### 3.1.63

#### **script**

set of procedural functions normally executed as part of an [event cascade](#)

### 3.1.64

#### **scripting**

process of creating or referring to a script

### 3.1.65

#### **scripting language**

system of syntactical and semantic constructs used to define and automate procedures and processes on a computer

### 3.1.66

#### **sensor node**

[node](#) that enables the [user](#) to interact with the [world](#) in the scene graph hierarchy

### 3.1.67

#### **separator character**

[UTF-8](#) character used to separate syntactical entities in an [X3D file](#)

### 3.1.68

#### **sibling**

[node](#) which shares a [parent](#) with other nodes

### 3.1.69

#### **simulation tick**

smallest time unit capable of being identified in a digital simulation of analog time

### 3.1.70

#### **special group node**

[grouping node](#) that exhibits special behaviour (e.g., Switch or LOD)

### 3.1.71

#### **texel**

[pixel](#) in an [image](#) used as a [texture](#)

### 3.1.72

#### **texture**

[image](#) used to create visual appearance effects when applied to [geometry nodes](#)

### 3.1.73

#### **texture coordinates**

set of coordinates used to map a texture to geometry

### 3.1.74

#### **time**

monotonically increasing value generated by a node

### 3.1.75

#### **timestamp**

that part of an [event](#) that describes the time the [event](#) occurred and that caused the message to be sent

### 3.1.76

#### **transformation hierarchy**

subset of the [scene graph](#) consisting of [nodes](#) that have well-defined coordinate systems

### 3.1.77

#### **transparency chunk**

section of a PNG file containing transparency information (derived from [ISO/IEC 15948](#))

### 3.1.78

#### **traverse**

process the [nodes](#) in a [scene graph](#) in the correct order

### 3.1.79

#### **user**

person or agent who uses and interacts with [X3D files](#) by means of a [browser](#)

### **3.1.80 viewer**

location, direction, and viewing angle in a [virtual world](#) that determines the portion of the virtual world presented by the [browser](#) to the [user](#)

### **3.1.81 virtual world**

See [world](#).

### **3.1.82 white space**

one or more consecutive occurrences of a [separator character](#)

### **3.1.83 world**

collection of one or more [X3D files](#) and other multimedia content that, when interpreted by an [X3D browser](#), presents an interactive experience to the [user](#) consistent with the [author's](#) intent

### **3.1.84 world coordinate space**

coordinate system in which each X3D [world](#) is defined

### **3.1.85 X3D browser**

See [browser](#).

### **3.1.86 X3D document server**

computer program that locates and transmits [X3D files](#) and supporting files in response to requests from [browsers](#)

### **3.1.87 X3D file**

set of X3D nodes and statements as defined in this part of ISO/IEC 19775

### **3.1.88 XY plane**

plane perpendicular to the Z-axis that passes through the point  $Z = 0.0$

### **3.1.89 YZ plane**

plane perpendicular to the X-axis that passes through the point  $X = 0.0$

### **3.1.90**

#### **ZX plane**

plane perpendicular to the Y-axis that passes through the point  $Y = 0.0$

## **3.2 Acronyms and abbreviations**

For the purposes of this part of ISO/IEC 19775, the following expansion of acronyms and abbreviations apply:

### **3.2.1**

#### **CAD**

Computer-Assisted Design

### **3.2.2**

#### **HSV**

Hue, Saturation, and Value colour model

[\[FOLEY\]](#)

### **3.2.3**

#### **JPEG**

Joint Photographic Experts Group

[\[ISO/IEC 10918-1\]](#)

### **3.2.4**

#### **MIDI**

Musical Instrument Digital Interface. A standard for digital music representation

[\[MIDI\]](#)

### **3.2.5**

#### **MIME**

Multipurpose Internet Mail Extension

[\[IETF RFC2077\]](#)

### **3.2.6**

#### **MPEG**

Moving Picture Experts Group

[\[ISO/IEC 11172-1\]](#)

### **3.2.7**

#### **PNG**

Portable Network Graphics. A specification for representing two-dimensional images in files

[\[ISO/IEC 15948\]](#)

### **3.2.8**

#### **RGB**

Red, Green, and Blue colour model

[\[FOLEY\]](#)

### **3.2.9**

#### **RURL**

Relative Uniform Resource Locator

[\[IETF RFC1808\]](#)

### **3.2.10**

#### **SAI**

Scene Access Interface

[\[ISO/IEC 19775-2\]](#)

### **3.2.11**

#### **UCS**

Universal multiple-octet coded Character Set

[\[ISO/IEC 10646\]](#)

### **3.2.12**

#### **URI**

Universal Resource Identifier

[\[IETF RFC1630\]](#)

### **3.2.13**

#### **URL**

Uniform Resource Locator

[\[IETF RFC1738\]](#)

### **3.2.14**

#### **URN**

Universal Resource Name

[\[IETF RFC2141\]](#)

### **3.2.15**

#### **UTF-8**

variable-length 8-bit Universal multiple-octet coded character set Transformation Format

[\[ISO/IEC 10646\]](#)







## Extensible 3D (X3D)

### Part 1: Architecture and base components

# 24 Environmental effects component

---

## 24.1 Introduction

### 24.1.1 Name

The name of this component is "EnvironmentalEffects". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

### 24.1.2 Overview

This clause describes the Environmental Effects component of this part of ISO/IEC 19775. Nodes in this component support the creation of realistic environmental effects such as panoramic backgrounds and fog. [Table 24.1](#) provides links to the major topics in this clause.

**Table 24.1 — Topics**

- |  |
|--|
| <ul style="list-style-type: none"> <li>• <a href="#">24.1 Introduction</a> <ul style="list-style-type: none"> <li>◦ <a href="#">24.1.1 Name</a></li> <li>◦ <a href="#">24.1.2 Overview</a></li> </ul> </li> <li>• <a href="#">24.2 Concepts</a> <ul style="list-style-type: none"> <li>◦ <a href="#">24.2.1 Backgrounds</a></li> <li>◦ <a href="#">24.2.2 Fog semantics</a> <ul style="list-style-type: none"> <li>▪ <a href="#">24.2.2.1 Overview</a></li> <li>▪ <a href="#">24.2.2.2 Global fog semantics</a></li> <li>▪ <a href="#">24.2.2.3 Local fog semantics</a></li> <li>▪ <a href="#">24.2.2.4 Local and bindable fog interaction</a></li> <li>▪ <a href="#">24.2.2.5 Fog colour calculation</a></li> </ul> </li> </ul> </li> <li>• <a href="#">24.3 Abstract types</a> <ul style="list-style-type: none"> <li>◦ <a href="#">24.3.1 X3DBackgroundNode</a></li> <li>◦ <a href="#">24.3.2 X3DFogObject</a></li> </ul> </li> <li>• <a href="#">24.4 Node reference</a> <ul style="list-style-type: none"> <li>◦ <a href="#">24.4.1 Background</a></li> </ul> </li> </ul> |
|--|

- [24.4.2 Fog](#)
- [24.4.3 FogCoordinate](#)
- [24.4.4 LocalFog](#)
- [24.4.5 TextureBackground](#)
- [24.5 Support levels](#)
- [Figure 24.1 — X3DBackgroundNode field relationships](#)
- [Table 24.1 — Topics](#)
- [Table 24.2 — Environmental effects component support levels](#)

## 24.2 Concepts

### 24.2.1 Backgrounds

Background nodes are used to specify a colour backdrop that simulates ground and sky, as well as a background texture, or *panorama*, that is placed behind all geometry in the scene and in front of the ground and sky. Background nodes are specified in the local coordinate system and are affected by the accumulated rotation of their ancestors as described below. X3D supports two kinds of background nodes: a simple background node that contains a set of *url* fields for specifying static image files that compose the backdrop (see [24.4.1 Background](#)), and a complex background node containing arbitrary X3DTexture nodes that compose the backdrop (see [24.4.3 TextureBackground](#)). Both types of background node descend from the **baseabstract node** type [X3DBackgroundNode](#). Applications should use the [Background](#) node for simplicity, and the [TextureBackground](#) node for more flexibility and additional features.

Background nodes are bindable nodes as described in [7.2.2 Bindable children nodes](#). There exists a Background stack, in which the top-most [X3DBackgroundNode](#) node on the stack is the currently active [X3DBackgroundNode](#). To move an [X3DBackgroundNode](#) node to the top of the stack, a `TRUE` value is sent to the `set_bind` field. Once active, the [X3DBackgroundNode](#) node is then bound to the browser's view. A `FALSE` value sent to `set_bind` removes the [X3DBackgroundNode](#) from the stack and unbinds it from the browser's view. .

The backdrop is conceptually a partial sphere (the ground) enclosed inside of a full sphere (the sky) in the local coordinate system with the viewer placed at the centre of the spheres. Both spheres have infinite radius and each is painted with concentric circles of interpolated colour perpendicular to the local Y-axis of the sphere. The [X3DBackgroundNode](#) node is subject to the accumulated rotations of its ancestors' transformations. Scaling and translation transformations are ignored. The sky sphere is always slightly farther away from the viewer than the ground partial sphere causing the ground to appear in front of the sky where they overlap.

The `skyColor` field specifies the colour of the sky at various angles on the sky sphere. Angles for `skyColor` are specified in angle base units. The following assumes that the angle base units are radians. The equivalent values apply if an angle base unit other than radians is specified. The first value of the `skyColor` field specifies the colour of the sky at 0.0 radians representing the zenith (*i.e.*, straight up from the viewer). The

*skyAngle* field specifies the angles from the zenith in which concentric circles of colour appear. The zenith of the sphere is implicitly defined to be 0.0 radians, the natural horizon is at  $\pi/2$  radians, and the nadir (*i.e.*, straight down from the viewer) is at  $\pi$  radians. *skyAngle* is restricted to non-decreasing values in the range  $[0.0, \pi]$ . There shall be one more *skyColor* value than there are *skyAngle* values. The first colour value is the colour at the zenith, which is not specified in the *skyAngle* field. If the last *skyAngle* is less than  $\pi$ , then the colour band between the last *skyAngle* and the nadir is clamped to the last *skyColor*. The sky colour is linearly interpolated between the specified *skyColor* values.

The *groundColor* field specifies the colour of the ground at the various angles on the ground partial sphere. Angles for *groundColor* are specified in angle base units. The following assumes that the angle base units are radians. The equivalent values apply if an angle base unit other than radians is specified. The first value of the *groundColor* field specifies the colour of the ground at 0.0 radians representing the nadir (*i.e.*, straight down from the user). The *groundAngle* field specifies the angles from the nadir that the concentric circles of colour appear. The nadir of the sphere is implicitly defined at 0.0 radians. *groundAngle* is restricted to non-decreasing values in the range  $[0.0, \pi/2]$ . There shall be one more *groundColor* value than there are *groundAngle* values. The first colour value is for the nadir which is not specified in the *groundAngle* field. If the last *groundAngle* is less than  $\pi/2$ , the region between the last *groundAngle* and the equator is non-existent. The ground colour is linearly interpolated between the specified *groundColor* values.

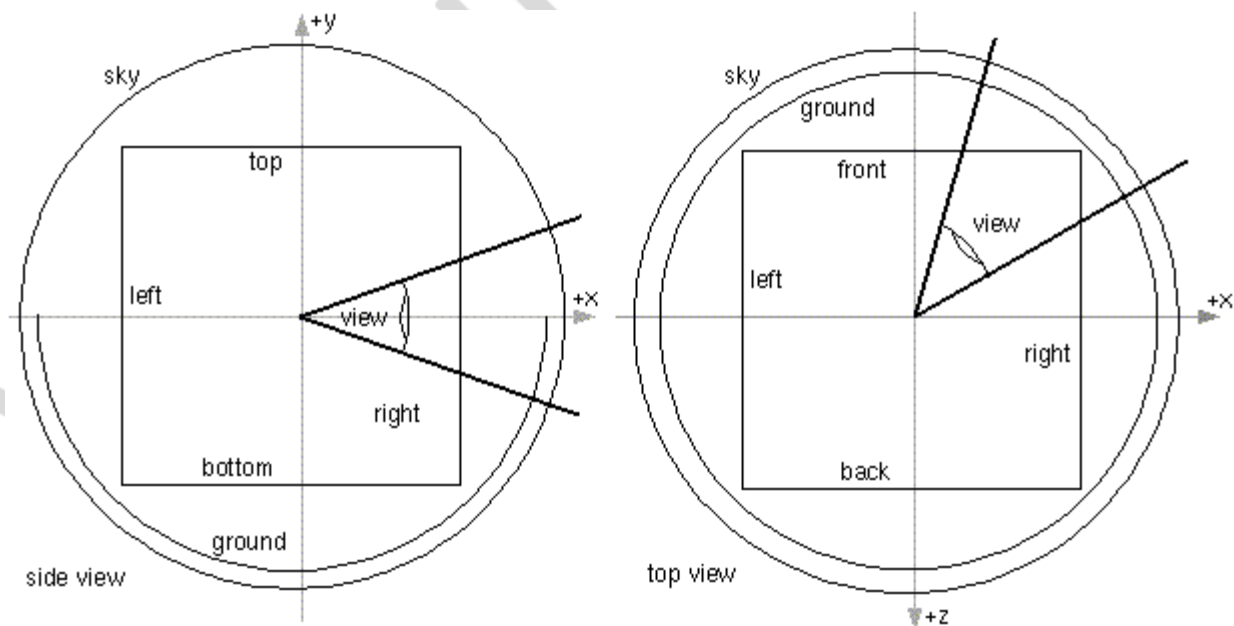
The *back*, *bottom*, *front*, *left*, *right*, and *top* fields specify a set of images that define a background panorama between the ground/sky backdrop and the scene's geometry. The panorama consists of six images, each of which is mapped onto a face of an infinitely large cube contained within the backdrop spheres and centred in the local coordinate system. The images are applied individually to each face of the cube. On the front, back, right, and left faces of the cube, when viewed from the origin looking down the negative Z-axis with the Y-axis as the view up direction, each image is mapped onto the corresponding face with the same orientation as if the image were displayed normally in 2D (*back* to back face, *front* to front face, *left* to left face, and *right* to right face). On the top face of the cube, when viewed from the origin looking along the +Y-axis with the +Z-axis as the view up direction, the *top* image is mapped onto the face with the same orientation as if the image were displayed normally in 2D. On the bottom face of the box, when viewed from the origin along the negative Y-axis with the negative Z-axis as the view up direction, the *bottom* image is mapped onto the face with the same orientation as if the image were displayed normally in 2D.

[Figure 24.1](#) illustrates the *X3DBackgroundNode* node backdrop and background textures.

Alpha values in the panorama images (*i.e.*, two or four component images) specify that the panorama is semi-transparent or transparent in regions, allowing earlier rendered layers or the *groundColor* and *skyColor* to be visible.

See [18 Texturing component](#) for a general description of texture maps.

Often, the *bottom* and *top* images will not be specified, to allow sky and ground to show. The other four images may depict surrounding mountains or other distant scenery.



**Figure 24.1 — X3D Background Node field relationships**

Panorama images may be one component (greyscale), two component (greyscale plus alpha), three component (full RGB colour), or four-component (full RGB colour plus alpha).

Ground colours, sky colours, and panoramic images do not translate with respect to the viewer, though they do rotate with respect to the viewer. That is, the viewer can never get any closer to the background, but can turn to examine all sides of the panorama cube, and can look up and down to see the concentric rings of ground and sky (if visible).

*X3D Background Node* nodes are not affected by [X3D Fog Object](#) nodes. Therefore, if a *X3D Background Node* node is active (*i.e.*, bound) while an *X3D Fog Object* node is active, the *X3D Background Node* node will be displayed with no fogging effects. It is the author's responsibility to set the *X3D Background Node* values to match the *X3D Fog Object* node values (EXAMPLE ground colours fade to fog colour with distance and panorama images tinted with fog colour). *X3D Background Node* nodes are not affected by light sources.

## 24.2.2 Fog semantics

### 24.2.2.1 Overview

This part of ISO/IEC 19775 supports two types of fog: global and local.

#### 24.2.2.2 Global fog semantics

Global fog applies to the entire world and is specified using a [Fog](#) node. Global fog blends the colours of all objects with the fog colour based on distance from the object to the camera. The further the distance the greater the amount of fog colour.

### 24.2.2.3 Local fog semantics

Local fog applies only within the same transformation hierarchy that contains the [LocalFog](#) node. This limits the effect of the fog to subsets of the world and supports the creation of realistic effects such as a smoke-filled room inside a larger building that is not smoke-filled. If a local fog and a global fog are both defined and active, the lighting contribution from the local fog shall be used instead of the global effect.

Local fog effects shall not affect nodes derived from [X3DBackgroundNode](#).

### 24.2.2.4 Local and bindable fog interaction

If a global [Fog](#) node is bound and a [LocalFog](#) node is enabled, the LocalFog node shall have precedence over the globally bound Fog node in determining the fog colour contribution to the lighting equations defined in [17 Lighting component](#).

### 24.2.2.5 Fog colour calculation

During the traversal of the scene graph, if more than one [LocalFog](#) node is encountered in the path from the root to a given renderable leaf node, only the contribution of the LocalFog instance closest to the leaf node shall be used. All other fog values shall be ignored.

## 24.3 Abstract types

### 24.3.1 X3DBackgroundNode

```
X3DBackgroundNode : X3DBindableNode {
  SFBool [in] set_bind
  MFFloat [in,out] groundAngle [] [0,π/2]
  MFColor [in,out] groundColor [] [0,1]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFFloat [in,out] skyAngle [] [0,π]
  MFColor [in,out] skyColor 0 0 0 [0,1]
  SFFloat [in,out] transparency 0 [0,1]
  SFTime [out] bindTime
  SFBool [out] isBound
}
```

*X3DBackgroundNode* is the abstract type from which all backgrounds inherit. *X3DBackgroundNode* is a bindable node that, when bound, defines the panoramic background for the scene. For complete information on backgrounds, see [24.2.1 Backgrounds](#).

### 24.3.2 X3DFogObject

```
X3DFogObject {
  SFColor [in,out] color 1 1 1 [0,1]
  SFString [in,out] fogType "LINEAR" ["LINEAR"|"EXPONENTIAL"]
  SFFloat [in,out] visibilityRange 0 [0,-∞)
}
```

*X3DFogObject* is the abstract **type interface** that describes a node that influences the lighting equation through the use of fog semantics. It defines the basic colour and rendering effects that influence the lighting equations as described in [17 Lighting component](#).



## 24.4 Node reference

### 24.4.1 Background

```

Background : X3DBackgroundNode {
  SFBool [in] set_bind
  MFFloat [in,out] groundAngle [] [0,π/2]
  MFColor [in,out] groundColor [] [0,1]
  MFString [in,out] backUrl [] [URI]
  MFString [in,out] bottomUrl [] [URI]
  MFString [in,out] frontUrl [] [URI]
  MFString [in,out] leftUrl [] [URI]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFString [in,out] rightUrl [] [URI]
  MFString [in,out] topUrl [] [URI]
  MFFloat [in,out] skyAngle [] [0,π]
  MFColor [in,out] skyColor 0 0 0 [0,1]
  SFFloat [in,out] transparency 0 [0,1]
  SFTime [out] bindTime
  SFBool [out] isBound
}

```

A background node that uses six static images to compose the backdrop. The common fields of the Background node are described in [24.2 Concepts](#). For the *backUrl*, *bottomUrl*, *frontUrl*, *leftUrl*, *rightUrl*, *topUrl* fields, browsers shall support the JPEG (see [2.\[JPEG\]](#)) and PNG (see [ISO/IEC 15948](#)) image file formats, and in addition, may support any other image format (EXAMPLE CGM) that can be rendered into a 2D image. Support for the GIF (see [GIF](#)) format is recommended (including transparency) . More detail on the *url* fields can be found in [9.2.1 URLs](#).

### 24.4.2 Fog

```

Fog : X3DBindableNode, X3DFogObject {
  SFBool [in] set_bind
  SFColor [in,out] color 1 1 1 [0,1]
  SFString [in,out] fogType "LINEAR" ["LINEAR"|"EXPONENTIAL"]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFFloat [in,out] visibilityRange 0 [0,∞)
  SFTime [out] bindTime
  SFBool [out] isBound
}

```

The Fog node provides a way to simulate atmospheric effects by blending objects with the colour specified by the *color* field based on the distances of the various objects from the viewer. The distances are calculated in the coordinate space of the Fog node. The *visibilityRange* specifies the distance in length base units (in the local coordinate system) at which objects are totally obscured by the fog. Objects located outside the *visibilityRange* from the viewer are drawn with a constant colour of *color*. Objects very close to the viewer are blended very little with the fog *color*. A *visibilityRange* of 0.0 disables the Fog node. The *visibilityRange* is affected by the scaling transformations of the Fog node's parents; translations and rotations have no effect on *visibilityRange*. Values of the *visibilityRange* field shall be in the range  $[0, \infty)$ .

Since Fog nodes are bindable children nodes (see [7.2.2 Bindable children nodes](#)), a Fog node stack exists, in which the top-most Fog node on the stack is currently active. To push a Fog node onto the top of the stack, a `TRUE` value is sent to the *set\_bind* field. Once active, the Fog node is bound to the browser view. A `FALSE` value sent to *set\_bind*, pops the Fog node from the stack and unbinds it from the browser viewer. More details on the Fog node stack can be found in [7.2.2 Bindable children nodes](#).

The *fogType* field controls how much of the fog colour is blended with the object as a function of distance. If *fogType* is "LINEAR", the amount of blending is a linear function of



the distance, resulting in a depth cueing effect. If *fogType* is "EXPONENTIAL," an exponential increase in blending is used, resulting in a more natural fog appearance.

The effect of fog on lighting calculations is described in [17 Lighting component](#).

### 24.4.3 FogCoordinate

```
FogCoordinate : X3DGeometricPropertyNode {
  MFFloat [in,out] depth [] [0,1]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This node defines a set of explicit fog depths on a per-vertex basis. This depth value shall be applied per-vertex and used to replace the automatically generated depth. Fog coordinates take precedence over implicitly generated depths; specifying fog coordinates will result in the implicit depth (specified by the *visibilityRange* field of a node derived from [X3DFogObject](#)) being ignored. Details on lighting equations can be found in [17.2.2 Lighting model](#).

One depth value per vertex shall be supplied. If the user does not provide a sufficient number of depth values, the last value defined shall be replicated for any further vertices. If too many depth values are supplied, the excess depth values shall be ignored.

### 24.4.4 LocalFog

```
LocalFog : X3DChildNode, X3DFogObject {
  SFColor [in,out] color 1 1 1 [0,1]
  SFBool [in,out] enabled TRUE
  SFString [in,out] fogType "LINEAR" ["LINEAR"|"EXPONENTIAL"]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFFloat [in,out] visibilityRange 0 [0,-∞)
}
```

The LocalFog node provides a way to simulate atmospheric effects by blending objects with the colour specified by the color field based on the distances of the various objects from the viewer. The distances are calculated in the coordinate space of the LocalFog node. The *visibilityRange* field specifies the distance in metres (in the local coordinate system) at which objects are totally obscured by the fog. Objects located outside the *visibilityRange* from the viewer are drawn with a constant colour of *color*. Objects very close to the viewer are blended very little with the fog color. A *visibilityRange* of 0.0 disables the LocalFog node. The *visibilityRange* is affected by the scaling transformations of the LocalFog node's parents; translations and rotations have no affect on *visibilityRange*.

The *fogType* field controls how much of the fog colour is blended with the object as a function of distance. If *fogType* is "LINEAR", the amount of blending is a linear function of the distance, resulting in a depth cueing effect. If *fogType* is "EXPONENTIAL", an exponential increase in blending is used, resulting in a more natural fog appearance.

The effect of fog on lighting calculations is described in [17 Lighting component](#).

### 24.4.5 TextureBackground

```
TextureBackground : X3DBackgroundNode {
  SFBool [in] set_bind
  MFFloat [in,out] groundAngle [] [0,π/2]
  MFColor [in,out] groundColor [] [0,1]
```

```

SFNode [in,out] backTexture  NULL [X3DTexture2DNode,MultiTexture]
SFNode [in,out] bottomTexture NULL [X3DTexture2DNode,MultiTexture]
SFNode [in,out] frontTexture  NULL [X3DTexture2DNode,MultiTexture]
SFNode [in,out] leftTexture   NULL [X3DTexture2DNode,MultiTexture]
SFNode [in,out] metadata     NULL [X3DMetadataObject]
SFNode [in,out] rightTexture  NULL [X3DTexture2DNode,MultiTexture]
SFNode [in,out] topTexture    NULL [X3DTexture2DNode,MultiTexture]
MFFloat [in,out] skyAngle    [] [0,π]
MFColor [in,out] skyColor    0 0 0 [0,1]
SFFloat [in,out] transparency 0 [0,1]
SFTime [out] bindTime
SFBool [out] isBound
}

```

The `TextureBackground` node uses six individual texture nodes to compose the backdrop. Unlike the [Background](#) node, which only supports static image formats referenced by URL fields, the contents of the `TextureBackground` node can be arbitrary texture types, including [ImageTexture](#), [PixelTexture](#), [MovieTexture](#) and [MultiTexture](#). The common fields of the `TextureBackground` node are described in [24.2 Concepts](#).

`TextureBackground` supports the creation of rich backgrounds with animation. It also allows the world author to attach load sensors (see [9.4.3 LoadSensor](#)) to the node's texture fields to receive notification of when elements of the background are loaded.

`TextureBackground` supports a *transparency* value that allows the scene to overlay other elements in an application. A *transparency* value of zero specifies that the background is fully opaque obscuring all content in the underlying window. A *transparency* value of one specifies that the background specified by the `TextureBackground` node is fully transparent causing the `TextureBackground` to not be visible so that all underlying content appears as the background. The value of the *transparency* field is applied to the *skyColor* and *groundColor* by first converting the *transparency* value to an alpha value using the formula:

$$\text{alpha} = (1 - \text{transparency})$$

The alpha value is then multiplied against the components of the *skyColor* and *groundColor* (including the alpha component, if provided) to obtain the color that is applied to the underlying window content. The *transparency* value is not applied to the six texture fields. Transparency of these fields can be achieved by using alpha values within their images.

For the *backTexture*, *bottomTexture*, *frontTexture*, *leftTexture*, *rightTexture*, *topTexture* fields, browsers shall support any X3DTexture node types supported in the currently supported profile.

## 24.5 Support levels

The Environmental Effects component provides three levels of support as specified in [Table 24.2](#). Level 1 is intended to support simple backgrounds for lightweight profiles. Level 2 provides additional environmental effects, including full background features, fog, and limited texture backgrounds. Level 3 provides full support for texture backgrounds.

**Table 24.2 — Environmental effects component support levels**

Level	Prerequisites	Nodes/Features	Support

1	Core 1 Time 1 Grouping 1		
		<i>X3DBackgroundNode</i> (abstract)	n/a
		Background	<i>groundAngle</i> and <i>groundColor</i> optionally supported. <i>backURL</i> , <i>frontURL</i> , <i>leftURL</i> , <i>rightURL</i> , <i>topURL</i> optionally supported. <i>skyAngle</i> optionally supported. One <i>skyColor</i> .
2	Core 1 Time 1 Grouping 1		
		All Level 1 Environmental Effects nodes	All fields fully supported.
		Fog	All fields fully supported.
3	Core 1 Time 1 Grouping 1		
		All Level 2 Environmental Effects nodes	All fields fully supported.
		TextureBackground	All fields fully supported.
4	Core 1 Time 1 Grouping 1		
		All Level 3 Environmental Effects nodes	All fields fully supported.
		FogCoordinate	All fields fully supported.
		LocalFog	All fields fully supported.





## Extensible 3D (X3D) Part 1: Architecture and base components

### Annex C

(normative)

### Interactive profile

---



#### C.1 General

This annex defines the X3D components that comprise the Interactive profile. This includes not only the nodes that shall be supported but also which fields in the supported nodes may be ignored.

This profile is targeted towards:

- implementing a lightweight playback engine that supports rich graphics and interactivity,
- possible implementation in a low-footprint engine requiring limited navigation and environmental sensor control (EXAMPLE an applet or small browser plug-in), and
- allowing a broader range of implementations by eliminating some complexity of a complete X3D implementation.

#### C.2 Topics

[Table C.1](#) provides links to the major topics in this annex.

**Table C.1 — Topics**

- |   |
|---|
| <ul style="list-style-type: none"><li>• <a href="#">C.1 General</a></li><li>• <a href="#">C.2 Topics</a></li><li>• <a href="#">C.3 Component support</a></li><li>• <a href="#">C.4 Conformance criteria</a></li><li>• <a href="#">C.5 Node set</a></li><li>• <a href="#">C.6 Other limitations</a></li><li>• <a href="#">Table C.1 — Topics</a></li></ul> |
|---|

- [Table C.2 — Components and levels](#)
- [Table C.3 — Nodes for conforming to the Interactive profile](#)
- [Table C.4 — Other limitations](#)

## C.3 Component support

[Table C.2](#) lists the components and their levels which shall be supported in the Interactive profile. Tables C.2 and C.3 describe limitations on required support for nodes and fields contained within these components.

**Table C.2 — Components and levels**

Component	Level	Reference
Core	1	<a href="#">7.5 Support levels</a>
Time	1	<a href="#">8.5 Support levels</a>
Networking	2	<a href="#">9.5 Support levels</a>
Grouping	2	<a href="#">10.5 Support levels</a>
Rendering	3	<a href="#">11.5 Support levels</a>
Shape	1	<a href="#">12.5 Support levels</a>
Geometry3D	3	<a href="#">13.4 Support levels</a>
Lighting	2	<a href="#">17.5 Support levels</a>
Texturing	2	<a href="#">18.5 Support levels</a>
Interpolation	2	<a href="#">19.5 Support levels</a>
Pointing device sensor	1	<a href="#">20.5 Support levels</a>
Key device sensor	1	<a href="#">21.5 Support levels</a>
Environmental sensor	1	<a href="#">22.5 Support levels</a>
Navigation	1	<a href="#">23.4 Support levels</a>
Environmental effects	1	<a href="#">24.5 Support levels</a>
Event utilities	1	<a href="#">30.5 Support levels</a>

## C.4 Conformance criteria

Conformance to this profile shall include conformance criteria defined by the

specifications for those components and levels listed in [Table C.2](#).

In Tables C.3 and C.4, the first column defines the item for which conformance is being defined. In some cases, general limits are defined but are later overridden in specific cases by more restrictive limits. The second column defines the requirements for a X3D file conforming to the Interactive profile; if a X3D file contains any items that exceed these limits, it may not be possible for a X3D browser conforming to the Interactive profile to successfully parse that X3D file. The third column defines the minimum complexity for a X3D scene that a X3D browser conforming to the Interactive profile shall be able to present to the user. Fields flagged as "not supported" may be supported by browsers which conform to the Interactive profile. The word "ignore" in the minimum browser support column refers only to the display of the item; in particular, *set\_* events to ignored inputOutput fields shall still generate corresponding *\_changed* events.

## C.5 Node set

[Table C.3](#) lists the nodes which shall be supported in the Interactive profile and specifies any fields in these nodes for which this profile requires less than full support.

**Table C.3 — Nodes for conforming to the Interactive profile**

Item	X3D File Limit	Minimum Browser Support
Anchor	No restrictions.	Full support.
Appearance	No restrictions.	<i>textureTransform</i> optionally supported. <i>lineProperties</i> not supported. <i>fillProperties</i> not supported.
Background	No restrictions.	<i>groundAngle</i> and <i>groundColor</i> optionally supported. <i>backURL</i> , <i>frontURL</i> , <i>leftURL</i> , <i>rightURL</i> , <i>topURL</i> optionally supported. <i>skyAngle</i> optionally supported. One <i>skyColor</i> .
BooleanFilter	No restrictions.	Full support.
BooleanSequencer	No restrictions.	Full support.
BooleanToggle	No restrictions.	Full support.
BooleanTrigger	No restrictions.	Full support.
Box	No	Full support.

	restrictions.	
Color	15,000 colours.	15,000 colours.
ColorInterpolator	Restrictions as for all interpolators.	Full support as for all interpolators.
ColorRGBA	15,000 colours.	15,000 colours. Alpha component optionally supported.
Cone	No restrictions.	Full support.
Coordinate	65,535 points	65,535 points.
CoordinateInterpolator	15,000 coordinates per <i>keyValue</i> . Restrictions as for all interpolators.	15,000 coordinates per <i>keyValue</i> . Support as for all interpolators.
Cylinder	No restrictions.	Full support.
CylinderSensor	No restrictions.	Full support.
DirectionalLight	No restrictions.	Not scoped by parent Group or Transform.
ElevationGrid	No restrictions.	<i>ccw</i> optionally supported.
Group	Restrictions as for all groups.	Support as for all groups.
ImageTexture	JPEG ( <a href="#">2. [JPEG]</a> ) and PNG ( <a href="#">2. [115948]</a> ) format.	JPEG ( <a href="#">2. [JPEG]</a> ) and PNG ( <a href="#">2. [115948]</a> ) format.
		10 vertices per face. 5000 faces. 65,535 indices in any index field.  <i>ccw</i> optionally supported. <i>set_colorIndex</i> optionally supported. <i>set_normalIndex</i> optionally supported. <i>normal</i>



IndexedFaceSet	10 vertices per face. 5000 faces. Less than 65,535 indices.	optionally supported. Only convex indexed face sets supported. Hence, <i>convex</i> optionally supported. For <i>creaseAngle</i> , only 0 and $\pi$ radians supported. <i>normalIndex</i> optionally supported.  Face list shall be well-defined as follows: <ol style="list-style-type: none"> <li>1. Each face is terminated with -1, including the last face in the array.</li> <li>2. Each face contains at least three non-coincident vertices.</li> <li>3. A given <i>coordIndex</i> is not repeated in a face.</li> <li>4. The vertices of a face shall define a planar polygon.</li> <li>5. The vertices of a face shall not define a self-intersecting polygon.</li> </ol>
IndexedLineSet	15,000 total vertices. 15,000 indices in any index field.	15,000 total vertices. 15,000 indices in any index field. <i>set_colorIndex</i> optionally supported. <i>set_coordIndex</i> optionally supported.
IndexedTriangleFanSet	5,000 total faces. 15,000 indices in any index field.	Full support.
IndexedTriangleSet	5,000 total faces. 15,000 indices in any index field.	Full support.
IndexedTriangleStripSet	5,000 total faces. 15,000 indices in any index field.	Full support.
Inline	No	All fields fully supported except

	restrictions.	<i>load</i> which is optionally supported.
IntegerSequencer	No restrictions.	Full support.
IntegerTrigger	No restrictions.	Full support.
KeySensor	No restrictions.	Full support.
LineSet	15,000 total vertices.	15,000 total vertices.
Material	No restrictions.	<i>ambientIntensity</i> optionally supported. <i>shininess</i> optionally supported. <i>specularColor</i> optionally supported. A Material with <i>emissiveColor</i> not equal to (0,0,0), <i>diffuseColor</i> equal to (0,0,0) is an unlit material. One-bit transparency; transparency values $\geq 0.5$ transparent.
MetadataBoolean	No restrictions.	Full support.
MetadataDouble	No restrictions.	Full support.
MetadataFloat	No restrictions.	Full support.
MetadataInteger	No restrictions.	Full support.
MetadataSet	No restrictions.	Full support.
MetadataString	No restrictions.	Full support.
MultiTexture	No restrictions.	At least two textures displayed per node with any number specified. Full support.
MultiTextureCoordinate	15,000 coordinates.	15,000 coordinates.
MultiTextureTransform	No restrictions.	At least two textures displayed per node with any number specified. Full support.

NavigationInfo	No restrictions.	<i>avatarSize</i> optionally supported. <i>speed</i> optionally supported. <i>visibilityLimit</i> optionally supported. For <i>type</i> , only "ANY", "FLY", "EXAMINE", and "LOOKAT" modes supported.
Normal	15,000 normals	15,000 normals.
NormalInterpolator	15,000 normals	15,000 normals except as for all interpolators.
OrientationInterpolator	Restrictions as for all interpolators.	Full support except as for all interpolators.
PixelTexture	512 width. 512 height.	512 width. 512 height. Display fully transparent and fully opaque pixels.
PlaneSensor	No restrictions.	Full support.
PointLight	No restrictions.	<i>radius</i> optionally supported. Linear attenuation.
PointSet	5000 points.	5000 points.
PositionInterpolator	Restrictions as for all interpolators.	Full support except as for all interpolators.
ProximitySensor	No restrictions.	<i>position_changed</i> optionally supported. <i>orientation_changed</i> optionally supported.
ScalarInterpolator	Restrictions as for all interpolators.	Full support except as for all interpolators.
Shape	No restrictions.	Full support.
Sphere	No restrictions.	Full support.
SphereSensor	No restrictions.	Full support.
SpotLight	No restriction	<i>beamWidth</i> optionally supported. <i>radius</i> optionally supported. Linear attenuation.

StringSensor	No restrictions.	Full support.
Switch	No restrictions	Full support.
TextureCoordinate	15,000 coordinates.	15,000 coordinates.
TextureCoordinateGenerator	No restrictions.	Full support.
TextureTransform	No restrictions.	Full support.
TimeSensor	No restrictions.	<i>pause</i> optionally supported. <i>isPaused</i> optionally supported. <i>resumeTime</i> optionally supported.
TimeTrigger	No restrictions.	Full support.
TouchSensor	No restrictions.	Full support.
Transform	Restrictions as for all groups.	Full support except as for all groups.
TriangleFanSet	5,000 triangles per fan. 15,000 total triangles.	Full support.
TriangleSet	15,000 triangles.	Full support.
TriangleStripSet	5,000 triangles per strip. 15,000 total triangles.	Full support.
Viewpoint	No restrictions.	<i>fieldOfView</i> optionally supported. <i>description</i> optionally supported. <i>retainUserOffsets</i> optionally supported. All other fields fully supported.
VisibilitySensor	No restrictions.	Always visible.
WorldInfo	No restrictions.	Full support.

## C.6 Other limitations

[Table C.4](#) specifies other aspects of X3D functionality which are supported by this profile. Note that general items refer only to those specific nodes listed in [Table C.3](#).

**Table C.4 — Other limitations**

Item	X3D File Limit	Minimum Browser Support
All groups	500 children.	500 children. Ignore <i>bboxCenter</i> and <i>bboxSize</i> .
All interpolators	1000 key-value pairs.	1000 key-value pairs.
All lights	8 simultaneous lights.	8 simultaneous lights.
Names for DEF/field	50 utf8 octets.	50 utf8 octets.
All <i>url</i> fields	10 URLs.	10 URLs. URN's ignored. Support `http`, `file`, and `ftp` protocols. Support relative URLs where relevant.
SFBool	No restrictions.	Full support.
SFColor	No restrictions.	Full support.
SFColorRGBA	No restrictions.	Full support.
SFDouble	Mp restrictions.	Full support. Range $\pm 1e\pm 12$ . Precision $1e-7$ .
SFFloat	No restrictions.	Full support.
SFImage	512 width. 512 height.	512 width. 512 height.
SFInt32	No restrictions.	Full support.
SFNode	No restrictions.	Full support.
SFRotation	No restrictions.	Full support.
SFString	30,000 utf8 octets.	30,000 utf8 octets.
SFTime	No restrictions.	Full support.
SFVec2d	15,000 values.	15,000 values.
SFVec2f	15,000 values.	15,000 values.

SFVec3d	15,000 values.	15,000 values.
SFVec3f	15,000 values.	15,000 values.
MFCColor	15,000 values.	15,000 values.
MFCColorRGBA	15,000 values.	15,000 values.
MFDouble	1000 values.	1000 values.
MFFloat	1,000 values.	1,000 values.
MFInt32	20,000 values.	20,000 values.
MFNode	500 values.	500 values.
MFRotation	1,000 values.	1,000 values.
MFString	30,000 utf8 octets per string, 10 strings.	30,000 utf8 octets per string, 10 strings.
MFTime	1,000 values.	1,000 values.
MFVec2d	15,000 values.	15,000 values.
MFVec2f	15,000 values.	15,000 values.
MFVec3d	15,000 values.	15,000 values.
MFVec3f	15,000 values.	15,000 values.





## Extensible 3D (X3D) Part 1: Architecture and base components

### 4 Concepts

Editors note: multiple sections in the Concepts clause will receive additions and modifications to describe how X3D models are included and interact with external surfaces such as HTML5/DOM Web-page presentations. Current focus is on open-source implementation and evaluation using [X3DOM](#) and [X\\_ITE](#).

---



#### 4.1 General

##### 4.1.1 Topics in this clause

This clause describes the X3D core concepts, including how X3D scenes are authored and played back, the run-time semantics of the X3D scene, modularization through components and profiles, conformance via support levels, data encoding semantics, programmatic access, and networking considerations.

[Table 4.1](#) provides links to the major topics in this clause.

**Table 4.1 — Topics**

- [4.1 General](#)
  - [4.1.1 Topics in this clause](#)
  - [4.1.2 Overview](#)
  - [4.1.3 Conventions used](#)
- [4.2 Authoring and playback](#)
  - [4.2.1 X3D browsers](#)
  - [4.2.2 X3D generators](#)
  - [4.2.3 X3D loaders](#)
- [4.3 The scene graph](#)
  - [4.3.1 Overview](#)
  - [4.3.2 Root nodes](#)
  - [4.3.3 Scene graph hierarchy](#)
  - [4.3.4 Descendant and ancestor nodes](#)
  - [4.3.5 Transformation hierarchy](#)
  - [4.3.6 Standard units and coordinate system](#)



- [4.3.7 Behaviour graph](#)
- [4.4 Run-time environment](#)
  - [4.4.1 Overview](#)
  - [4.4.2 Object model](#)
    - [4.4.2.1 Overview](#)
    - [4.4.2.2 Field semantics](#)
    - [4.4.2.3 Interface hierarchy](#)
    - [4.4.2.4 Modifying objects](#)
      - [4.4.2.4.1 Routes](#)
      - [4.4.2.4.2 Modifying objects via programmatic access](#)
    - [4.4.2.5 Object life cycle](#)
  - [4.4.3 DEF/USE semantics](#)
  - [4.4.4 Prototype semantics](#)
    - [4.4.4.1 Introduction](#)
    - [4.4.4.2 PROTO interface declaration semantics](#)
    - [4.4.4.3 PROTO definition semantics](#)
    - [4.4.4.4 Prototype scoping rules](#)
  - [4.4.5 External prototype semantics](#)
    - [4.4.5.1 Introduction](#)
    - [4.4.5.2 EXTERNPROTO interface semantics](#)
    - [4.4.5.3 EXTERNPROTO URL semantics](#)
  - [4.4.6 Import/Export semantics](#)
  - [4.4.7 Run-time name scope](#)
  - [4.4.8 Event model](#)
    - [4.4.8.1 Events](#)
    - [4.4.8.2 Routes](#)
    - [4.4.8.3 Execution model](#)
    - [4.4.8.4 Loops](#)
    - [4.4.8.5 Fan-in and fan-out](#)
    - [4.4.8.6 Internal/external event passing](#)
- [4.5 Components](#)
  - [4.5.1 Overview](#)
  - [4.5.2 Defining components](#)
  - [4.5.3 Base components](#)
- [4.6 Profiles](#)
  - [4.6.1 Overview](#)
  - [4.6.2 Defining profiles](#)
  - [4.6.3 Relationship between profiles and components](#)
- [4.7 Support levels](#)
- [4.8 Data encodings](#)
- [4.9 Scene access interface \(SAI\)](#)
- [4.10 Component and profile registration](#)
- [Figure 4.1 — X3D Architecture](#)
- [Figure 4.2 — Interface hierarchy](#)
- [Figure 4.3 — Conceptual execution model](#)

- [Table 4.1 — Topics](#)
- [Table 4.2 — Standard units](#)
- [Table 4.3 — Derived units](#)
- [Table 4.4 — Rules for mapping PROTOTYPE declarations to node instances](#)
- [Table 4.5 — Example support level table](#)

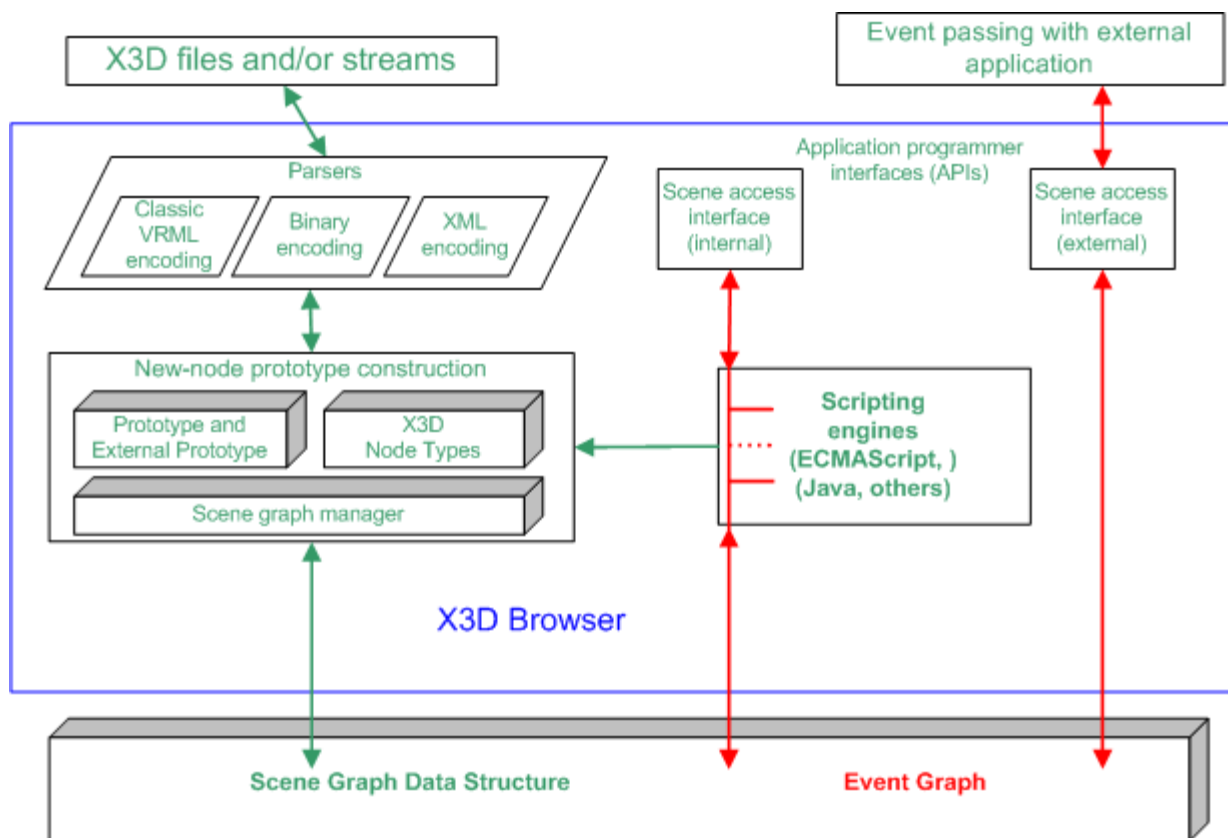
## 4.1.2 Overview

Conceptually, each X3D application is a 3D time-based space that contains graphic and aural objects that can be loaded over a network and dynamically modified through a variety of mechanisms. The semantics of X3D describe an abstract functional behaviour of time-based, interactive 3D, multimedia information. X3D does not define physical devices or any other implementation-dependent concepts (*e.g.*, screen resolution and input devices). X3D is intended for a wide variety of devices and applications, and provides wide latitude in interpretation and implementation of the functionality. For example, X3D does not assume the existence of a mouse or 2D display device.

Each X3D application:

- a. implicitly establishes a world coordinate space for all objects defined, as well as all objects included by the application;
- b. explicitly defines and composes a set of 2D, 3D and multimedia objects;
- c. can specify hyperlinks to other files and applications;
- d. can define object behaviours;
- e. can connect to external modules or applications via programming and scripting languages.

The X3D system architecture is shown in [Figure 4.1](#).



**Figure 4.1 — X3D architecture**

The abstract structure of the sequence of statements that form an X3D world is specified in [7.2.5 Abstract X3D structure](#).

### 4.1.3 Conventions used

The following conventions are used throughout this part of ISO/IEC 19775:

*Italics* are used for field names, and are also used when new terms are introduced and equation variables are referenced.

A `fixed-space` font is used for URL addresses and source code examples.

Node type names are appropriately capitalized (*e.g.*, "The Billboard node is a grouping node..."). However, the concept of the node is often referred to in lower case in order to refer to the semantics of the node, not the node itself (*e.g.*, "To rotate the billboard...").

The form "0xhh" expresses a byte as a hexadecimal number representing the bit configuration for that byte.

Throughout this part of ISO/IEC 19775, references to International Standards cite the number of the standard and hyperlinks to the reference in [2 Normative references](#). References to portions of this International Standard consist of the clause or subclause number followed by the title of the clause or subclause. The text consisting of the number and title is hyperlinked to the referenced material. References to external documents that are not International Standards are denoted using the "x.[ABCD]" notation, where "x" denotes in which clause the reference is described and "[ABCD]" is

an abbreviation of the reference title. For the Bibliography, the "x." is omitted.

In addition, the first reference to a node or node type in a subclause will be hyperlinked to the definition of that node or node type.

EXAMPLE "2.[ABCD]" refers to a reference described in [2 Normative references](#) and [ABCD] refers to a reference described in the [Bibliography](#).

## 4.2 Authoring and playback

### 4.2.1 X3D browsers

The interpretation, execution, and presentation of X3D files occurs using a mechanism known as a *browser*, which displays the shapes and sounds in the scene graph. This presentation is known as a *virtual world* and is navigated in the browser by a human or mechanical entity, known as a *user*. The world is displayed as if experienced from a particular location; that position and orientation in the world is known as the *viewer*. The browser may provide navigation paradigms (such as walking or flying) that enable the user to move the viewer through the virtual world.

In addition to navigation, the browser provides a limited mechanism allowing the user to interact with the world through sensor nodes in the scene graph hierarchy. Sensors respond to user interaction with geometric objects in the world, the movement of the user through the world, or the passage of time. Additionally, the X3D Scene Access Interface (SAI) defined in [Part 2 of this International Standard](#) provides mechanisms for getting user input, and for getting and setting the current viewpoint. To provide navigation capabilities, a viewer may use the SAI to provide the user with the ability to navigate. Additionally, authors may use scripting or programming languages with bindings to the SAI to implement their own navigation algorithms. Other profiles may specify navigation capabilities as a requirement of the viewer; implementations of such viewers will typically do so by making use of the SAI.

The visual presentation of geometric objects in an X3D world follows a conceptual model designed to resemble the physical characteristics of light. The X3D lighting model describes how appearance properties and lights in the world are combined to produce displayed colours (see [17 Lighting component](#) for details).

### 4.2.2 X3D generators

A *generator* is a human or computerized creator of X3D files. It is the responsibility of the generator to ensure the correctness of the X3D file and the availability of supporting assets (*e.g.*, images, audio clips, other X3D files) referenced therein. It is also the responsibility of the generator to insure that the functionality represented in the X3D file is correctly stated in the profile, component and level information in the header statement of the file.

### 4.2.3 X3D loaders

A *loader* is a program responsible for loading X3D content but does not apply any run-time execution to the content. Geometry is presented as though time has not run,

although the loader is free to load textures and other remotely defined content. A time zero loader is typically found in modelling tools that intend to construct or modify existing X3D content without evaluating the run-time aspects of the specification.

A second form of loader may load files and allow run-time execution of content, but it does so as part of a larger user interface and 3D graphics rendering engine. Such loaders might be used to load individual models such as trees in a game environment, but the run-time evaluation of the X3D content is dependent on the external application, and is not self contained in the same fashion as an X3D browser.

## 4.3 The scene graph

### 4.3.1 Overview

The basic unit of the X3D run-time environment is the *scene graph*. This structure contains all the objects in the system and their relationships. Relationships are contained along several axes of the scene graph. The *transformation hierarchy* describes the spatial relationship of rendering objects. The *behavior graph* describes the connections between fields and the flow of events through the system. Each scene graph may also interact with external surfaces such as HTML5/DOM Web-page presentations.

### 4.3.2 Root nodes

An X3D file contains zero or more root nodes. The root nodes for an X3D file are those nodes defined by the node statements or USE statements that are not contained in other node or PROTO statements. Root nodes shall be children nodes as specified in [10 Grouping component](#) or the LayerSet node as specified in [35.4.2 LayerSet](#).

X3D4 goals related to HTML5/DOM:

- Usage of either X3D or X3DCanvas capabilities for style, other HTML attributes
- url (or src) field to simply refer to an X3D model to load (see current X\_ITE approach)
- specifying that multiple encodings are allowed
- whether or not multiple distinct scenes can be loaded at once, or require separate declarations

### 4.3.3 Scene graph hierarchy

An X3D scene graph is a directed acyclic graph. Nodes can contain specific fields with one or more children nodes which participate in the hierarchy. These may, in turn, contain nodes (or instances of nodes). This hierarchy of nodes is called the *scene graph*. Each arc in the graph from A to B means that node A has a field whose value directly contains node B. See [\[FOLEY\]](#) for details on hierarchical scene graphs.

### 4.3.4 Descendant and ancestor nodes

The *descendants* of a node are all of the nodes in its fields, as well as all of those nodes'

descendants. The *ancestors* of a node are all of the nodes that have the node as a descendant.

### 4.3.5 Transformation hierarchy

The transformation hierarchy includes all of the root nodes and root node descendants that are considered to have one or more particular locations in the virtual world. X3D includes the notion of *local coordinate systems*, defined in terms of transformations from ancestor coordinate systems. The coordinate system in which the root nodes are displayed is called the *world coordinate system*.

An X3D browser's task is to present an X3D file to the user; it does this by presenting the transformation hierarchy to the user. The transformation hierarchy describes the directly perceptible parts of the virtual world.

Some nodes, such as sensors and environmental nodes, are in the scene graph but not affected by the transformation hierarchy. These include [CoordinateInterpolator](#), [Script](#), [TimeSensor](#), and [WorldInfo](#).

Some nodes, such as [Switch](#) or [LOD](#), contain a list of children, of which at most one is traversed during rendering. However, for the purposes of computing scene position, all children of these nodes are considered to be part of the transformation hierarchy, whether they are traversed during rendering or not. For instance, a [Viewpoint](#) node which is a child of a Switch whose `whichChoice` field is set to -1 (indicating that none of its children should be traversed during rendering) still uses the local coordinate space of the Switch to determine its position in the scene.

The transformation hierarchy shall be a directed acyclic graph; a node in the transformation hierarchy that is its own ancestor is considered invalid and shall be ignored. The following is an example of a node in the scene graph that is its own ancestor:

```
DEF T Transform {
  children {
    shape { ... }
    USE T
  }
}
```

### 4.3.6 Standard units and coordinate system

ISO/IEC 19775 defines the initial base unit of measure of the world coordinate system to be metres. However, the world coordinate units may be modified by specifying a different length unit using the UNIT statement. All other coordinate systems are then built from transformations based upon the specified world coordinate system. Other measurements used in this International Standard have their own initial base units.

[Table 4.2](#) lists the initial base units for ISO/IEC 19775, including the reference for each unit in [ISO 80000](#).

**Table 4.2 — Standard units**

Category	Initial base unit	Reference
----------	-------------------	-----------

angle	radian	ISO 80000-3:2006 item 3-5.a
force	newton	ISO 80000-4:2006 item 4-9.a and item 4-9.1
length	metre	ISO 80000-3:2006 item 3-1.a
mass	kilogram	ISO 80000-4:2006 item 4-1.a

The initial base units for the entire hierarchy of an X3D world may be changed to another default base unit by using one or more UNIT statements as specified in [7 Core component](#). In this International Standard, the initial base units of measure are assumed. Any ranges specified in initial base units apply to their equivalent limits in the specified default base unit. The browser shall convert the default base unit to initial base units as necessary for correct processing.

The base unit of time is seconds and cannot be changed.

Additional units, called *derived units* are used in this International Standard. A derived unit depends on the current base units. The value for a derived unit can be calculated using the appropriate formula from Table 4.3:

**Table 4.3 — Derived units**

Category	Initial base unit	Reference
acceleration	length/second <sup>2</sup>	ISO 80000-3:2006 item 3-9.a
angular_velocity angular_rate	angle/second	ISO 80000-3:2006 item 3-10.a
area	length <sup>2</sup>	ISO 80000-3:2006 item 3-3.a
speed	length/second	ISO 80000-3:2006 item 3-8.a and item 3-8.1
volume	length <sup>3</sup>	ISO 80000-3:2006 item 3-4.a

The standard colour space used by this International Standard is RGB where each colour component has the range [0.,1.].

ISO/IEC 19775 uses a Cartesian, right-handed, three-dimensional coordinate system. By default, the viewer is on the Z-axis looking down the -Z-axis toward the origin with +X to the right and +Y straight up. A modelling transformation (see the [Transform](#) node definition in [10 Grouping component](#) and the [Billboard](#) node definition in [23 Navigation component](#)) or viewing transformation (see the [X3DViewpointNode](#) node type definition in [23 Navigation component](#)) can be used to alter this default projection.

### 4.3.7 Behaviour graph



The event model of X3D allows the declaration of connections between fields (routes) and a model for the propagation of events along those connections. The behavior graph is the collection of these field connections. It can be changed dynamically by rerouting, adding or breaking connections. Events are injected into the system and propagate through the behavior graph in a well defined order.

Fields can only be routed to other fields with the same data type, unless a component supports an extension to this rule.

## 4.4 Run-time environment

### 4.4.1 Overview

The X3D run-time environment maintains the current state of the scene graph, renders the scene as needed, receives input from a variety of sources (*Sensors*) and performs changes to the scene graph in response to instructions from the behavioral system. The X3D run-time environment manages the life cycle of objects, including built-in and user-defined objects and programmatic scripts. The run-time environment coordinates the processing of *Events*, the primary means of generating behaviors in X3D. The run-time environment also manages interoperation between the X3D browser and host application for file delivery, hyperlinking, page integration and external programmatic access.

The run-time environment manages objects. X3D supports several types of *built-in objects* that contain generally useful functionality in the run-time environment. There are built-in objects to represent data structures such as an *SFVec3f* 3D vector value, nodes such as geometry (e.g., [Cylinder](#)), and ROUTEs between nodes. Each node contains zero or more *fields* that define storage for data values, and/or zero or more *events* for sending messages to/from the object. Nodes are instantiated by declaring them in a file or by using procedural code at run-time. The author may create new node types using the prototyping mechanism (see [4.4.4 Prototype semantics](#)). These nodes become part of the run-time environment and behave exactly like built-in nodes. New nodes can be created declaratively by including a prototype declaration in a file, by including an external prototype referencing a prototype declaration in a separate location, or by using a native prototype declaration provided by the run-time environment itself. PROTOs may only be used to create other nodes, not fields or routes.

*Events* are the primary means of generating behaviors in the X3D run-time environment. Events are used throughout X3D: driving time-based animations; handling object picking; detecting user movement and collision; changing the scene graph hierarchy. The run-time environment manages the propagation of events through the system and order of evaluation according to a well-defined set of rules.

An author of X3D content can control the creation and management of scenes, rendering and behavior, and loading of media assets. The loading and incorporation of authored extensions, which can be written in X3D or an external language, can also be controlled. The ability to make content-defined extensions is provided in profiles that support the Prototyping mechanism.

## 4.4.2 Object model

### 4.4.2.1 Overview

The X3D system is made up of abstract individual entities called *objects*. This part of ISO/IEC 19775 defines a functional specification for each object type but does not dictate implementation. A compliant implementation of an object shall behave according to its functional specification as provided in [5 Field type reference](#), clauses 7 through 40 describing components, [Part 2 of ISO/IEC 19775](#) or additional parts of this standard that define object, field or node types. An X3D author arranges objects in the scene graph using one of the declarative X3D encodings described in [ISO/IEC 19776](#) or other future encoding formats, or at run time using built-in scripting (if the supported profile provides it) or some other form of programmatic access to the scene graph (see [Part 2 of ISO/IEC 19775](#)).

Objects representing lightweight concepts such as data storage and operations on data of that type are called *fields* and are derived from the [X3DField](#) object. Objects representing more complete spatial or temporal processing concepts are called *nodes* and are derived from the [X3DNode](#) object. Nodes contain one or more fields that hold data values or send or receive events for that node.

Some nodes implement additional functionality by inheritance of *interfaces* that represent common properties or functionality, such as bounding boxes for visual objects and grouping nodes or a specification that a particular object represents metadata. In addition, X3D defines object types for accessing scene graph information not stored in fields or nodes, such as ROUTEs, PROTO declarations, Component/Profile information and world metadata.

A field may contain either a single value of the given type or an array of such types. Throughout this document, a field type containing a single value is said to be of the given type and is prefixed by the characters *SF* (e.g., field *a* is of type *SFVec3f*), while a field containing an array has its type prefixed by the characters *MF* (e.g., field *b* is of type *MFVec3f*). A field may contain a reference to one or more nodes by using the *SFNode* and *MFNode* field types.

Each object has the following common characteristics:

- a. **A type name.** Examples include *SFVec3f*, *MFCOLOR*, *SFFloat*, [Group](#), [Background](#), or [SpotLight](#).
- b. **An implementation.** The implementation of each object defines how it reacts to changes in its property values, what other property values it alters as a result of these changes, and how it effects the state of the run-time environment. This part of ISO/IEC 19775 defines the functional semantics of built-in nodes (*i.e.*, nodes with implementations that are provided by the X3D browser).

An object derived from *X3DNode* has the following additional characteristics:

- d. **Zero or more field values.** Field values are stored in the X3D file along with the nodes or fields, and encode the state of the virtual world.
- e. **Zero or more events that it can receive and send.** Each node may receive events to its fields which will result in some change to the node's state. Each node

may also generate events from its fields to report changes in the node's state. Events generated from one node can be connected to fields of other nodes to propagate these changes. This is done using the ROUTE statement in the file or through an SAI service reference.

- f. **A name.** Nodes can be named using either the DEF statement in the file or at run-time through an SAI service call. This is used by other statements to reference a specific instantiation of a node. It is also be used to locate a specific named node within the scene hierarchy.

Node implementations can come from two sources, built-in nodes and prototypes. Built-in nodes are nodes that are available to the author as specified by the applicable profile and/or component declarations. Different components define different sets of built-in nodes.

Additionally, X3D supports content extensions using prototypes. Prototypes are objects that the author creates using PROTO or EXTERNPROTO statements. These objects are written in the same declarative notation used to describe nodes in the scene graph. They add new object types to the system which are only available for the lifetime of the session into which they are loaded. Some profiles may not include support of these extension capabilities. The semantics of prototypes are discussed in [4.4.4, Prototype semantics](#), and [4.4.5, External prototype semantics](#).

Both prototypes and built-in nodes are available for instantiation using similar mechanisms. An object can be instantiated declaratively or at run-time using the SAI services specified in [Part 2 of ISO/IEC 19775](#). All prototypes inherit from the base node type [X3DPrototypeInstance](#).

#### 4.4.2.2 Field semantics

Fields define the persistent state of nodes, and values which nodes may send or receive in the form of events. X3D supports four types of access to a node's fields:

- a. *initializeOnly* access, which allows content to supply an initial value for the field but does not allow subsequent changes to its value;
- b. *inputOnly* access, which means that the node may receive an event to change the value of its field, but does not allow the field's value to be read;
- c. *outputOnly* access, which means that the node may send an event when its value changes, but does not allow the field's value to be written; and
- d. *inputOutput* access, which allows full access to the field: content may supply an initial value for the field, the node may receive an event to change the value of its field, and the node may send an event when its value changes.

An inputOutput field can receive events like an inputOnly field, can generate events like an outputOnly field, and can be stored in X3D files like an initializeOnly field. An inputOutput field named *zzz* can be referred to as '*set\_zzz*' and treated as an inputOnly, and can be referred to as '*zzz\_changed*' and treated as an outputOnly field. Within ISO/IEC 19775, fields with inputOutput access or inputOnly access are collectively referred to as *input* fields, fields with inputOutput access or outputOnly access are collectively referred to as *output* fields, and the events these fields receive and send are called *input events* and *output events*, respectively.

The initial value of an inputOutput field is its value in the X3D file, or the default value for the node in which it is contained, if a value is not specified. When an inputOutput field receives an event it shall generate an event with the same value and timestamp. The following sources, in precedence order, shall be used to determine the initial value of the inputOutput field:

- e. the user-defined value in the instantiation (if one is specified);
- f. the default value for that field as specified in the node or prototype definition.

The recommendations for naming initializeOnly fields, inputOutput fields, outputOnly fields, and inputOnly fields for built-in nodes are as follows:

- g. All names containing multiple words start with a lower case letter, and the first letter of all subsequent words is capitalized (*e.g.*, *addChildren*), with the exception of *set\_* and *\_changed*, as described below.
- h. It is recommended that all inputOnly fields have the prefix "*set\_*", with the exception of the *addChildren* and *removeChildren* fields.
- i. Certain inputOnly fields and outputOnly fields of type SFTime do not use the "*set\_*" prefix or "*\_changed*" suffix.
- j. It is recommended that all other outputOnly fields have the suffix "*\_changed*" appended, with the exception of outputOnly fields of type SFBool.

### 4.4.2.3 Interface hierarchy

Most object types derive some of their interfaces and functionality from other object types in the system. These are known as its *supertypes*, and an object is said to be *derived* from these supertypes. Likewise, these supertypes may derive their capabilities from other object types, forming a chain all the way to a small number of base types from which all the others are ultimately derived. The graph describing the relationship between all object types in the system is called the *interface hierarchy*. In this part of ISO/IEC 19775, the object hierarchy specifies conceptual relationships between objects but does not necessarily dictate actual implementation.

[Figure 4.2](#) depicts the object hierarchy for object types defined in this part of ISO/IEC 19775 for all versions. A specification of which object types are available for which versions may be found in [Annex L Version content](#).

NOTE Not all object types are supported in certain component levels, profiles or versions; refer to the individual component and profile specifications in this part of ISO/IEC 19775 for details.

```
X3DField +------ X3DArrayField +-
          +- SFBool
          +- SFColor
          +- SFColorRGBA
          +- SFDouble
          +- SFFloat
          +- SFImage
          +- SFInt32
          +- SFMatrix3d
          +- SFMatrix3f
          +- SFMatrix4d
          +- SFMatrix4f
          +- SFNode
          +- SFRotation
          +- SFString
          +- SFTime
          +- SFVec2d
          +- SFVec2f
          +- SFVec3d
          +- SFVec3f
          +- MFBool
          +- MFColor
          +- MFColorRGBA
          +- MFDouble
          +- MFFloat
          +- MFImage
          +- MFInt32
          +- MFMatrix3d
          +- MFMatrix3f
          +- MFMatrix4d
          +- MFMatrix4f
          +- MFNode
          +- MFRotation
          +- MFString
          +- MFTime
          +- MFVec2d
          +- MFVec2f
          +- MFVec3d
          +- MFVec3f
```

```

+- SFVec4d          +- MFVec4d
+- SFVec4f          +- MFVec4f

```

```

X3DBoundedObject
X3DFogObject
X3DPickableObject
X3DProgrammableShaderObject
X3DMetadataObject
X3DUrlObject

```

```

X3DNode
|
+- Contact
+- Contour2D
+- EaseInEaseOut
+- GeoOrigin (deprecated)
+- LayerSet
+- MetadataBoolean (X3DMetadataObject)*
+- MetadataDouble (X3DMetadataObject)*
+- MetadataFloat (X3DMetadataObject)*
+- MetadataInteger (X3DMetadataObject)*
+- MetadataSet (X3DMetadataObject)*
+- MetadataString (X3DMetadataObject)*
+- NurbsTextureCoordinate
+- RigidBody
+- ShaderPart (X3DUrlObject)*
+- ShaderProgram (X3DUrlObject, X3DProgrammableShaderObject)*
+- TextureProperties

```

```

|-- X3DAppearanceNode -- Appearance

```

```

|-- X3DAppearanceChildNode --
|   +- AcousticProperties
|   +- FillProperties
|   +- LineProperties
|   +- PointProperties

```

```

|   +- X3DMaterialNode -- X3DOneSidedMaterialNode -- Material
|   |   +- PhysicalMaterial
|   |   +- UnlitMaterial
|   |   +- TwoSidedMaterial (deprecated)

```

```

|   X3DMaterialNode | Material
|   X3DMaterialNode | TwoSidedMaterial

```

```

|   +- X3DShaderNode -- ComposedShader (X3DProgrammableShaderObject)*
|   |   +- PackagedShader (X3DUrlObject,
|   |   |   +- ProgramShader

```

```

X3DProgrammableShaderObject)*

```

```

|   +- X3DTextureNode -- MultiTexture

```

```

|   + X3DSingleTextureNode -- X3DEnvironmentTextureNode -

```

```

+- ComposedCubeMapTexture

```

```

+- GeneratedCubeMapTexture

```

```

+- ImageCubeMapTexture (X3DUrlObject)*

```

```

ImageTexture (X3DUrlObject)*

```

```

+- X3DTexture2DNode --

```

```

MovieTexture (X3DSoundSourceNode, X3DUrlObject)*

```

```

+-

```

```

PixelTexture

```

```

+-

```

```

ComposedTexture3D

```

```

+- X3DTexture3DNode --

```

```

ImageTexture3D (X3DUrlObject)*

```

```

+-

```

```

PixelTexture3D

```

```

+-

```

```

X3DEnvironmentTextureNode

```

```

+ ComposedCubeMapTexture

```

```

+ GeneratedCubeMapTexture

```

```

+ ImageCubeMapTexture (X3DUrlObject)*

```

```

X3DTexture2DNode

```

```

ImageTexture (X3DUrlObject)*

```

```

MovieTexture (X3DSoundSourceNode, X3DUrlObject)*

```

```

PixelTexture

```

```

X3DTexture3DNode

```

```

ComposedTexture3D

```

```

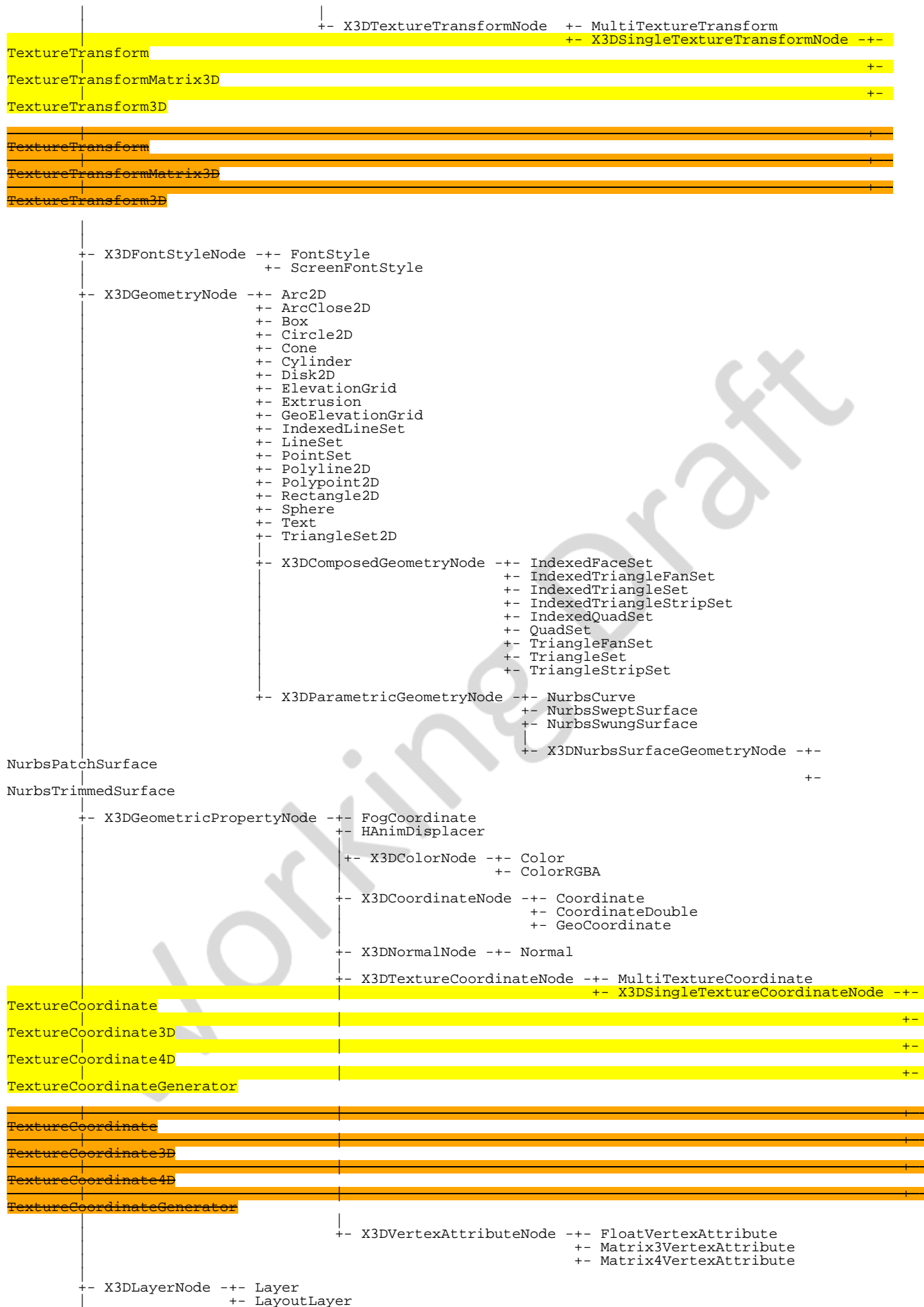
ImageTexture3D (X3DUrlObject)*

```

```

PixelTexture3D

```





```

+- X3DNBodyCollisionSpaceNode (X3DBoundedObject)* +- CollisionSpace
+- X3DNurbsControlCurveNode +- ContourPolyline2D
  +- NurbsCurve2D
+- X3DParticleEmitterNode +- ConeEmitter
  +- ExplosionEmitter
  +- PointEmitter
  +- PolylineEmitter
  +- SurfaceEmitter
  +- VolumeEmitter
+- X3DParticlePhysicsModelNode +- BoundedPhysicsModel
  +- ForcePhysicsModel
  +- WindPhysicsModel
+- X3DProtoInstance
+- X3DRigidJointNode +- BallJoint
  +- DoubleAxisHingeJoint
  +- MotorJoint
  +- SingleAxisHingeJoint
  +- SliderJoint
  +- UniversalJoint
+- X3DVolumeRenderStyleNode +- ProjectionVolumeStyle
  +- X3DComposableVolumeRenderStyle +- BlendedVolumeStyle
    +- BoundaryEnhancementVolumeStyle
    +- CartoonVolumeStyle
    +- ComposedVolumeStyle
    +- EdgeEnhancementVolumeStyle
    +- OpacityMapVolumeStyle
    +- ProjectionVolumeStyle
    +- ShadedVolumeStyle
    +- SilhouetteEnhancementVolumeStyle
    +- ToneMappedVolumeStyle
+- X3DChildNode +- BooleanFilter
  +- BooleanToggle
  +- ClipPlane
  +- CollisionCollection
  +- DISEntityManager
  +- GeoLOD (X3DBoundedObject)*
  +- HAnimHumanoid (X3DBoundedObject)*
  +- HAnimMotion
  +- Inline (X3DUrlObject, X3DBoundedObject)*
  +- LocalFog (X3DFogObject)*
  +- NurbsOrientationInterpolator
  +- NurbsPositionInterpolator
  +- NurbsSet (X3DBoundedObject)*
  +- NurbsSurfaceInterpolator
  +- RigidBodyCollection
  +- StaticGroup (X3DBoundedObject)*
  +- ViewpointGroup
+- X3DBindableNode +- Fog (X3DFogObject)*
  +- GeoViewpoint
  +- NavigationInfo
  +- ListenerPoint
  +- X3DBackgroundNode +- Background
    +- TextureBackground
  +- X3DViewpointNode +- GeoViewpoint
    +- OrthoViewpoint
    +- Viewpoint
    +- ViewpointGroup
+- X3DFollowerNode +- X3DChaserNode +- ColorChaser
  +- CoordinateChaser
  +- OrientationChaser
  +- PositionChaser
  +- PositionChaser2D
  +- ScalerChaser
  +- TexCoordChaser2D
  +- X3DDamperNode +- ColorDamper
    +- CoordinateDamper
    +- OrientationDamper
    +- PositionDamper
    +- PositionDamper2D
    +- ScalerDamper
    +- TexCoordDamper
+- X3DGroupingNode (X3DBoundedObject)* +- Anchor
  +- Billboard
  +- CADAssembly
  +- CADLayer
  +- CADPart (X3DProductStructureChildNode)*
  +- Collision (X3DSensorNode)*
  +- EspduTransform (X3DSensorNode)*
  +- GeoLocation
  +- GeoTransform
  +- Group
(X3DProductStructureChildNode)*

```



```

+- HAnimJoint
+- HAnimSegment
+- HAnimSite
+- LayoutGroup
+- LOD
+- PickableGroup (X3DPickableObject)*
+- ScreenGroup
+- Switch
+- Transform
+- X3DViewportNode --+ Viewport

+- X3DInfoNode --+ DISEntityTypeMapping
  +- GeoMetadata
  +- WorldInfo

+- X3DInterpolatorNode --+ ColorInterpolator
  +- CoordinateInterpolator
  +- CoordinateInterpolator2D
  +- GeoPositionInterpolator
  +- NormalInterpolator
  +- OrientationInterpolator
  +- PositionInterpolator
  +- PositionInterpolator2D
  +- ScalarInterpolator
  +- SplinePositionInterpolator
  +- SplinePositionInterpolator2D
  +- SplineScalarInterpolator
  +- SquadOrientationInterpolator

+- X3DLayoutNode --+ Layout

+- X3DLightNode --+ DirectionalLight
  +- PointLight
  +- SpotLight

+- X3DNBodyCollidableNode (X3DBoundedObject)* --+ CollidableOffset
  +- CollidableShape

+- X3DProductStructureChildNode --+ CADAssembly (X3DGroupingNode)*
  +- CADFace (X3DBoundedObject)*
  +- CADPart (X3DGroupingNode)*

+- X3DTextureProjectorNode --+ TextureProjectorPerspective
  +- TextureProjectorParallel

+- X3DScriptNode (X3DUrlObject)* --+ Script

+- X3DSensorNode --+ Collision (X3DGroupingNode)*
  +- CollisionSensor
  +- EspduTransform (X3DGroupingNode)*
  +- ReceiverPdu (X3DBoundedObject)*
  +- SignalPdu (X3DBoundedObject)*
  +- TimeSensor (X3DTimeDependentNode)*
  +- TransmitterPdu (X3DBoundedObject)*
  +- X3DEnvironmentalSensorNode --+ GeoProximitySensor
    +- ProximitySensor
    +- TransformSensor
    +- VisibilitySensor
  +- X3DKeyDeviceSensorNode --+ KeySensor
    +- StringSensor
  +- X3DNetworkSensorNode --+ LoadSensor
  +- X3DPickSensorNode --+ LinePickSensor
    +- PointPickSensor
    +- PrimitivePickSensor
    +- VolumePickSensor
  +- X3DPointingDeviceSensorNode --+ X3DDragSensorNode --+
    |
    |
    |
    +- X3DTouchSensorNode --+
      +-

CylinderSensor
PlaneSensor
SphereSensor
GeoTouchSensor
TouchSensor

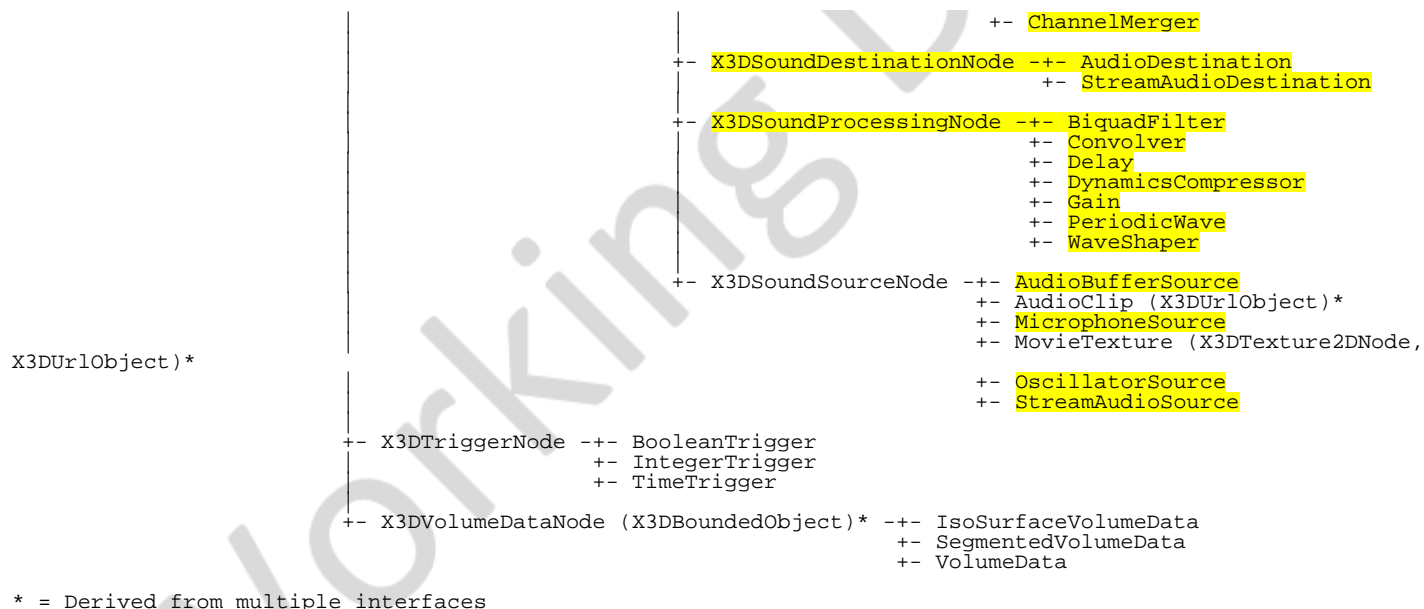
+- X3DSequencerNode --+ BooleanSequencer
  +- IntegerSequencer

+- X3DShapeNode (X3DBoundedObject)* --+ ParticleSystem
  +- Shape

+- X3DSoundNode --+ Sound
  +- SpatialSound

+- X3DTimeDependentNode --+ TimeSensor (X3DSensorNode)*
  +- X3DSoundAnalysisNode --+ Analyser
  +- X3DSoundChannelNode --+ ChannelSplitter

```



**Figure 4.2 — Interface hierarchy**

The object hierarchy defines both *abstract* interfaces and *concrete* node types. Abstract interfaces define functionality that is inherited by other interfaces and/or nodes, but are never instantiated in the scene graph as objects. Concrete node types derive from one or more abstract interfaces and may be instantiated in the scene graph. Thus, the live scene graph consists only of instances of concrete node types. Components defined in this part of ISO/IEC 19775 are required to implement the functionality of abstract interfaces only insofar as that functionality is made available via one of the derived concrete node types. [Part 2 of ISO/IEC 19775](#) defines the means by which applications may access the functionality provided in both abstract interfaces and concrete nodes via programmatic means.

The two main types of object from which most others are derived are [X3DNode](#) and [X3DField](#). Nodes are the objects used in the declarative language to form the scene graph, while fields are contained within nodes and hold the data items for nodes. Some field objects contain simple data values like integers or arrays of strings. Other field objects contain references to nodes. It is this ability of [X3DNode](#) to contain [X3DField](#), and [X3DField](#) to contain references to [X3DNode](#), that makes it possible for X3D to form scene graph hierarchies.

#### EXAMPLE

```

Transform { translation 1 2 3
  children [
    Shape {
      geometry Box { }
    }
    Group {
      children [ ... ]
    }
  ]
}
  
```

In the above example, the Transform contains a simple field, `translation`, which contains a vector of 3 numbers. It also contains a `children` field which may contain an array of other nodes. In this case it has two, a Shape and a Group. The Shape and the Group both contain fields which may have other objects as well.

Derivation makes it possible to strongly type all objects. In the above example, the children field is constrained to contain a list of objects derived from an object type

called [X3DChildNode](#). Both [Shape](#) and [Group](#) are derived (indirectly) from this object and can therefore be placed in the children field. The geometry field of Shape, on the other hand, can only contain a single node derived from [X3DGeometryNode](#). [Box](#) is derived from this object and can therefore be placed in the geometry field. But Box is not derived from [X3DChildNode](#), so it cannot be placed in the children field. Likewise, Group is not derived from [X3DGeometryNode](#) and can therefore not be placed in the geometry field.

The above example exhibits another quality of derivation. [Transform](#) is derived from [X3DGroupingNode](#) and therefore inherits its children field. This makes the specification of Transform simpler because it does not need to describe the functionality of the children field. Because it is derived from [X3DGroupingNode](#), the author knows it contains a children field which behaves like the one in Group which is also derived from [X3DGroupingNode](#).

## 4.4.2.4 Modifying objects

### 4.4.2.4.1 Routes

There are several ways to modify the fields of an object. Using one of the X3D file formats, an author can declare a set of nodes, the initial state of their fields, and interconnections between the fields called *Routes*. X3D uses an event propagation, or *dataflow* model to change the values of fields at run-time. As part of its abstract specification, the behavior of a node in response to events sent to its fields, and the conditions under which its fields send events out, is described.

EXAMPLE It is possible to create a scene with run-time behavior using only this event propagation model:

```
DEF TS TimeSensor {
  loop TRUE
  cycleInterval 5
}
DEF I PositionInterpolator {
  key [ 0 0.5 1 ]
  keyValue [ 0 -1 0, 0 1 0, 0 -1 0 ]
}
DEF T Transform {
  children [
    Shape {
      geometry Box { }
    }
  ]
}
ROUTE ts.fraction_changed TO I.set_fraction
ROUTE I.value_changed TO T.set_translation
```

This example bounces a box up and down repeatedly over a five-second interval. The TimeSensor object is defined to send an event continuously out of its `fraction` field. This event sends a floating point value which varies from 0 to 1 over a 5 second interval, as specified by the `cycleInterval`. Its `loop` field tells it to do so repeatedly. This fraction value is sent to the `fraction` field of a PositionInterpolator. This object is defined to send an event out of its `value` field whenever it receives an event on its fraction field. The value is determined by the `key` and `keyValue` fields. In this case it sends a vector whose y value varies between -1 and +1 and back again over the interval. This value is sent to the `translation` field of the Transform node. This node is defined to set the position of its children according to the value of `translation`. [4.4.8.2 Routes](#) contains more information on routing.

### 4.4.2.4.2 Modifying objects via programmatic access

The routing mechanism is simple, but is limited to changing field values of nodes, and only changes that are designed into a given node set. For greater flexibility, some

profiles provide programmatic access to objects in the system. This allows field values to be set and read, and functions to be called. Mechanisms are also provided to allow PROTO objects to be found, which in turn allows objects of that type to be instantiated.

There are two types of programmatic access in X3D: External access (EXAMPLE access from a containing HTML page or embedding native application) and Internal scripts using any of the supported scripting languages.

Programmatic access to objects is provided via *interfaces* to those objects. The interface of an object (its set of data and function properties) is specified, and is also referred to as the *object type*. An object type that represents a node is also referred to as a *node type*. Object types may be either abstract or concrete. Abstract object types are not instantiable. Instead, they are used to derive other object types or to indicate that a field may contain a node of any of the derivative node types. Concrete node types are those derived from abstract node types and are instantiable. A compliant implementation of an object's interface shall support the interface specifications as defined in [Part 2 of ISO/IEC 19775](#).

See [4.9, Application programmer interfaces](#) for additional information.

#### 4.4.2.5 Object life cycle

Nodes have a life cycle: they are created, used and eventually destroyed. A node is considered live if one or more of the following is true:

- a. The node is a root node in the scene.
- b. The node is referenced by a field of a live node.
- c. There is a reference from a live script to the node.
- d. There is an external programmatic reference to the node.

Rules b and c are applied recursively to cover the entire live scene graph.

Nodes instanced from a file are created implicitly by the browser upon encountering a node instance or upon instancing a prototype's scene graph. Nodes may also be instanced programmatically; in this case there are additional discrete steps in the node's life cycle. Refer to [Part 2 of ISO/IEC 19775](#) for more details.

#### 4.4.3 DEF/USE semantics

Node names are limited in scope to a single X3D file, prototype definition, or string submitted to either CreateX3DFromString, CreateX3DFromStream, or CreateX3DFromURL browser service or a construction for SFNodes within a script. The USE statement does not create a copy of the node. Instead, the same node is inserted into the scene graph a second time, resulting in the node having multiple parents (see [4.3.5 Transformation hierarchy](#), for restrictions on self-referential nodes).

Node names shall be unique in the context within which the associated DEF keyword occurs.

TODO: describe how, when an external environment exists,

- Abstract definition of how events can be exchanged between external environment and scene graph.
- Syntax for multiple encoding/language bindings may be defined in related specifications, e.g. updates to 19777-1 JavaScript, 19776-1 XML Encoding, and (eventually) 19776-5 JSON.
- For example, HTML5/DOM id attribute can be used for performing event callbacks using JavaScript, and thus has a similar role to DEF when events are ROUTEd.
- Editors discussion: examples should not go into an annex, will need to go into other file encodings and language bindings.
- Pending eventual ISO submission and review of those specifications, we will need example usage and some specification details publicly available to support implementation efforts.

## 4.4.4 Prototype semantics

### 4.4.4.1 Introduction

The PROTO statement defines a new node type in terms of already defined (built-in or prototyped) node types. Once defined, prototyped node types may be instantiated in the scene graph exactly like the built-in node types.

Node type names shall be unique in each X3D file. The results are undefined if a prototype is given the same name as a built-in node type or a previously defined prototype in the same scope.

### 4.4.4.2 PROTO interface declaration semantics

The prototype interface defines the fields and field access types for the new node type. The interface declaration includes the types, names and default values (for initializeOnly and inputOutput fields) for the prototype's fields.

The interface declaration may contain inputOutput field declarations, which are a convenient way of defining an initializeOnly field, inputOnly field, and outputOnly field at the same time. If an inputOutput field named *zzz* is declared, it is equivalent to separately declaring an initializeOnly field named *zzz*, an inputOnly field named *set\_zzz*, and an outputOnly field named *zzz\_changed*.

Each prototype instance can be considered to be a complete copy of the prototype, with its own field values and copy of the prototype definition. A prototyped node type is instantiated using standard node syntax. For example, the following prototype (which has an empty interface declaration):

```
PROTO Cube [ ] { Box { } }
```

may be instantiated as follows:

```
Shape { geometry Cube { } }
```

It is recommended that user-defined field names defined in PROTO interface declarations statements follow the naming conventions described in [4.4.2.2 Field semantics](#).

If an `outputOnly` field in the prototype declaration is associated with an `inputOutput` field in the prototype definition, the initial value of the associated `outputOnly` field shall be the initial value of the `inputOutput` field. If the `outputOnly` field is associated with multiple `inputOutput` fields, the results are undefined.

#### 4.4.4.3 PROTO definition semantics

A prototype definition consists of one or more nodes, nested PROTO statements, and ROUTE statements. The first node type determines how instantiations of the prototype can be used in an X3D file. An instantiation is created by filling in the parameters of the prototype declaration and inserting copies of the first node (and its scene graph) wherever the prototype instantiation occurs.

**EXAMPLE** If the first node in the prototype definition is a Material node, instantiations of the prototype can be used wherever a Material node can be used. Any other nodes and accompanying scene graphs are not part of the transformation hierarchy, but may be referenced by ROUTE statements or Script nodes in the prototype definition.

Nodes in the prototype definition may have their fields associated with the fields of the prototype interface declaration by using IS statements in the body of the node. When prototype instances are read from an X3D file, field values for the fields of the prototype interface may be given. If given, the field values are used for all nodes in the prototype definition that have IS statements for those fields. Similarly, when an input field of a prototype instance is sent an event, the event is delivered to all nodes that have IS statements for that field. When a node in a prototype instance generates an output event that has an IS statement, the event is sent to any input fields connected (via ROUTE) to the prototype instance's output field.

IS statements may appear inside the prototype definition wherever fields may appear. IS statements shall refer to fields defined in the prototype declaration. Results are undefined if an IS statement refers to a non-existent declaration. Results are undefined if the type of the field being associated by the IS statement does not match the type declared in the prototype's interface declaration. For example, it is illegal to associate an `SFColor` with an `SFVec3f`. It is also illegal to associate an `SFColor` with an `MFCColor` or *vice versa*.

Results are undefined if an IS statement:

- `inputOnly` field is associated with a `initializeOnly` field or an `outputOnly` field;
- `outputOnly` field is associated with a `initializeOnly` field or `inputOnly` field;
- `initializeOnly` field is associated with an `inputOnly` field or `outputOnly` field.

An `inputOutput` field in the prototype interface may be associated only with an `inputOutput` field in the prototype definition, but an `inputOutput` field in the prototype definition may be associated with either an `inputOutput` field, `inputOnly` field, or `outputOnly` field in the prototype interface. When associating an `inputOutput` field in a prototype definition with an `inputOnly` field or `outputOnly` field in the prototype declaration, it is valid to use either the shorthand `inputOutput` field name (e.g., *translation*) or the explicit field name (e.g., *set\_translation* or *translation\_changed*). [Table 4.4](#) defines the rules for mapping between the access types of fields in a prototype declarations and the access types for fields in its primary scene graph's nodes (*yes* denotes a legal mapping, *no* denotes an error).



**Table 4.4 — Rules for mapping PROTOTYPE declarations to node instances**

	Prototype declaration				
	<u>inputOutput</u>	<u>initializeOnly</u>	<u>inputOnly</u>	<u>outputOnly</u>	
Prototype definition	<u>inputOutput</u>	yes	yes	yes	yes
	<u>intializeOnly</u>	no	yes	no	no
	<u>inputOnly</u>	no	no	yes	no
	<u>outputOnly</u>	no	no	no	yes

Results are undefined if a field of a node in the prototype definition is associated with more than one field in the prototype's interface (*i.e.*, multiple IS statements for a field in a node in the prototype definition), but multiple IS statements for the fields in the prototype interface declaration is valid. Results are undefined if a field of a node in a prototype definition is both defined with initial values (*i.e.*, field statement) and associated by an IS statement with a field in the prototype's interface. If a prototype interface has an outputOnly field  $E$  associated with multiple outputOnly fields in the prototype definition  $ED_i$ , the value of  $E$  is the value of the field that generated the event with the greatest timestamp. If two or more of the outputOnly fields generated events with identical timestamps, results are undefined.

#### 4.4.4.4 Prototype scoping rules

Prototype definitions appearing inside a prototype definition (*i.e.*, nested) are local to the enclosing prototype. IS statements inside a nested prototype's implementation may refer to the prototype declarations of the innermost prototype.

A PROTO statement establishes a DEF/USE name scope separate from the rest of the scene and separate from any nested PROTO statements. Nodes given a name by a DEF construct inside the prototype may not be referenced in a USE construct outside of the prototype's scope. Nodes given a name by a DEF construct outside the prototype scope may not be referenced in a USE construct inside the prototype scope.

A prototype may be instantiated in a file anywhere after the completion of the prototype definition. A prototype may not be instantiated inside its own implementation (*i.e.*, recursive prototypes are illegal).

### 4.4.5 External prototype semantics

#### 4.4.5.1 Introduction

The EXTERNPROTO statement defines a new node type. It is equivalent to the PROTO statement, with two exceptions. First, the implementation of the node type is stored externally, either in an X3D file containing an appropriate PROTO statement or using some other implementation-dependent mechanism. Second, default values for fields are



not given since the implementation will define appropriate defaults.

#### 4.4.5.2 EXTERNPROTO interface semantics

The semantics of the EXTERNPROTO are exactly the same as for a PROTO statement, except that default field values are not specified locally. In addition, events sent to an instance of an externally prototyped node may be ignored until the implementation of the node is found.

Until the definition has been loaded, the browser shall determine the initial value of inputOutput fields using the following rules (in order of precedence):

- a. the user-defined value in the instantiation (if one is specified);
- b. the default value for that field type.

For outputOnly fields, the initial value on startup will be the default value for that field type. During the loading of an EXTERNPROTO, if an initial value of an outputOnly field is found, that value is applied to the field and no event is generated.

The names and types of the fields of the interface declaration shall be a subset of those defined in the implementation. Declaring a field with a non-matching name is an error, as is declaring a field with a matching name but a different type.

It is recommended that user-defined field names defined in EXTERNPROTO interface statements follow the naming conventions described in [4.4.2.2 Field semantics](#).

#### 4.4.5.3 EXTERNPROTO URL semantics

The string or strings specified after the interface declaration give the location of the prototype's implementation. If multiple strings are specified, the browser searches in the order of preference. For more information on URLs, see [9 Networking component](#).

If a URL in an EXTERNPROTO statement refers to an X3D file, the first PROTO statement found in the X3D file (excluding EXTERNPROTOS) is used to define the external prototype's definition. The name of that prototype does not need to match the name given in the EXTERNPROTO statement. Results are undefined if a URL in an EXTERNPROTO statement refers to a non-X3D file

To enable the creation of libraries of reusable PROTO definitions, browsers shall recognize EXTERNPROTO URLs that end with "#name" to mean the PROTO statement for "name" in the given X3D file. For example, a library of standard materials might be stored in an X3D file called "materials.x3dv" that looks like:

```
#X3D V3.0 utf8
PROTO Gold [] { Material { ... } }
PROTO Silver [] { Material { ... } }
...etc.
```

A material from this library **could** **might** be used as follows:

```
#X3D V3.0 utf8
EXTERNPROTO GoldFromLibrary [] "http://.../materials.x3dv#Gold"
...
Shape {
  appearance Appearance { material GoldFromLibrary {} }
  geometry ...
}
...
```

## 4.4.6 Import/Export semantics

The IMPORT feature allows authors to incorporate content defined within Inline nodes or created programmatically into the namespace of the containing file for the purposes of event routing. In contrast with external prototyping (see [4.4.5 External prototype semantics](#)), which allows access to individual fields of nodes defined as prototypes in external files, IMPORT provides access to all the fields of an externally defined node with a single statement (see [9.2.5 IMPORT statement](#)).

Importing nodes from an Inlined file is accomplished with two statements: IMPORT and EXPORT. The IMPORT statement is used in the containing file to define which nodes of an Inline are to be incorporated into the containing file's namespace. The EXPORT statement is used in the file being Inlined, to control access over which nodes within a file are visible to other files (see [9.2.6 EXPORT statement](#)). EXPORT statements are not allowed in prototype declarations.

## 4.4.7 Run-time name scope

Each X3D browser defines a run-time name scope that contains all of the root nodes currently contained by the scene graph and all of the descendant nodes of the root nodes, with the exception of nodes hidden inside another name scope. Prototypes establish a name scope and therefore nodes inside prototype instances are hidden from the parent name scope.

Each Inline node or prototype instance also defines a run-time name scope, consisting of all of the root nodes of the file referred to by the Inline node or all of the root nodes of the prototype definition, restricted as above. Other nodes or extension mechanism may be introduced which specify their own name scope.

The IMPORT feature allows nodes defined within files referenced from [Inline](#) nodes to be incorporated into the run-time name scope of the containing scene graph. Once an IMPORT statement has been encountered, the new name may be used exactly like any other node name for the purposes of routing or programmatic access (*i.e.*, may be used in ROUTE statements and accessed as a field from the Scene Access Interface). Names imported from an Inline shall be explicitly declared as exportable within the content of the inlined file, using the EXPORT statement; only names exported using the EXPORT statement are available to be imported into other run-time name scopes. The optional AS keyword allows a unique name to be assigned to the imported node in order to avoid name conflicts in the containing scene graph's run-time name scope.

Nodes created dynamically (using the X3D Scene Access Interface) are not part of any name scope, until they are added to the scene graph, at which point they become part of the same name scope of their parent node(s). A node may be part of more than one run-time name scope. A node shall be removed from a name scope when it is removed from the scene graph.

## 4.4.8 Event model

### 4.4.8.1 Events

*Events* are the primary means of generating behaviors in the X3D run-time environment. Events are used throughout X3D: driving time-based animations; handling object picking; detecting user movement and collision; changing the scene graph hierarchy. The run-time environment manages the propagation of events through the system according to a well-defined set of rules.

Nodes define input fields (*i.e.*, fields with `inputOutput` or `inputOnly` access) that trigger behavior. When a given event occurs, the node receives notification and can potentially change internal state and the value of one or more of its fields. Nodes also define output fields (*i.e.*, fields with `inputOutput` or `outputOnly` access) that are sent upon signal state changes or other occurrences within the node. Events sent to input fields and events sent by output fields are referred to collectively in ISO/IEC 19775 as *Events*.

TODO: determine whether we need to further elaborate this definition when considering external events.

#### 4.4.8.2 Routes

*Routes* allows an author to declaratively connect the output events of a node to input events of other nodes, providing a way to implement complex behaviors without imperative programming. When a routed output event is fired, the corresponding destination input event receives notification and can process a response to that change. This processing can change the state of the node, generate additional events, or change the structure of the scene graph. Routes may be created declaratively in an X3D file or programmatically via an SAI call.

Routes are not nodes. The ROUTE statement is a construct for establishing event paths between specified fields of nodes. ROUTE statements may either appear at the top level of an X3D file or inside a node wherever fields may appear. It can appear after its source or destination node and placing a ROUTE statement within a node does not associate it with that node in any way. A ROUTE statement does follow the name scoping rules as described in [4.4.7 Run-time name scope](#).

The type of the destination field shall be the same as the source type, unless a component or support level permits an extension to this rule.

Redundant routing is ignored. If an X3D file repeats a routing path, the second and subsequent identical routes are ignored. This also applies for routes created dynamically using the X3D SAI.

Nodes created through the X3D prototyping mechanism give authors an opportunity to create custom processing of incoming events. Events coming into a prototyped node through an interface field can be routed to internal nodes for processing, or routed to other interface fields for propagation outside the node. An author can also add programmatic processing logic to an interface field using the internal scripting support of the Script node.

#### 4.4.8.3 Execution model

Once a sensor or Script has generated an *initial event*, the event is propagated from the field producing the event along any ROUTEs to other nodes. These other nodes may respond by generating additional events, continuing until all routes have been

honoured. This process is called an *event cascade*. All events generated during a given event cascade are assigned the same timestamp as the initial event, since all are considered to happen instantaneously.

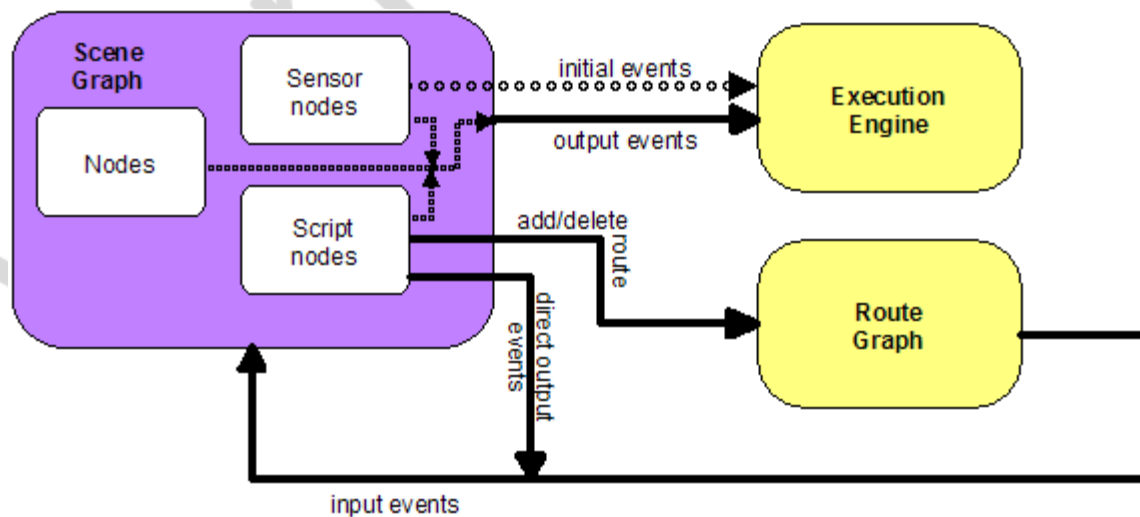
Some sensors generate multiple events simultaneously. Similarly, it is possible that asynchronously generated events **could** **might** arrive at the identical time as one or more sensor generated event. In these cases, all events generated are part of the same initial event cascade and each event has the same timestamp. The order in which the events are applied is not considered significant. Conforming X3D worlds shall be able to accommodate simultaneous events in arbitrary order.

After all events of the initial event cascade are honoured, post-event processing performs actions stimulated by the event cascade. The browser shall perform the following sequence of actions during a single timestamp:

- a. Update camera based on currently bound Viewpoint's position and orientation.
- b. Evaluate input from sensors.
- c. Evaluate routes.
- d. If any events were generated from steps b and c, go to step b and continue.
- e. If particle system evaluation is to take place, evaluate the particle systems here.
- f. If physics model evaluation is to take place, evaluate the physics model.

For profiles that support [Script](#) nodes and the Scene Access Interface, the above order may have several intermediate steps. Details are described in [29 Scripting](#) and [2\[I.19775-2\]](#).

[Figure 4.3](#) provides a conceptual illustration of the execution model.



**Figure 4.3 — Conceptual execution model**

Nodes that contain output events shall produce at most one event per field per timestamp. If a field is connected to another field via a ROUTE, an implementation shall send only one event per ROUTE per timestamp. This also applies to scripts where the rules for determining the appropriate action for sending output events are defined in [29 Scripting component](#).

#### 4.4.8.4 Loops

Event cascades may contain *loops* where an event  $E$  is routed to a node that generates an event that eventually results in  $E$  being generated again. See [4.4.8.3 Execution model](#), for the loop breaking rule that limits each event to one event per timestamp. This rule shall also be used to break loops created by cyclic dependencies between different sensor nodes.

#### 4.4.8.5 Fan-in and fan-out

*Fan-in* occurs when two or more routes have the same destination field. All events are considered to have been received simultaneously; therefore, the order in which they are processed is not considered relevant.

*Fan-out* occurs when one field is the source for more than one route. This results in sending any event generated by the field along all routes. All events are considered to have been sent simultaneously; therefore, the order in which they are processed is not considered relevant.

#### 4.4.8.6 Internal/external event passing

TODO: describe how, when an external environment exists,

- Abstract definition of when events are be exchanged between external environment and scene graph.
- Essentially the external presentation event loop must complete each render/interaction cycle before passing events to a contained X3D scene, and
- Event loop for a contained X3D scene must complete each render/interaction cycle before passing events to an external presentation.

## 4.5 Components

### 4.5.1 Overview

An X3D component is a set of related functionality consisting of various X3D objects and services as described below.

Components are specified in this standard or may be defined elsewhere. This standard specifies a set of requirements which shall be satisfied for a component to be considered an X3D component. Components may be organized into support levels as provided by the component specification. The support levels are assigned an integer identifier starting with level 1 as the simplest support level. Higher numbered support levels (if specified) should incorporate all of the functionality of lower numbered support levels. Thus, the support levels support a hierarchy of functionality.

New components may be defined either through creation of a new part to this International Standard or through registration. Functionality may be added to an already defined component by amending the appropriate part of this International Standard or through registration. Such new functionality shall be in the form of one or more new levels that augment the functionality already provided. Levels already



defined shall not be subdivided. Each such addition shall satisfy the requirements for component definition stated above.

## 4.5.2 Defining components

The following are the requirements for defining components:

- a. All node objects within a component shall be derived, either directly or indirectly, from the [X3DNode](#) class.
- b. All field objects within a component shall be derived from the [X3DField](#) or [X3DArrayField](#) classes.
- c. The names for nodes and fields shall follow the naming semantics set forth in this standard including those for scoping.

Several components are defined in this standard as shown in the [Component index](#). These components are defined in their respective parts of this International Standard. In all cases, the X3D extension mechanism may be used to add new levels to the components or may be used to define separate new components.

Each component definition is comprised of:

- d. a name for the component suitable for use in the COMPONENT statement;
- e. one or more levels starting with Level 1;
- f. a list of prerequisites for the component (each prerequisite consisting of a statement of which level in which other component is required for support of the component being defined);
- g. a conceptual description of the functionality being provided;
- h. a definition of nodes being provided with an indication of in which level each node is; and
- i. a statement of conformance for the component.

## 4.5.3 Base components

Components are specified in this standard or may be defined elsewhere. See the [Component index](#) for a list of the components of X3D which have been formally accepted by the governing body.

Each component is presented by describing the functionality to be supported. This is followed by the specification of the abstract nodes of the component, if any. Following the abstract node specifications, the concrete nodes of the component are specified. Finally, the support levels are specified.

The support levels are specified in a table in which the first column presents the number of each support level. The second column specifies the prerequisite components that are required by the particular support level for the component being specified. Each new level is presented with its prerequisites in a separate row of the table. Subsequent rows until the next new level are used to specify node support for that level. The third column specifies the nodes and other features of the component that are to be supported, in whole or in part, by the indicated support level. The fourth column specifies any constraints on the particular feature or node for the indicated

support level. For each support level  $i+1$ , all features of the previous support level shall also be supported.

In the second column, each prerequisite for a support level is listed by a component name and a support level within that component. These table entries indicate that, for the browser to claim support for that level of the component, the browser implementation shall also support the component and support level(s) listed as a prerequisite. If there are no prerequisites, the word "None" is specified.

In the third column, abstract nodes introduced at that support level are listed first followed by the concrete nodes introduced at that support level.

In the fourth column, a listing of "n/a" means "not applicable". When it is indicated that a field is "optionally supported", an X3D browser is not required to support that field. If all fields of a node are to be entirely supported, the phrase "Full support" is used.

[Table 4.5](#) is an example of the format for a support level table:

**Table 4.5 — Example support level table**

Level	Prerequisites	Nodes/Features	Support
<b>1</b>	Core 1 Networking 2		
		<i>X3DTimeDependentNode</i> (abstract)	n/a
		Node1Name	fieldi optionally supported.
		Node2Name	All fields fully supported.
<b>2</b>			
		Level 1 nodes	All fields as supported by Level 1.
		NodeName	All fields fully supported.

Any new components defined by amendment or in new parts of this International Standard shall specify their functionality using the same format.

## 4.6 Profiles

### 4.6.1 Overview

ISO/IEC 19775 supports the concept of profiles. A profile is a named collection of functionality and requirements that shall be supported in order for an implementation to conform to that profile. Profiles are defined as a set of components and levels of each component as well as the minimum support criteria for all of the objects contained



within that set.

This part of ISO/IEC 19775 defines seven profiles satisfying varying sets of requirements:

- a. Core profile (see [Annex A](#))
- b. Interchange profile (see [Annex B](#))
- c. Interactive profile (see [Annex C](#))
- d. MPEG-4 interactive profile (see [Annex D](#))
- e. Immersive profile (see [Annex E](#))
- f. Full profile (see [Annex F](#))
- g. CADInterchange profile (see [Annex H](#))

Each set of requirements is directed at supporting the needs of a particular constituency. Not all constituencies may be satisfied by the functionality represented by these profiles. Therefore, this part of ISO/IEC 19775 allows for defining additional profiles either through amendment to this part of this International Standard or by registration.

A system that conforms to a given profile supports the full set of objects and capabilities defined for that profile.

## 4.6.2 Defining profiles

A profile definition consists of the following:

- a. a name for the profile suitable for use in the PROFILE statement;
- b. an introduction defining the purpose for the profile;
- c. a list of the components and levels within those components which comprise the profile;
- d. a statement of conformance criteria for the profile;
- e. a table containing the node type set supported by the profile stating the X3D File Limit and Minimum Browser Support for each node type;
- f. a table of other limitations for the profile; and
- g. any other information specific to the profile.

## 4.6.3 Relationship between profiles and components

A profile consists of a collection of components at given support levels. A user may also supplement the predefined set of components for a given profile by specifying extra component statements (see [7.2.5.4 COMPONENT statement](#)). If the user supplies additional component declarations in addition to the components and levels defined as part of the profile, the resultant components supported shall be the union of all components and levels requested. That is, a user cannot force a lower level of component conformance onto a profile by explicitly declaring the component with a lower level of support than that defined by the profile.

A profile definition shall be internally consistent. If a profile contains components that list prerequisites that are not covered by the component levels declared for that profile,

the prerequisites shall not be automatically made available. Authors wishing to use these missing prerequisites shall explicitly declare the component and level required through the use of the COMPONENT statement.

## 4.7 Support levels

The X3D specification may be supported at varying *Levels*, or qualities of service. Any X3D component may designate a level of service by using a numbering scheme in which higher-numbered levels denote increasing qualities of service. A higher level of service may indicate any of the following:

- a. The presence (or absence) of features;
- b. Improved support for a particular feature;
- c. More rigorously defined semantics; or
- d. More stringent conformance requirements.

Note that service levels between different features do not necessarily correspond. For example, a profile may contain one component supported at level 2 and another at level 1. Any profile may combine components defined at different service levels, provided that the features interoperate properly, the behavior is deterministic (within practical limits) and the conformance requirements for that profile and components are well-defined.

## 4.8 Data encodings

The X3D run-time architecture is independent of the data encoding format. X3D content and applications can be authored in a variety of encodings, including textual (XML and Classic VRML encodings) and binary, either compressed or uncompressed. ISO/IEC 19775 contains an abstract encoding specification that defines the structure of the X3D scene: hierarchical relationships among objects, initial values for objects, and dataflow connections between objects. All concrete data encodings for X3D shall conform to this abstract specification.

Browsers and generators may support any or all of the standard encoding formats, depending on their application needs and the conformance requirements of a specific component or profile.

X3D encodings are fully specified in the parts of [ISO/IEC 19776](#).

## 4.9 Scene access interface (SAI)

X3D provides a set of application programmer interfaces (APIs), called the Scene Access Interface (SAI), that defines run-time access to the scene. Using the SAI a developer may create and destroy nodes, send events to nodes, create connections between nodes (*routes*), read or set field values in nodes, traverse the scene graph, and control the operations of the browser. Programmatic access may be *internal* (*i.e.*, used to create customized elements within the scene graph) or *external* (*i.e.*, connecting to program elements outside the scene such as in a host application such as a web browser). Internal access is supported via a special node called a [Script](#) node.

Script nodes allow developers to connect programming language functions and object classes to the scene graph. Fields of a script are automatically mapped to properties and methods of the object associated with that script. Script node code may generate events which are propagated back to the scene graph by the run-time environment. External access is supported through integration between the X3D run-time system and a variety of programming language run-time libraries.

The X3D SAI is specified as a set of language-independent services and bindings to several programming and scripting languages. A complete specification of the X3D SAI services and the component model interfaces may be found in [2. \[19775-2\]](#). The language bindings for the services defined in ISO/IEC 19775-2 are specified in [2. \[19777\]](#). Internal programmatic access is enabled through the Script node, described in [29 Scripting component](#).

TODO: determine whether we need to further elaborate this definition when considering external environments.

## 4.10 Component and profile registration

This part of ISO/IEC 19775 allows new concepts to be defined by registration of components, new levels within components, and profiles. Registration shall not be used to modify any existing component, level of a component, or profile. New functionality is registered using the established procedures of the [ISO International Register of Items<sup>1\)</sup>](#). These procedures require the proposer to supply all information for a new registered item except for the level number. The level number (if applicable) is assigned and managed by the ISO International Registration Authority for Graphical Items. Registration shall be according to the procedures in [ISO/IEC 9973](#).

<sup>1)</sup>Contact information for the ISO-designated Registration Authority for Items registered under the ISO/IEC 9973 procedures is available at the ISO Maintenance Agencies and Registration Authorities web site: [http://www.iso.org/iso/standards\\_development/maintenance\\_agencies.htm](http://www.iso.org/iso/standards_development/maintenance_agencies.htm).





## Extensible 3D (X3D) Part 1: Architecture and base components

### 25 Geospatial component

#### 25.1 Introduction

##### 25.1.1 Name

The name of this component is "Geospatial". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 25.1.2 Overview

This clause describes the Geospatial component of this part of ISO/IEC 19775. This includes how to associate real world locations to elements in the X3D world as well as specifying nodes particularly tuned for geospatial applications. [Table 25.1](#) provides links to the major topics in this clause.

**Table 25.1 — Topics**

- [25.1 Introduction](#)
  - [25.1.1 Name](#)
  - [25.1.2 Overview](#)
- [25.2 Concepts](#)
  - [25.2.1 Overview](#)
  - [25.2.2 Spatial reference frames](#)
  - [25.2.3 Specifying a spatial reference frame](#)
  - [25.2.4 Specifying geospatial coordinates](#)
  - [25.2.5 Dealing with high-precision coordinates](#)
  - [25.2.6 Geospatial navigation issues](#)
- [25.3 Node reference](#)
  - [25.3.1 GeoCoordinate](#)
  - [25.3.2 GeoElevationGrid](#)
  - [25.3.3 GeoLocation](#)
  - [25.3.4 GeoLOD](#)
  - [25.3.5 GeoMetadata](#)
  - [25.3.6 GeoOrigin](#) (deprecated)

- [25.3.7 GeoPositionInterpolator](#)
- [25.3.8 GeoProximitySensor](#)
- [25.3.9 GeoTouchSensor](#)
- [25.3.10 GeoTransform](#)
- [25.3.11 GeoViewpoint](#)
- [25.4 Support levels](#)
- [Figure 25.1 — Loading of GeoLOD levels](#)
- [Table 25.1 — Topics](#)
- [Table 25.2 — Supported spatial reference frames](#)
- [Table 25.3 — Supported earth ellipsoids](#)
- [Table 25.4 — Supported earth geoids](#)
- [Table 25.5 — GeoMetadata keywords and values](#)
- [Table 25.6 — Geospatial component support levels](#)

## 🔴 25.2 Concepts

### 25.2.1 Overview

This section contains discussions of various important concepts that are integral to the Geospatial component, providing support for geographic and geospatial applications. This support includes the ability to embed geospatial coordinates in certain X3D nodes, to support high-precision geospatial modeling, and to handle large multi-resolution terrain databases. These concepts are described below. The Geospatial component includes conventions that are defined by the Spatial Reference Model (see [ISO/IEC 18026](#)).

In total, the following nodes comprise the Geospatial component. These nodes are defined as follows.

- [GeoCoordinate](#)
- [GeoElevationGrid](#)
- [GeoLocation](#)
- [GeoLOD](#)
- [GeoMetadata](#)
- [GeoOrigin](#)
- [GeoPositionInterpolator](#)
- [GeoProximitySensor](#)
- [GeoTouchSensor](#)
- [GeoTransform](#)
- [GeoViewpoint](#)

### 25.2.2 Spatial reference frames

X3D defines an implicit Cartesian, right-handed three-dimensional coordinate system

for modeling purposes, as defined in [4.3.6 Standard units and coordinate system](#). However, most geo-referenced data are provided in a geodetic or projective spatial reference frame. A geodetic (or geographic) spatial reference frame is related to the ellipsoid used to model the earth, for example the latitude/longitude system. A projective spatial reference frame employs a projection of the ellipsoid onto some simple surface such as a cone or a cylinder, for example, the Lambert Conformal Conic (LCC) or the Universal Transverse Mercator (UTM) projections. In order to be useful to the geospatial community, X3D provides support for a number of nodes that can use spatial reference frames for modeling purposes. The spatial reference frames supported by X3D are defined in [Table 25.2](#).

**Table 25.2 — Supported spatial reference frames**

Code	Name
GD	Geodetic spatial reference frame
GC	Geocentric spatial reference frame
UTM	Universal Transverse Mercator
WM	Web Mercator

The code GDC shall be synonymous to GD, and the code GCC shall be synonymous to GC. However, these two synonyms may be subject to future deprecation. In addition to these spatial reference frames, X3D defines 23 standard ellipsoids in order to model the shape of the earth. These are all defined in [Table 25.3](#).

**Table 25.3 — Supported earth ellipsoids**

Code	Ellipsoid Name	Semi-Major Axis (metres)	Inv. Flattening ( $F^{-1}$ )
AA	Airy 1830	6377563.396	299.3249646
AM	Modified Airy	6377340.189	299.3249646
AN	Australian National	6378160	298.25
BN	Bessel 1841 (Namibia)	6377483.865	299.1528128
BR	Bessel 1841 (Ethiopia Indonesia...)	6377397.155	299.1528128
CC	Clarke 1866	6378206.4	294.9786982
CD	Clarke 1880	6378249.145	293.465
EA	Everest (India 1830)	6377276.345	300.8017
EB	Everest (Sabah & Sarawak)	6377298.556	300.8017

<b>EC</b>	Everest (India 1956)	6377301.243	300.8017
<b>ED</b>	Everest (W. Malaysia 1969)	6377295.664	300.8017
<b>EE</b>	Everest (W. Malaysia & Singapore 1948)	6377304.063	300.8017
<b>EF</b>	Everest (Pakistan)	6377309.613	300.8017
<b>FA</b>	Modified Fischer 1960	6378155	298.3
<b>HE</b>	Helmert 1906	6378200	298.3
<b>HO</b>	Hough 1960	6378270	297
<b>ID</b>	Indonesian 1974	6378160	298.247
<b>IN</b>	International 1924	6378388	297
<b>KA</b>	Krassovsky 1940	6378245	298.3
<b>RF</b>	Geodetic Reference System 1980 (GRS 80)	6378137	298.257222101
<b>SA</b>	South American 1969	6378160	298.25
<b>WD</b>	WGS 72	6378135	298.26
<b>WE</b>	WGS 84	6378137	298.257223563

Finally, X3D supports the specification of a geoid representing mean sea level. The list of geoids supported is presented in [Table 25.4](#).

**Table 25.4 — Supported earth geoids**

Code	Name
<b>WGS84</b>	WGS84 geoid

Internally, an X3D browser will transform all geographic coordinates into earth-fixed geocentric coordinates (*i.e.*, an  $(x,y,z)$  displacement from the center of the earth in units of length base units). This is a 3D Cartesian coordinate system that best integrates with X3D's implicit coordinate system. In addition, an offset may be applied to these geocentric coordinates if a **(deprecated)** [GeoOrigin](#) node is supplied (see [25.2.5 Dealing with high-precision coordinates](#)). The resulting coordinates are cast to single-precision and are the final values used for rendering. This process means that we provide support for increased precision around an area of interest, and also enable data specified in multiple spatial reference frames to be fused into a single context.

### 25.2.3 Specifying a spatial reference frame



All the X3D nodes that allow inclusion of geographic coordinates support a field called *geoSystem*. This field is used to specify the particular spatial reference frame that will be used for the geospatial coordinates in that node. This is an MFString field that can include a number of arguments to fully designate the spatial reference frame. Each argument appears in a separate string within the MFString array. Argument matching is case sensitive. Optional arguments may appear in any order. The following values are supported.

- **"GD"** - Geodetic spatial reference frame (latitude/longitude). An optional argument may be used to specify the ellipsoid using one of the ellipsoid codes that are defined in [Table 25.3](#). If no ellipsoid is specified, then "WE" is assumed (*i.e.*, the WGS84 ellipsoid). An optional "WGS84" string can be specified if you wish all elevations to relative to the WGS84 geoid (*i.e.*, mean sea level) (see [Table 25.4](#)); otherwise, all elevations will be relative to the ellipsoid. An example spatial reference frame definition of this format is [ "GD", "WD" ], for a geodetic spatial reference frame based upon the WGS72 ellipsoid with all elevations being relative to that ellipsoid.
- **"UTM"** - Universal Transverse Mercator. One further required argument must be supplied for UTM in order to specify the zone number (1..60). This is given in the form "Z<n>", where <n> is the zone number. An optional argument of "S" may be supplied in order to specify that the coordinates are in the southern hemisphere (otherwise, northern hemisphere will be assumed). A further optional argument may be used to specify the ellipsoid using one of the ellipsoid codes that are defined in [Table 25.3](#). If no ellipsoid is specified, then "WE" is assumed (*i.e.*, the WGS84 ellipsoid). An optional "WGS84" string can be specified if you wish all elevations to relative to the WGS84 geoid (*i.e.*, mean sea level (see [Table 25.4](#))); otherwise, all elevations will be relative to the ellipsoid. An example spatial reference frame definition of this format is [ "UTM", "Z10", "S", "GD" ], for a southern hemisphere UTM spatial reference frame in zone 10 with all elevations being with respect to mean sea level.
- **"GC"** - Earth-fixed Geocentric with respect to the WGS84 ellipsoid. No additional arguments are supported. An example spatial reference frame definition of this format is [ "GC" ].
- **"WM"** - Web Mercator projection used for all web mapping (slippy maps). An example spatial reference frame definition of this format is [ "WM" ].

If no *geoSystem* field is specified, the default value is [ "GD", "WE" ].

## 25.2.4 Specifying geospatial coordinates

Once the spatial reference frame has been defined, a single geographic coordinate is specified as an SFVec3d. Lists of geographic coordinates are encoded as an MFVec3d. The meaning of each component value depends upon the particular spatial reference frame that was defined via the *geoSystem* field in the same node. Given the following *geoSystem* definitions, the meaning of each component is defined as follows.

- **GD:** (<latitude>, <longitude>, <elevation>) or (<longitude>, <latitude>, <elevation>). The order of latitude and longitude is controlled by the *geoSystem*

field. If "latitude\_first" is specified, the order is latitude then longitude. If "longitude\_first" is specified, the order is longitude then latitude. If neither is specified, "latitude\_first" is the default. Elevation is always specified third. Latitude and longitude are given in units of angle base units. The following assumes an angle base unit of degrees. If a UNIT statement for angle base units has been provided, the following values for latitude and/or longitude should be suitable converted to that angle base units. Latitude is in the range  $-90..+90$ , and longitude can be in the range  $-180..+180$  or  $0..360$  (0 deg longitude is the same point in both cases). Longitudinal values are relative to the Greenwich Prime Meridian. Elevation is given in units of length base units above the ellipsoid (the default) or above the WGS84 geoid (if you supplied the "WGS84" parameter in the *geoSystem* field).

EXAMPLE (37.4506, -122.1834, 0) is the latitude/longitude coordinate for Menlo Park, California, USA.

- **UTM:** (<northing>, <easting>, <elevation>) or (<easting>, <northing>, <elevation>). The order of northing and easting is controlled by the *geoSystem* field. If "northing\_first" is specified, the order is northing then easting. If "easting\_first" is specified, the order is easting then northing. If neither is specified, "northing\_first" is the default. Elevation is always specified third. Northings, eastings, and elevation are all given in units of length base units. The zone of the coordinate, and whether it is in the southern hemisphere, are defined in the *geoSystem* string. Elevation is given with reference to the ellipsoid (the default) or the WGS84 geoid (if the "WGS84" parameter is specified in the *geoSystem* field).

EXAMPLE (4145173, 572227, 0) is the zone 10 northern hemisphere UTM coordinate for Menlo Park, California, USA.

- **GC:** (<x>, <y>, <z>). These values are all given in units of metres. The coordinate represents an offset from the center of the planet, based upon the WGS84 ellipsoid. The z-axis passes through the poles while the x-axis cuts through the latitude/longitude coordinate (0,0) degrees.

EXAMPLE (-2700301, -4290762, 3857213) is the geocentric coordinate for Menlo Park, California, USA.

- **WM:** (<x>, <y>, <elevation>). These values are all given in units of metres. The x and y values represent Web Mercator coordinates with an origin at 0 degrees latitude and longitude, based upon the WGS84 ellipsoid. The elevation is also with respect to the WGS84 ellipsoid.

EXAMPLE (-13601393.87, 4502102.12, 0) is the Web Mercator coordinate for Menlo Park, California, USA.

## 25.2.5 Dealing with high-precision coordinates

Most computer graphics systems, including X3D, use single-precision floating point values to model and render all geometry. This is a natural design constraint since computer graphics typically deals with small screens (up to around 1600 x 1280 pixels),

and locally bounded regions. As a result, there is no need to use double-precision values because any increases in accuracy that it brings would be lost in sub-pixel noise.

However, single-precision is insufficient to model data over the range of the earth at accurate ground resolutions. With only 23 bits of mantissa, a coordinate can be accurate to only one part in 8 million ( $2^{23}-1$ ); or about 6 or 7 decimal digits of precision. Since the equatorial radius of the earth (considered as an example planetary body) is 6,378,137 m (under the WGS84 ellipsoid), it is not possible to achieve resolutions better than around 0.8 metres using single-precision floating point numbers ( $6,378,137 / 8,388,607 = 0.8$ ). Below this threshold, various floating point rounding artifacts such as vertices coalescing and camera jitter will occur.

This geo-referencing problem is one avoided by establishing a geo-referenced local coordinate system (LCS). An absolute geo-referenced location is defined as the origin of the LCS. This becomes the reference point that correlates to the X3D world's (0,0,0) origin. Any subsequent geospatial locations are translated into X3D's Cartesian coordinate system relative to this LCS origin. Moreover, by allowing the user to define these local frames easily, the creator of the geo-referenced data uses the accuracy of a single-precision floating point representation by creating X3D worlds of only limited local extent. This is the purpose of the *GeoOrigin* node as specified via the *geoOrigin* field of the geographic X3D nodes. The *GeoOrigin* node and all *geoOrigin* fields are often unnecessary deprecated since browsers can automatically provide local origins as necessary).

To illustrate this concept, imagine an example where the [GeoOrigin](#) is specified as (310385.0 e, 4361550.0 n, 0 m, zone 13) in UTM coordinates. This may be transformed to a double-precision geocentric coordinate of (-1459877.12, -4715646.92, 4025213.19). Then a supplied absolute UTM coordinate of (310400.0 e, 4361600.0 n, 0 m, zone 13) may be transformed internally to a geocentric coordinate of (-1459854.51, -4715620.48, 4025252.11). Finally, this absolute geocentric coordinate can be transformed to a single-precision local Cartesian coordinate system by subtracting the *GeoOrigin* location to give (22.61, 26.44, 38.92), which is within single-precision range.

## 25.2.6 Geospatial navigation issues

There are a number of navigation issues that are specific to the task of browsing large geographic areas. One important issue is addressed here, that of elevation scaled velocity.

The velocity at which users can navigate around a world should depend upon their height above the terrain.

**EXAMPLE** When flying over the coast at a height of 100 metres above the terrain, a velocity of 100 metres per second could might be considered relatively fast. However, when approaching the earth from outer space, a velocity of 100 metres per second would be intolerably slow. Creators of geographic visualization systems have therefore learned to scale the velocity of the user's navigation in an attempt to maintain a constant pixel flow across the screen. A simple linear relationship between velocity and the user's elevation above an ellipsoid such as WGS84 often provides an acceptable and easily computable solution to this problem. This behavior is addressed by the [GeoViewpoint](#) node.

## 25.3 Node reference

### 25.3.1 GeoCoordinate

```
GeoCoordinate : X3DCoordinateNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFVec3d [in,out] point [] (-∞,∞)
  SFNode [] geoOrigin NULL [GeoOrigin] (deprecated)
  MFString [] geoSystem ["GD", "WE"] [see 25.2.3]
}
```

The `GeoCoordinate` node specifies a list of coordinates in a spatial reference frame. It is used in the `coord` field of vertex-based geometry nodes including [IndexedFaceSet](#), [IndexedLineSet](#), and [PointSet](#).

The `geoOrigin` field is used to specify a local coordinate frame for extended precision as described in [25.2.5 Dealing with high-precision coordinates](#).

The `geoSystem` field is used to define the spatial reference frame and is described in [25.2.3 Specifying a spatial reference frame](#).

The `point` array is used to contain the actual geospatial coordinates and should be provided in a format consistent with that specified for the particular `geoSystem` (see above). The geospatial coordinates are transparently transformed into a geocentric, curved-earth representation. For example, this would allow a geographer to create a X3D world where all coordinates are specified in terms of latitude, longitude, and elevation.

### 25.3.2 GeoElevationGrid

```
GeoElevationGrid : X3DGeometryNode {
  MFDouble [in] set_height
  SFNode [in,out] color NULL [X3DColorNode]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFNode [in,out] normal NULL [X3DNormalNode]
  SFNode [in,out] texCoord NULL [X3DTextureCoordinateNode]
  SFFloat [in,out] yScale 1.0 [0,∞)
  SFBool [] ccw TRUE
  SFBool [] colorPerVertex TRUE
  SFDouble [] creaseAngle 0 [0,∞)
  SFVec3d [] geoGridOrigin 0 0 0 (-∞,∞)
  SFNode [] geoOrigin NULL [GeoOrigin] (deprecated)
  MFString [] geoSystem ["GD", "WE"] [see 25.2.3]
  MFDouble [] height [0 0] (-∞,∞)
  SFBool [] normalPerVertex TRUE
  SFBool [] solid TRUE
  SFInt32 [] xDimension 0 (0,∞)
  SFDouble [] xSpacing 1.0 [0,∞)
  SFInt32 [] zDimension 0 (0,∞)
  SFDouble [] zSpacing 1.0 [0,∞)
}
```

The `GeoElevationGrid` node specifies a uniform grid of elevation values within some spatial reference frame. These are then transparently transformed into a geocentric, curved-earth representation. For example, this would allow a geographer to create a height field where all coordinates are specified in terms of latitude, longitude, and elevation.

The fields `color`, `colorPerVertex`, `texCoord`, `normal`, and `normalPerVertex` all have the same meaning as for [ElevationGrid](#) (see [13.3.4 ElevationGrid](#)). Similarly, if necessary, tessellation is applied as specified in [13.3.4 ElevationGrid](#).

The `ccw`, `solid`, and `creaseAngle` fields are described in [11.2.3 Common geometry fields](#).

The *geoOrigin* field is used to specify a local coordinate frame for extended precision as described in [25.2.5 Dealing with high-precision coordinates](#).

The *geoSystem* field is used to define the spatial reference frame and is described in [25.2.3 Specifying a spatial reference frame](#).

The *geoGridOrigin* field specifies the geographic coordinate for the south-west corner (bottom-left) of the dataset. This value should be specified as described in [25.2.4 Specifying geospatial coordinates](#).

The *height* array contains  $xDimension \times zDimension$  floating point values that represent elevation above the ellipsoid or the geoid, as appropriate. These values are given in row-major order from west to east, south to north. When the *geoSystem* is "GD", *xSpacing* refers to the number of units of longitude in angle base units between adjacent height values and *zSpacing* refers to the number of units of latitude in angle base units between vertical height values. When the *geoSystem* is "UTM", *xSpacing* refers to the number of eastings (length base units) between adjacent height values and *zSpacing* refers to the number of northings (length base units) between vertical height values.

EXAMPLE If  $xDimension = n$  and the grid spans  $d$  units horizontally, the *xSpacing* value should be set to:

$$d / (n-1).$$

The *yScale* value can be used to produce a vertical exaggeration of the data when it is displayed. By default, this value is 1.0 (no exaggeration). If this value is set greater than 1.0, all heights will appear larger than actual.

### 25.3.3 GeoLocation

```
GeoLocation : X3DGroupingNode {
  MFNode [in]  addChilden      [X3DChildNode]
  MFNode [in]  removeChildren  [X3DChildNode]
  MFNode [in,out] children     [] [X3DChildNode]
  SFBool [in,out] bboxDisplay  FALSE
  SFVec3d [in,out] geoCoords   0 0 0 (-∞,∞)
  SFNode [in,out] metadata     NULL [X3DMetadataObject]
  SFBool [in,out] visible      TRUE
  SFNode []   geoOrigin        NULL [GeoOrigin] (deprecated)
  MFString []  geoSystem       ["GD","WE"] [see 25.2.3]
  SFVec3f []   bboxCenter      0 0 0 (-∞,∞)
  SFVec3f []   bboxSize        -1 -1 -1 [0,∞) or -1 -1 -1
}
```

The GeoLocation node provides the ability to geo-reference any standard X3D model. That is, to take an ordinary X3D model, contained within the *children* field of the node, and to specify its geospatial location. This node is a grouping node that can be thought of as a Transform node. However, the GeoLocation node specifies an absolute location, not a relative one, so content developers should not nest GeoLocation nodes within each other.

The *geoOrigin* field is used to specify a local coordinate frame for extended precision as described in [25.2.5 Dealing with high-precision coordinates](#).

The *geoSystem* field is used to define the spatial reference frame and is described in [25.2.3 Specifying a spatial reference frame](#).

The geometry of the nodes in *children* is to be specified in units of metres in X3D



coordinates relative to the location specified by the *geoCoords* field. The *geoCoords* field should be provided in the format described in [25.2.3 Specifying a spatial reference frame](#).

The *geoCoords* field can be used to dynamically update the geospatial location of the model; for example, an event **could** **might** be sent from a [GeoPositionInterpolator](#) node.

In addition to placing a X3D model at the correct geospatial location, the *GeoLocation* node will also adjust the orientation of the model appropriately. The standard X3D coordinate system specifies that the +Y axis = up, +Z = out of the screen, and +X = towards the right. The *GeoLocation* node will set the orientation so that the +Y axis is the up direction for that local area (the normal to the tangent plane on the ellipsoid), -Z points towards the north pole, and +X is east.

### 25.3.4 GeoLOD

```

GeoLOD : X3DChildNode. X3DBoundedObject {
  SFBool [in out] bboxDisplay FALSE
  SFNode [in.out] metadata NULL [X3DMetadataObject]
  SFBool [in out] visible TRUE
  MFNode [out] children [X3DChildNode]
  SFInt32 [out] level_changed
  SFVec3d [] center 0 0 0 (-∞,∞)
  MFString [] child1Url [] [URI]
  MFString [] child2Url [] [URI]
  MFString [] child3Url [] [URI]
  MFString [] child4Url [] [URI]
  SFNode [] geoOrigin NULL [GeoOrigin] deprecated
  MFString [] geoSystem ["GD","WE"] [see 25.2.3]
  SFFloat [] range 10 [0,∞)
  MFString [] rootUrl [] [URI]
  MFNode [] rootNode [] [X3DChildNode]
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}

```

The *GeoLOD* node provides a terrain-specialized form of the [LOD](#) node. It is a grouping node that specifies two different levels of detail for an object using a tree structure, where 0 to 4 children can be specified, and also efficiently manages the loading and unloading of these levels of detail.

The level of detail is switched depending upon whether the user is closer or farther than *range* length base units from the geospatial coordinate *center*. The *center* field should be specified as described in [25.2.4 Specifying geospatial coordinates](#).

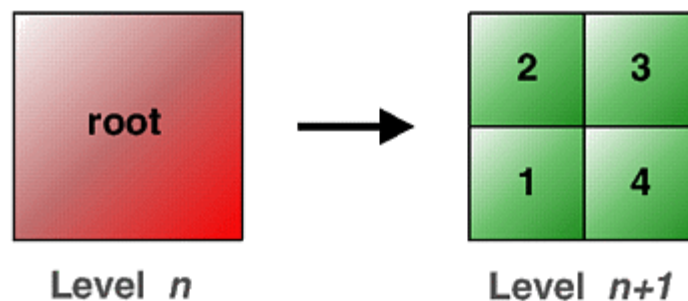
The *geoOrigin* field is used to specify a local coordinate frame for extended precision as described in [25.2.5 Dealing with high-precision coordinates](#).

The *geoSystem* field is used to define the spatial reference frame and is described in [25.2.3 Specifying a spatial reference frame](#).

**The *visible* field specifies whether or not the content within a node is visually displayed. The value of this field has no effect on animation behaviors, collision behaviors, event passing, or other non-visual characteristics.**

When the user is outside the specified range, only the geometry for *rootUrl* or *rootNode* are displayed. When the viewer enters the specified range, this geometry is replaced with the contents of the four children files defined by *child1Url* through *child4Url*. The four children files are loaded into memory only when the user is within the specified range. Similarly, these are unloaded from memory when the user leaves this range. [Figure 25.1](#) illustrates this process. The child URLs shall be arranged in the same order

as in the figure; *i.e.*, *child1Url* represents the bottom-left quadtree child. It is valid to specify less than four child URLs; in which case, the GeoLOD should switch to the children nodes when all of the specified URLs have been loaded. This latter feature can support tree structures other than quadtrees, such as binary trees.



**Figure 25.1 — Loading of GeoLOD levels**

The *rootUrl* and *rootNode* fields provide two different ways to specify the geometry of the root tile. You may use one or the other. The *rootNode* field lets you include the geometry for the root tile directly within the X3D file; whereas the *rootUrl* field lets you specify a URL for a file that contains the geometry. The result of specifying a value for both of these fields is undefined.

The *children* field is used to expose a portion of the scene graph for the currently loaded set of nodes. The value returned as an event is an MFNode containing the currently selected tile. This will either be the node specified by the *rootNode* field or the nodes specified by the *child1Url*, *child2Url*, *child3Url*, and *child4Url* fields. The GeoLOD node shall generate a new *children* output event each time the scene graph is changed (EXAMPLE whenever nodes are loaded or unloaded). When the new children event is generated, the GeoLOD node shall also generate a *level\_changed* event with value 0 or 1, where 0 indicates the *rootNode* field and 1 indicates the nodes specified by the *child1Url*, *child2Url*, *child3Url*, and *child4Url* fields.

The GeoLOD node may optionally be implemented with support for a cache of the most recent nodes that have been loaded. This cache should be global across all GeoLOD instances in a scene. This will improve performance when navigating large terrain models by avoiding excessive loading and unloading when a user makes small changes in viewpoint.

### 25.3.5 GeoMetadata

```
GeoMetadata : X3DInfoNode, X3DUrlObject {
  MFNode [in,out] data []
  SFString [in,out] description ""
  SFBool [in,out] load TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTime [in,out] refresh 0.0 [0,∞)
  MFString [in,out] summary []
  MFString [in,out] url [] [URI]
}
```

The GeoMetadata node supports the specification of metadata describing any number of geospatial nodes. This is similar to a [WorldInfo](#) node, but specifically for describing geospatial information.



There are a number of standards and representations for geospatial metadata. Rather than adopt any particular standard, the purpose of the GeoMetadata node is to provide links to any of these complete metadata descriptions, with the option to also supply a short, human-readable summary. More specific metadata can be specified using the *metadata* field available in each node.

The *url* field is used to specify a hypertext link to an external, complete metadata description. Multiple URL strings can be specified in order to provide alternative locations for the same metadata information. The summary field may be used to specify the format of the metadata in the case where this cannot be deduced easily.

The *summary* string array contains a set of keyword/value pairs, with each keyword and its subsequent value contained in a separate string; *i.e.*, there should always be an even (or zero) number of strings. This provides a simple, extensible mechanism to include metadata elements that are human-readable and easy to parse. [Table 25.5](#) specifies a number of keywords and the format that should be used to describe their values. If an unknown keyword is found, it (and its associated value) are ignored.

**Table 25.5 — GeoMetadata keywords and values**

Keyword	Value
title	A name to succinctly identify the dataset to user. For example, "San Francisco, CA".
description	A brief textual description or summary of the content of the dataset. For example, "LANDSAT 7 satellite imagery taken over northern Scotland".
coordinateSystem	The spatial reference frame used to represent the data ( <i>e.g.</i> , GD, UTM, or LCC). The list of valid codes that can be used in this field are defined in <a href="#">ISO/IEC 18026</a> . In the case of UTM, the zone number should also be specified in the format "UTM Zx", where the zone number is in the range [1,60]. For example, "UTM Z11".
horizontalDatum	The name of the geodetic datum. The list of valid codes that can be used in this field are defined in <a href="#">ISO/IEC 18026</a> . For example, "W84".
verticalDatum	The name of the vertical datum (geoid). The list of valid codes that can be used in this field are defined in <a href="#">ISO/IEC 18026</a> . For example, "W84".
ellipsoid	The name of the geodetic ellipsoid. The list of valid codes that can be used in this field are defined in <a href="#">ISO/IEC 18026</a> . For example, "WE".
extent	The bounding coordinates for the dataset given in spatial reference frame specified by the <i>coordinateSystem</i> keyword. These are provided in the order eastmost, southmost, westmost, northmost. An example for GD is: "-180.0 -90.0 180.0 90.0".

resolution	The resolution, or ground sample distance, given in units of length base units. For example, "30".
originator	A string defining the originator of the data, for example the author, agency, organization, publisher, etc. For example, "John Doe, Any Corporation, Some Town, Some Country"
copyright	Any appropriate copyright declaration that pertains to the data. For example, "(c) Copyright 2000, Any Corporation. All rights reserved. Freely distributable."
date	A single date/time, or a date/time range, defining the valid time period to which the data pertains. Dates are specified in the format "YYYY MM DD [HH:MM]". Years in the current time period should be specified using four digits ( <small>EXAMPLE</small> "1999" or "2001"). Years can have other than four digits and can be negative. A date range is specified by supplying two values separated by a "-" (hyphen) character. An optional time can be supplied should this level of accuracy be required. Times are to be specified in 24-hour format with respect to GMT. For example, "1999 01 01 00:00 - 1999 12 31 23:59".
metadataFormat	A string that specifies the format of the external metadata description specified by the <i>url</i> field of the GeoMetadata node. For example, "FGDC", "ISO TC211", "CEN TC287", or "OGC".
dataUrl	A hypertext link to the source data used to create the X3D node(s) to which this metadata pertains. Multiple <i>dataUrl</i> keyword/value pairs can be specified in order to provide alternative locations for the same source data. For example, "http://www.foo.bar/data/sf1".
dataFormat	A free-text string that describes the format of the source data used to create the X3D node(s) to which this metadata pertains. This refers to the source data specified by the <i>dataUrl</i> keyword (if present). For example, "USGS 5.5-min DEM".

The data field is used to list all of the other nodes in a scene by DEF name that reference the data described in the GeoMetadata node. For example, if the GeoMetadata node is describing a height field grid, the appropriate [GeoElevationGrid](#) node **could** **might** be included inside the data field. The nodes in the data field are not rendered, so DEF/USE can be used in order to first describe them and then to use them in the scene graph. This approach allows associating multiple data nodes with a single GeoMetadata node, specifying multiple GeoMetadata nodes within a single scene, and also provides a mechanism to easily locate all of the data that pertain to any particular metadata entry. If the data field is not specified, it is assumed that the GeoMetadata node pertains to the entire scene.

### 25.3.6 GeoOrigin **(deprecated)**

```
GeoOrigin : X3DNode {
  SFVec3d [in,out] geoCoords 0 0 0 (-∞,∞)
  SFNode [in,out] metadata NULL [X3DMetadataObject]
```

```
MFString []   geoSystem ["GD","WE"] [see 25.2.3]
SFBool  []   rotateYUp FALSE
}
```

GeoOrigin node usage is discouraged because different models built with separate GeoOrigin nodes cannot be easily combined. GeoOrigin is still needed in some situations to achieve correct visual fidelity. Relevant GeoOrigin examples may include fine positioning in a global context, to aid deployment to handheld devices which may use lower-precision arithmetic in their graphics pipelines.

GeoOrigin node usage is deprecated and its use is discouraged. The presence of a GeoOrigin node is tolerated but can be ignored in X3D scenes having version 3.0, 3.1 or 3.2. GeoOrigin node is not allowed in X3D scenes having version 3.3 or higher.

The GeoOrigin node defines an absolute geospatial location and an implicit local coordinate frame against which geometry is referenced. This node is used to translate from geographical coordinates into a local Cartesian coordinate system which can be managed by the X3D browser.

The *geoCoords* field is used to specify a local coordinate frame for extended precision as described in [25.2.5 Dealing with high-precision coordinates](#).

The *geoSystem* field is used to define the spatial reference frame and is described in [25.2.3 Specifying a spatial reference frame](#).

The *rotateYUp* field is used to specify whether coordinates of nodes that use this GeoOrigin are to be rotated such that their up direction is aligned with the X3D Y axis. The default behavior is to not perform this operation. This means that the local up direction will depend upon the location of the GeoOrigin with respect to the planet surface. The principal reason for performing the rotation is to ensure that standard navigation modes such as "FLY" and "WALK", which assume that +Y = up, will function correctly. Specifying *rotateYUp* to be `TRUE` may incur an extra computational overhead in order to perform the rotation for each point.

### 25.3.7 GeoPositionInterpolator

```
GeoPositionInterpolator : X3DInterpolatorNode {
  SFFloat [in]  set_fraction   (-∞,∞)
  MFFloat [in,out] key         []   (-∞,∞)
  MFVec3d [in,out] keyValue   []
  SFNode [in,out] metadata    NULL   [X3DMetadataObject]
  SFVec3d [out]  geovalue_changed
  SFVec3f [out]  value_changed
  SFNode []     geoOrigin     NULL   [GeoOrigin] deprecated
  MFString []   geoSystem    ["GD","WE"] [see 25.2.3]
}
```

The GeoPositionInterpolator node provides an interpolator capability where key values are specified in geographic coordinates and the interpolation is performed within the specified spatial reference frame.

The *geoOrigin* field is used to specify a local coordinate frame for extended precision as described in [25.2.5 Dealing with high-precision coordinates](#).

The *geoSystem* field is used to define the spatial reference frame and is described in [25.2.3 Specifying a spatial reference frame](#).

The fields *key*, *set\_fraction*, and *value\_changed* have the same meaning as in the

PositionInterpolator node.

The *keyValue* array is used to contain the actual coordinates and should be provided in a format consistent with that specified for the particular *geoSystem*.

The *geovalue\_changed* field outputs the the interpolated coordinate in the spatial reference frame specified by *geoSystem*. This can be passed to other GeoX3D nodes that support a field of this form (e.g., [GeoViewpoint](#) and [GeoLocation](#)).

### 25.3.8 GeoProximitySensor

```
GeoProximitySensor : X3DEnvironmentalSensorNode {
  SFBool [in,out] enabled TRUE
  SFVec3d [in,out] geoCenter 0 0 0 (-∞,∞) (deprecated as of vs. 3.3)
  SFVec3d [in,out] center 0 0 0 (-∞,∞) (starting with vs. 3.3)
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFVec3f [in,out] size 0 0 0 [0,∞)
  SFVec3f [out] centerOfRotation_changed
  SFTime [out] enterTime
  SFTime [out] exitTime
  SFVec3d [out] geoCoord_changed
  SFBool [out] isActive
  SFRotation [out] orientation_changed
  SFVec3f [out] position_changed
  SFNode [] geoOrigin NULL [GeoOrigin] (deprecated)
  MFString [] geoSystem ["GD","WE"] [see 25.2.3]
}
```

The *GeoProximitySensor* node generates events when the viewer enters, exits, and moves within a region in space (defined by a box).

A *GeoProximitySensor* node generates *isActive* events as the viewer enters and exits the rectangular box defined by its *geoCenter* and *size* fields. This box is oriented tangent to the ellipsoid in a local coordinate system. Starting with version 3.3, the *geoCenter* field is renamed *center*.

The fields *geoSystem* and *geoOrigin* are described in [25.2.3 Specifying a spatial reference frame](#) and [25.2.5 Dealing with high-precision coordinates](#), respectively.

The *geoCoord\_changed* generates an event that returns the geospatial coordinates of the viewer's position in the spatial reference frame specified by *geoSystem* for the viewer's position whenever a *position\_changed* event is generated. The *geoCoord\_changed* value corresponds to the world position returned by *position\_changed*.

The remaining fields are defined in [22.4.1 ProximitySensor](#).

### 25.3.9 GeoTouchSensor

```
GeoTouchSensor : X3DTouchSensorNode {
  SFString [in,out] description ""
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFVec3f [out] hitNormal_changed
  SFVec3f [out] hitPoint_changed
  SFVec2f [out] hitTexCoord_changed
  SFVec3d [out] hitGeoCoord_changed
  SFBool [out] isActive
  SFBool [out] isOver
  SFTime [out] touchTime
  SFNode [] geoOrigin NULL [GeoOrigin] (deprecated)
  MFString [] geoSystem ["GD","WE"] [see 25.2.3]
}
```

A *GeoTouchSensor* node tracks the location and state of a pointing device and detects

when the user points at geometry contained by the parent group of the GeoTouchSensor. This node provides the same functionality as a [TouchSensor](#) but also provides the ability to return the geographic coordinate under the pointing device.

The *description* field in the GeoTouchSensor node specifies a textual description of the GeoTouchSensor node. This may be used by browser-specific user interfaces that wish to present users with more detailed information about the GeoTouchSensor.

A GeoTouchSensor can be enabled or disabled by sending an event of value `TRUE` or `FALSE` to the *enabled* field. A disabled GeoTouchSensor does not track user input or send events.

The *geoOrigin* field is used to specify a local coordinate frame for extended precision as described in [25.2.5 Dealing with high-precision coordinates](#).

The *geoSystem* field is used to define the spatial reference frame and is described in [25.2.3 Specifying a spatial reference frame](#).

The fields *hitNormal\_changed*, *hitPoint\_changed*, *hitTexCoord\_changed*, *isActive*, *isOver*, and *touchTime* all have the same meaning as in the TouchSensor node.

The *hitGeoCoord\_changed* field is generated while the pointing device is pointing towards the GeoTouchSensor's geometry (*i.e.*, when *isOver* is `TRUE`). It is a field containing the geospatial coordinate for the point of intersection between the pointing device's location and the underlying geometry. The value of the *geoSystem* string defines the spatial reference frame of the geospatial coordinate. For example, given the default *geoSystem* value of "GD", the *hitGeoCoord\_changed* field will be in the format (<latitude> <longitude> <elevation>) (see [25.2.4 Specifying geospatial coordinates](#)).

## 25.3.10 GeoTransform

```

GeoTransform : X3DGroupingNode {
  MFNode [in] addChildren [X3DChildNode]
  MFNode [in] removeChildren [X3DChildNode]
  MFNode [in,out] children [] [X3DChildNode]
  SFBool [in,out] bboxDisplay FALSE
  SFVec3d [in,out] geoCenter 0 0 0 (-∞,∞)
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFRotation [in,out] rotation 0 0 1 0 [-1,1] or (-∞,∞)
  SFVec3f [in,out] scale 1 1 1 (0,∞)
  SFRotation [in,out] scaleOrientation 0 0 1 0 [-1,1] or (-∞,∞)
  SFVec3f [in,out] translation 0 0 0 (-∞,∞)
  SFBool [in,out] visible TRUE
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
  SFNode [] geoOrigin NULL [GeoOrigin] (deprecated)
  MFString [] geoSystem ["GD","WE"] [see 25.2.3]
}

```

The GeoTransform node is a grouping node that defines a coordinate system for its children to support the translation and orientation of geometry built using GeoCoordinate nodes within the local world coordinate system. The X-Z plane of a GeoTransform coordinate system is tangent to the ellipsoid of the spatial reference frame at the location specified by the *geoCenter* field.

The *geoCenter* field specifies, in the spatial reference frame specified by the *geoSystem* field, the location at which the local coordinate system is centered.

The fields *geoSystem* and *geoOrigin* are described in [25.2.3 Specifying a spatial reference frame](#) and [25.2.5 Dealing with high-precision coordinates](#), respectively.



The remaining fields are defined in [10.4.4 Transform](#).

## 25.3.11 GeoViewpoint

```

GeoViewpoint : X3DViewpointNode {
  SFBool   [in]  set_bind
  SFVec3d  [in,out] centerOfRotation 0 0 0      (-∞,∞)
  SFString [in,out] description ""
  SFFloat  [in,out] farClippingPlane -1        -1 or (0,∞)
  SFFloat  [in,out] fieldOfView π/4           (0,π)
  SFBool   [in,out] jump TRUE
  SFNode   [in,out] metadata NULL             [X3DMetadataObject]
  SFFloat  [in,out] nearClippingPlane -1       -1 or (0,∞)
  SFRotation [in,out] orientation 0 0 1 0      (-∞,∞) or -1 1
  SFVec3d  [in,out] position 0 0 100000       (-∞,∞)
  SFBool   [in,out] retainUserOffsets FALSE
  SFTime   [out]  bindTime
  SFBool   [out]  isBound
  SFNode   []     geoOrigin NULL              [GeoOrigin] deprecated
  MFString []     geoSystem ["GD", "WE"]      [see 25.2.3]
  SFFloat  []     speedFactor 1.0             [0,∞)
}

```

The `GeoViewpoint` node allows the specification of a viewpoint in terms of a geospatial coordinate. This node can be used wherever a [Viewpoint](#) node can be used and can be combined with `Viewpoint` nodes in the same scene. The *fieldOfView*, *jump*, *description*, *set\_bind*, *bindTime*, and *isBound* fields and events have the same behavior as the standard `Viewpoint` node. When a `GeoViewpoint` node is bound, it overrides the currently bound `Viewpoint` and `NavigationInfo` nodes in the scene.

The *geoOrigin* field is used to specify a local coordinate frame for extended precision as described in [25.2.5 Dealing with high-precision coordinates](#).

The *geoSystem* field is used to define the spatial reference frame and is described in [25.2.3 Specifying a spatial reference frame](#).

The *position* is used to define the actual coordinate at which the viewpoint is to be located. It should be provided in a format consistent with that specified by *geoSystem*. There is also a *set\_position* field which can be routed from the *geovalue\_changed* field of a [GeoPositionInterpolator](#) node in order to animate the position of the `GeoViewpoint`.

The *orientation* string defines a relative orientation from the local orientation frame that is defined by the position field. By default, the orientation of the viewpoint will always be aligned such that the +Y axis is the up vector for the local area (the normal to the tangent plane on the ellipsoid), -Z points towards the north pole, and +X is east. Therefore, if a `GeoViewpoint` is created that always looked straight down, no matter where on the planetary body is being observed, an *orientation* value of [ 1 0 0 -1.57 ] is used. The *set\_orientation* field can be routed from the *value\_changed* field of an [OrientationInterpolator](#) in order to animate the orientation of the `GeoViewpoint`.

The `GeoViewpoint` node may be implemented as if there is an embedded `NavigationInfo` node that is bound and unbound with the `GeoViewpoint` node. As such, a X3D browser should internally set the *speed*, *avatarSize*, and *visibilityLimit* fields to an appropriate value for the viewpoint's elevation. The X3D browser should also continually update the speed field as the user moves in order to support elevation scaled velocity (see [25.2.6 Geospatial navigation issues](#)). It is recommended that the speed of user interaction be defined as:

$$( \text{elevation} / 10.0 ) \text{ speed base units}$$

where *elevation* is the user's elevation above the WGS84 ellipsoid in units of speed base units. It is also recommended that the same scaling factor be applied to the *avatarSize* vector.

The *speedFactor* field of the GeoViewpoint node is used as a multiplier to the elevation-based velocity that the node sets internally; *i.e.*, this is a relative value and not an absolute speed as is the case for the NavigationInfo node.

## 25.4 Support levels

The Geospatial component provides one level of support as specified in [Table 25.6](#).

**Table 25.6 — Geospatial component support levels**

Level	Prerequisites	Nodes/Features	Support
1	Core 1 Time 1 Networking 1 Grouping 3 Rendering 1 Shape 1 Geometry3D 1 Interpolator 1 Point device sensor 1 Navigation 1		
		GeoCoordinate	All fields fully supported.
		GeoElevationGrid	All fields fully supported.
		GeoLocation	All fields fully supported.
		GeoLOD	All fields fully supported.
		GeoMetadata	All fields fully supported.
		GeoOrigin (deprecated)	All fields fully supported.
		GeoPositionInterpolator	All fields fully supported.
		GeoTouchSensor	All fields fully supported.
		GeoViewpoint	All fields fully supported.



2	Core 1 Time 1 Networking 1 Grouping 3 Rendering 1 Shape 1 Geometry3D 1 Interpolator 1 Environmental device sensor 1 Navigation 1		
		All Level 1 Geospatial nodes	All fields fully supported.
		GeoProximitySensor	All fields fully supported.
		GeoTransform	All fields fully supported.





# Extensible 3D (X3D) Part 1: Architecture and base components

## Annex D

(normative)

### MPEG-4 interactive profile



#### D.1 General

This annex defines the X3D components which comprise the MPEG-4 interactive profile. This includes not only the nodes which shall be supported but also which fields in the supported nodes may be ignored.

This profile is targeted towards:

- providing the base point of interoperability with the MPEG-4 standard (see [2. \[14496-1\]](#)),
- implementing a lightweight playback engine that supports rich graphics and interactivity,
- possible implementation in a low-footprint engine requiring limited navigation and environmental sensor control (EXAMPLE an applet or small browser plug-in), and
- allowing a broader range of implementations by eliminating some complexity of a complete X3D implementation.

#### D.2 Topics

[Table D.1](#) provides links to the major topics in this annex.

**Table D.1 — Topics**

- |  |
|--|
| <ul style="list-style-type: none"><li>• <a href="#">D.1 General</a></li><li>• <a href="#">D.2 Topics</a></li><li>• <a href="#">D.3 Component support</a></li><li>• <a href="#">D.4 Conformance criteria</a></li><li>• <a href="#">D.5 Node set</a></li></ul> |
|--|



### [D.6 Other limitations](#)

- [Table D.1 — Topics](#)
- [Table D.2 — Components and levels](#)
- [Table D.3 — Nodes for conforming to the MPEG-4 interactive profile](#)
- [Table D.4 — Other limitations](#)

## D.3 Component support

[Table D.2](#) lists the components and their levels which shall be supported in the MPEG-4 interactive profile. Tables D.2 and D.3 describe limitations on required support for nodes and fields contained within these components.

**Table D.2 — Components and levels**

Component	Level	Reference
Core	1	<a href="#">7.5 Support levels</a>
Time	1	<a href="#">8.5 Support levels</a>
Networking	2	<a href="#">9.5 Support levels</a>
Grouping	2	<a href="#">10.5 Support levels</a>
Rendering	1	<a href="#">11.5 Support levels</a>
Shape	1	<a href="#">12.5 Support levels</a>
Geometry3D	2	<a href="#">13.4 Support levels</a>
Lighting	2	<a href="#">17.5 Support levels</a>
Texturing	1	<a href="#">18.5 Support levels</a>
Interpolation	2	<a href="#">19.5 Support levels</a>
Pointing device sensor	1	<a href="#">20.5 Support levels</a>
Environmental sensor	1	<a href="#">22.5 Support levels</a>
Navigation	1	<a href="#">23.4 Support levels</a>
Environmental effects	1	<a href="#">24.5 Support levels</a>

## D.4 Conformance criteria

Conformance to this profile shall include conformance criteria defined by the specifications for those components and levels listed in [Table D.2](#).

In Tables D.3 and D.4, the first column defines the item for which conformance is being defined. In some cases, general limits are defined but are later overridden in specific cases by more restrictive limits. The second column defines the requirements for a X3D file conforming to the MPEG-4 interactive profile; if a X3D file contains any items that exceed these limits, it may not be possible for a X3D browser conforming to the MPEG-4 interactive profile to successfully parse that X3D file. The third column defines the minimum complexity for a X3D scene that a X3D browser conforming to the MPEG-4 interactive profile shall be able to present to the user. Fields flagged as "not supported" may be supported by browsers which conform to the MPEG-4 interactive profile. The word "ignore" in the minimum browser support column refers only to the display of the item; in particular, *set\_* events to ignored inputOutput fields shall still generate corresponding *\_changed* events.

## D.5 Node set

[Table D.3](#) lists the nodes that shall be supported in the MPEG-4 interactive profile and specifies any fields in these nodes for which this profile requires less than full support.

**Table D.3 — Nodes for conforming to the MPEG-4 Interactive profile**

Item	File Limit	Minimum Browser Support
Anchor	No restrictions.	<i>addChildren</i> optionally supported. <i>removeChildren</i> optionally supported. Ignore <i>parameter</i> . Ignore <i>description</i> .
Appearance	No restrictions.	<i>textureTransform</i> optionally supported. <i>lineProperties</i> not supported. <i>fillProperties</i> not supported.
Background	No restrictions.	<i>groundAngle</i> and <i>groundColor</i> optionally supported. <i>backURL</i> , <i>frontURL</i> , <i>leftURL</i> , <i>rightURL</i> , <i>topURL</i> optionally supported. <i>skyAngle</i> optionally supported. One <i>skyColor</i> .
Box	No restrictions.	Full support.
Color	15,000 colours.	15,000 colours.
ColorInterpolator	Restrictions as for all interpolators.	Full support as for all interpolators.
ColorRGBA	15,000 colours.	15,000 colours. Alpha component optionally supported.
Cone	No restrictions.	Full support.

Coordinate	65,535 points	65,535 points.
CoordinateInterpolator	15,000 coordinates per <i>keyValueD</i> . Restrictions as for all interpolators.	15,000 coordinates per <i>keyValue</i> . Support as for all interpolators.
Cylinder	No restrictions.	Full support.
CylinderSensor	No restrictions.	Full support.
DirectionalLight	No restrictions.	Not scoped by parent Group or Transform.
ElevationGrid	No restrictions.	<i>ccw</i> optionally supported.
Group	Restrictions as for all groups.	Full support except as for all groups.
ImageTexture	JPEG ( <a href="#">2.[JPEG]</a> ) and PNG ( <a href="#">2.[115948]</a> ) format.	JPEG ( <a href="#">2.[JPEG]</a> ) and PNG ( <a href="#">2.[115948]</a> ) format.
IndexedFaceSet	10 vertices per faceD. 5000 faces. Less than 65,535 indices.	<i>ccw</i> optionally supported. <i>set_colorIndex</i> optionally supported. <i>set_normalIndex</i> optionally supported. <i>normal</i> optionally supported. Only convex indexed face sets supported. Hence, <i>convex</i> optionally supported. For <i>creaseAngle</i> , only 0 and $\pi$ radians supported (or the equivalent if a different angle base unit has been specified). <i>normalIndex</i> optionally supported. 10 vertices per face. 5000 faces. 65,535 indices in any index field. Face list shall be well-defined as follows: <ol style="list-style-type: none"> <li>1. Each face is terminated with <math>-1</math>, including the last face in the array.</li> <li>2. Each face contains at least three non-coincident vertices.</li> <li>3. A given <i>coordIndex</i> is not repeated in a face.</li> <li>4. The vertices of a face shall define a planar polygon.</li> </ol>

		5. The vertices of a face shall not define a self-intersecting polygon.
IndexedLineSet	15,000 total vertices. 15,000 indices in any index field.	15,000 total vertices. 15,000 indices in any index field. <i>set_colorIndex</i> optionally supported. <i>set_coordIndex</i> optionally supported.
Inline	No restrictions.	All fields except <i>load</i> which is optionally supported.
LineSet	15,000 total vertices.	15,000 total vertices.
Material	No restrictions.	<i>ambientIntensity</i> optionally supported. <i>shininess</i> optionally supported. <i>specularColor</i> optionally supported. A Material with <i>emissiveColor</i> not equal to (0,0,0), <i>diffuseColor</i> equal to (0,0,0) is an unlit material. One-bit transparency; transparency values $\geq 0.5$ transparent.
MetadataBoolean	No restrictions.	Full support.
MetadataDouble	No restrictions.	Full support.
MetadataFloat	No restrictions.	Full support.
MetadataInteger	No restrictions.	Full support.
MetadataSet	No restrictions.	Full support.
MetadataString	No restrictions.	Full support.
NavigationInfo	No restrictions.	<i>avatarSize</i> optionally supported. <i>speed</i> optionally supported. <i>type</i> optionally supported. <i>visibilityLimit</i> optionally supported.
NormalInterpolator	Restrictions as for all interpolators.	Full support except as for all interpolators.
OrientationInterpolator	Restrictions as for all interpolators.	Full support except as for all interpolators.

PixelTexture	512 width. 512 height.	512 width. 512 height. Display fully transparent and fully opaque pixels.
PlaneSensor	No restrictions.	Full support.
PointLight	No restrictions.	<i>radius</i> optionally supported. Linear attenuation.
PointSet	5000 points.	5000 points.
PositionInterpolator	Restrictions as for all interpolators.	Full support except as for all interpolators.
ProximitySensor	No restrictions.	<i>position_changed</i> optionally supported. <i>orientation_changed</i> optionally supported.
ScalarInterpolator	Restrictions as for all interpolators.	Full support except as for all interpolators.
Shape	No restrictions.	Full support.
Sphere	No restrictions.	Full support.
SphereSensor	No restrictions.	Full support.
SpotLight	No restriction	<i>beamWidth</i> optionally supported. <i>radius</i> optionally supported. Linear attenuation.
Switch	No restrictions	Full support.
TextureCoordinate	65,535 coordinates.	65,535 coordinates.
TextureTransform	No restrictions.	Full support.
TimeSensor	No restrictions.	<i>pause</i> optionally supported. <i>isPaused</i> optionally supported. <i>resumeTime</i> optionally supported.
TouchSensor	No restrictions.	Full support.
Transform	Restrictions as for all groups.	Full support except as for all groups.



Viewpoint	No restrictions.	<i>fieldOfView</i> optionally supported. <i>description</i> optionally supported.
WorldInfo	No restrictions.	Full support.

## D.6 Other limitations

[Table D.4](#) specifies other aspects of X3D functionality that are supported by this profile. Note that general items refer only to those specific nodes listed in [Table D.3](#).

**Table D.4 — Other limitations**

Item	X3D File Limit	Minimum Browser Support
All groups	500 children.	500 children. Ignore <i>bboxCenter</i> and <i>bboxSizD</i> .
All interpolators	1000 key-value pairs.	1000 key-value pairs.
All lights	8 simultaneous lights.	8 simultaneous lights.
Names for DEF/field	50 utf8 octets.	50 utf8 octets.
All <i>url</i> fields	10 URLs.	10 URLs. URN's ignored. Support `http`, `file`, and `ftp` protocols. Support relative URLs where relevant.
SFBool	No restrictions.	Full support.
SFColor	No restrictions.	Full support.
SFColorRGBA	No restrictions.	Full support.
SFDouble	Mp restrictions.	Full support. Range $\pm 1e\pm 12$ . Precision $1e-7$ .
SFFloat	No restrictions.	Full support.
SFImage	256 width. 256 height.	256 width. 256 height.
SFInt32	No restrictions.	Full support.
SFNode	No restrictions.	Full support.
SFRotation	No restrictions.	Full support.
SFString	30,000 utf8 octets.	30,000 utf8 octets.

SFTime	No restrictions.	Full support.
SFVec2d	15,000 values.	15,000 values.
SFVec2f	15,000 values.	15,000 values.
SFVec3d	15,000 values.	15,000 values.
SFVec3f	15,000 values.	15,000 values.
MFCColor	15,000 values.	15,000 values.
MFCColorRGBA	15,000 values.	15,000 values.
MFDDouble	1000 values.	1000 values.
MFFloat	1,000 values.	1,000 values.
MFInt32	20,000 values.	20,000 values.
MFNode	500 values.	500 values.
MFRotation	1,000 values.	1,000 values.
MFString	30,000 utf8 octets per string, 10 strings.	30,000 utf8 octets per string, 10 strings.
MFTime	1,000 values.	1,000 values.
MFVec2d	15,000 values.	15,000 values.
MFVec2f	15,000 values.	15,000 values.
MFVec3d	15,000 values.	15,000 values.
MFVec3f	15,000 values.	15,000 values.





## Extensible 3D (X3D) Part 1: Architecture and base components

### 5 Field type reference



#### 5.1 General

This clause describes the syntax and general semantics of *fields*, the elemental data types used by X3D to define the properties of nodes. Nodes are composed of fields whose types are defined in this clause. For more information on nodes, see [4.4.2 Object model](#).

[Table 5.1](#) provides links to the major topics in this clause.

**Table 5.1 — Topics**

- [5.1 General](#)
- [5.2 Abstract field types](#)
  - [5.2.1 Overview](#)
  - [5.2.2 X3DArrayField](#)
  - [5.2.3 X3DField](#)
- [5.3 Field types](#)
  - [5.3.1 SFBool and MFBool](#)
  - [5.3.2 SFColor and MFColor](#)
  - [5.3.3 SFColorRGBA and MFColorRGBA](#)
  - [5.3.4 SFDouble and MFDouble](#)
  - [5.3.5 SFFloat and MFFloat](#)
  - [5.3.6 SFImage and MFImage](#)
  - [5.3.7 SFInt32 and MFInt32](#)
  - [5.3.8 SFMatrix3d and MFMatrix3d](#)
  - [5.3.9 SFMatrix3f and MFMatrix3f](#)
  - [5.3.10 SFMatrix4d and MFMatrix4d](#)
  - [5.3.11 SFMatrix4f and MFMatrix4f](#)
  - [5.3.12 SFNode and MFNode](#)
  - [5.3.13 SFRotation and MFRotation](#)
  - [5.3.14 SFString and MFString](#)
  - [5.3.15 SFTime and MFTime](#)
  - [5.3.16 SFVec2d and MFVec2d](#)

- [5.3.17 SFVec2f and MFVec2f](#)
- [5.3.18 SFVec3d and MFVec3d](#)
- [5.3.19 SFVec3f and MFVec3f](#)
- [5.3.20 SFVec4d and MFVec4d](#)
- [5.3.21 SFVec4f and MFVec4f](#)



## 5.2 Abstract field types

### 5.2.1 Overview

There are two general classes of field types: field types that contain a single value (where a value may be a single number, a vector, or even an image), and field types that contain an ordered list of multiple values. Single-valued field types have names that begin with **SF**. Multiple-valued field types have names that begin with **MF**. Multiple-valued fields are written as an ordered list of values. If the field has zero values, the value is empty but still represented.

### 5.2.2 *X3DArrayField*

*X3DArrayField* is the abstract field type from which all field types that can contain multiple values are derived. All fields derived from *X3DArrayField* have names beginning with **MF**. MFxxxx fields may zero or more values, each of which shall be of the type indicated by the corresponding SFxxxx field type. It is illegal for any MFxxxx field to mix values of different SFxxxx field types.

EXAMPLE MFString is a field type that can contain zero or more character strings.

### 5.2.3 *X3DField*

*X3DField* is the abstract field type from which all single values field types are derived. All fields derived from *X3DField* have names beginning with **SF**. SFxxxx fields may only contain a single value of the type indicated by the name of the field type.

EXAMPLE SFBool is a field type that can contain a single Boolean value.



## 5.3 Field types

### 5.3.1 SFBool and MFBool

The SFBool field specifies a single Boolean value. The MFBool field specifies multiple Boolean values. Each Boolean value represents either `TRUE` or `FALSE`. How these values are represented is encoding dependent.

The default value of an uninitialized SFBool field is `FALSE`. The default value of an uninitialized MFBool field is the empty list.

### 5.3.2 SFCOLOR and MFCOLOR

The SFCOLOR field specifies one RGB (red-green-blue) colour triple. MFCOLOR specifies zero or more RGB triples. Each colour is written to the X3D file as an RGB triple of floating point numbers in the range 0.0 to 1.0.

The default value of an uninitialized SFCOLOR field is `(0 0 0)`. The default value of an uninitialized MFCOLOR field is the empty list.

### 5.3.3 SFCOLORRGBA and MFCOLORRGBA

The SFCOLORRGBA field specifies one RGBA (red-green-blue-alpha) colour quadruple that includes alpha (opacity) information. MFCOLORRGBA specifies zero or more RGBA quadruples. Each colour is written to the X3D file as an RGBA quadruple of floating point numbers in the range 0.0 to 1.0. Alpha values range from 0.0 (fully transparent) to 1.0 (fully opaque).

The default value of an uninitialized SFCOLORRGBA field is `(0 0 0 0)`. The default value of an uninitialized MFCOLORRGBA field is the empty list.

### 5.3.4 SFDOUBLE and MFDOUBLE

The SFDOUBLE field specifies one double-precision floating point number. MFDOUBLE specifies zero or more double-precision floating point numbers. SFDOUBLE and MFDOUBLE are represented in the X3D file as specified in the respective encoding.

Implementation of these fields is targeted at the double precision floating point capabilities of processors. However, it is allowable to implement this field using fixed point numbering provided at least 14 decimal digits of precision are maintained and that exponents have range of at least `[-12, 12]` for both positive and negative numbers.

The default value of an uninitialized SFDOUBLE field is `0.0`. The default value of an MFDOUBLE field is the empty list.

### 5.3.5 SFFLOAT and MFFLOAT

The SFFLOAT field specifies one single-precision floating point number. MFFLOAT specifies zero or more single-precision floating point numbers. SFFLOATS and MFFLOATS are represented in the X3D file as specified in the respective encoding.

Implementation of these fields is targeted at the single precision floating point capabilities of processors. However, it is allowable to implement this field using fixed point numbering provided at least six decimal digits of precision are maintained and that exponents have range of at least `[-12, 12]` for both positive and negative numbers.

The default value of an uninitialized SFFLOAT field is `0.0`. The default value of an MFFLOAT field is the empty list.

### 5.3.6 SFImage and MFImage

The SFImage field specifies a single uncompressed 2-dimensional pixel image. SFImage fields contain three integers representing the width, height and number of components in the image, followed by width×height hexadecimal or integer values representing the pixels in the image. MFImage fields contain zero or more SFImage fields. Each image in an MFImage field may contain different values for the width, height, and number of components in the image and hence may have a different number of hexadecimal or integer values.

Pixel values are limited to 256 levels of intensity (*i.e.*, 0-255 decimal or 0x00-0xFF hexadecimal). A one-component image specifies one-byte hexadecimal or integer values representing the intensity of the image. For example, 0xFF is full intensity in hexadecimal (255 in decimal), 0x00 is no intensity (0 in decimal). A two-component image specifies the intensity in the first (high) byte and the alpha opacity in the second (low) byte. Pixels in a three-component image specify the red component in the first (high) byte, followed by the green and blue components (*e.g.*, 0xFF0000 is red, 0x00FF00 is green, 0x0000FF is blue). Four-component images specify the alpha opacity byte after red/green/blue (*e.g.*, 0x0000FF80 is semi-transparent blue). A value of 0x00 is completely transparent, 0xFF is completely opaque. Note that alpha equals (1.0 -transparency), if alpha and transparency each range from 0.0 to 1.0.

Each pixel is read as a single unsigned number. For example, a 3-component pixel with value 0x0000FF may also be written as 0xFF (hexadecimal) or 255 (decimal). Pixels are specified from left to right, bottom to top. The first hexadecimal value is the lower left pixel and the last value is the upper right pixel.

The default value of an SFImage outputOnly field is (0 0 0). The default value of an MFImage field is the empty list.

### 5.3.7 SFInt32 and MFInt32

The SFInt32 field specifies one 32-bit integer. The MFInt32 field specifies zero or more 32-bit integers. SFInt32 and MFInt32 fields are signed integers.

The default value of an uninitialized SFInt32 field is 0. The default value of an MFInt32 field is the empty list.

### 5.3.8 SFMatrix3d and MFMatrix3d

The SFMatrix3d field specifies a 3×3 matrix of double-precision floating point numbers. MFMatrix3d specifies zero or more 3×3 matrices of double-precision floating point numbers. Each floating point number is represented in the X3D file as specified in the respective encoding.

SFMatrix3d matrices are organized in row-major fashion. The first row of the matrix stores information for the x dimension, and the second for the y dimension. Since these data types are commonly used for transformation matrices, translation values are stored in the third row.

The default value of an uninitialized SFMatrix3d field is the identity matrix [1 0 0 0 1 0

0 0 1]. The default value of an uninitialized MFMatrix3d field is the empty list.

### 5.3.9 SFMatrix3f and MFMatrix3f

The SFMatrix3f field specifies a  $3 \times 3$  matrix of single-precision floating point numbers. MFMatrix3f specifies zero or more  $3 \times 3$  matrices of single-precision floating point numbers. Each floating point number is represented in the X3D file as specified in the respective encoding.

SFMatrix3f matrices are organized in row-major fashion. The first row of the matrix stores information for the  $x$  dimension, and the second for the  $y$  dimension. Since these data types are commonly used for transformation matrices, translation values are stored in the third row.

The default value of an uninitialized SFMatrix3f field is the identity matrix [1 0 0 0 1 0 0 0 1]. The default value of an uninitialized MFMatrix3f field is the empty list.

### 5.3.10 SFMatrix4d and MFMatrix4d

The SFMatrix4d field specifies a  $4 \times 4$  matrix of double-precision floating point numbers. MFMatrix4d specifies zero or more  $4 \times 4$  matrices of double-precision floating point numbers. Each floating point number is represented in the X3D file as specified in the respective encoding.

SFMatrix4d matrices are organized in row-major fashion. The first row of the matrix stores information for the  $x$  dimension, the second for the  $y$  dimension, and the third for the  $z$  dimension. Since these data types are commonly used for transformation matrices, translation values are stored in the fourth row.

The default value of an uninitialized SFMatrix4d field is the identity matrix [1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1]. The default value of an uninitialized MFMatrix4d field is the empty list.

### 5.3.11 SFMatrix4f and MFMatrix4f

The SFMatrix4f field specifies a  $4 \times 4$  matrix of single-precision floating point numbers. MFMatrix4f specifies zero or more  $4 \times 4$  matrices of single-precision floating point numbers. Each floating point number is represented in the X3D file as specified in the respective encoding.

SFMatrix4f matrices are organized in row-major fashion. The first row of the matrix stores information for the  $x$  dimension, the second for the  $y$  dimension, and the third for the  $z$  dimension. Since these data types are commonly used for transformation matrices, translation values are stored in the fourth row.

The default value of an uninitialized SFMatrix4f field is the identity matrix [1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1]. The default value of an uninitialized MFMatrix4f field is the empty list.

### 5.3.12 SFNode and MFNode



The SFNode field specifies an X3D node. The MFNode field specifies zero or more nodes.

The default value of an uninitialized SFNode field is `NULL`. The default value of an MFNode field is the empty list.

### 5.3.13 SFRotation and MFRotation

The SFRotation field specifies one arbitrary rotation. The MFRotation field specifies zero or more arbitrary rotations. An SFRotation is written to the X3D file as four floating point values. The allowable form for a floating point number is defined in the specific encoding. The first three values specify a normalized rotation axis vector about which the rotation takes place. The fourth value specifies the amount of right-handed rotation about that axis in angle base units.

The 3x3 matrix representation of a rotation (x y z a) is

$$\begin{bmatrix} tx^2+c & txy+sz & txz-sy \\ txy-sz & ty^2+c & tyz+sx \\ txz+sy & tyz-sx & tz^2+c \end{bmatrix}$$

where  $c = \cos(a)$ ,  $s = \sin(a)$ , and  $t = 1-c$ .

The default value of an uninitialized SFRotation field is (0 0 1 0). The default value of an MFRotation field is the empty list.

### 5.3.14 SFString and MFString

The SFString and MFString fields contain strings encoded with the UTF-8 universal character set (see [ISO/IEC 10646](#)). SFString specifies a single string. The MFString specifies zero or more strings. Strings are specified as a sequence of UTF-8 octets.

Any characters (including linefeeds and '#') may appear within the string.

The default value of an uninitialized SFString outputOnly field is the empty string. The default value of an MFString field is the empty list.

Characters in [ISO/IEC 10646](#) are encoded in multiple octets. Code space is divided into four units, as follows:

```
+-----+-----+-----+-----+
| Group-octet | Plane-octet | Row-octet | Cell-octet |
+-----+-----+-----+-----+
```

[ISO/IEC 10646](#) allows two basic forms for characters:

- UCS-2 (Universal Coded Character Set-2). This form is also known as the Basic Multilingual Plane (BMP). Characters are encoded in the lower two octets (row and cell).
- UCS-4 (Universal Coded Character Set-4). Characters are encoded in the full four octets.

In addition, two transformation formats (UCS Transformation Format or UTF) are

accepted: UTF-8 and UTF-16. Each represents the nature of the transformation: 8-bit or 16-bit. UTF-8 and UTF-16 are referenced in [ISO/IEC 10646](#).

UTF-8 maintains transparency for all ASCII code values (0..127). It allows ASCII text (0x0..0x7F) to appear without any changes and encodes all characters from 0x80..0x7FFFFFFF into a series of six or fewer bytes.

If the most significant bit of the first character is 0, the remaining seven bits are interpreted as an ASCII character. Otherwise, the number of leading 1 bits indicates the number of bytes following. There is always a zero bit between the count bits and any data.

The first byte is one of the following. The X indicates bits available to encode the character:

0XXXXXXX	only one byte	0..0x7F (ASCII)	
110XXXXX	two bytes	Maximum character value is 0x7FF	
1110XXXX	three bytes	Maximum character value is 0xFFFF	
11110XXX	four bytes	Maximum character value is 0x1FFFFFF	
111110XX	five bytes	Maximum character value is 0x3FFFFFFF	
1111110X	six bytes	Maximum character value is 0x7FFFFFFF	

All following bytes have the format 10XXXXXX.

As a two byte example, the symbol for a registered trade mark ®, encoded as 0x00AE in UCS-2 of ISO 10646, has the following two byte encoding in UTF-8: 0xC2, 0xAE.

### 5.3.15 SFTIME and MFTime

The SFTIME field specifies a single time value. The MFTime field specifies zero or more time values. Time values are specified as a double-precision floating point number. The allowable form for a double precision floating point number is defined in the specific encoding. Time values are specified as the number of seconds from a specific time origin. Typically, SFTIME fields represent the number of seconds since Jan 1, 1970, 00:00:00 GMT.

The default value of an uninitialized SFTIME field is -1. The default value of an MFTime field is the empty list.

### 5.3.16 SFVec2d and MFVec2d

The SFVec2d field specifies a two-dimensional (2D) vector. An MFVec2d field specifies zero or more 2D vectors. SFVec2d's and MFVec2d's are represented as a pair of double-precision floating point values (see [5.3.4 SFDouble and MFDouble](#)). The allowable form for a double-precision floating point number is defined in the specific encoding.

The default value of an uninitialized SFVec2d field is (0 0). The default value of an MFVec2d field is the empty list.

### 5.3.17 SFVec2f and MFVec2f

The SFVec2f field specifies a two-dimensional (2D) vector. An MFVec2f field specifies zero or more 2D vectors. SFVec2f's and MFVec2f's are represented as a pair of single-precision floating point values (see [5.3.5 SFFloat and MFFloat](#)). The allowable form for a

single-precision floating point number is defined in the specific encoding.

The default value of an uninitialized SFVec2f field is (0 0). The default value of an MFVec2f field is the empty list.

### 5.3.18 SFVec3d and MFVec3d

The SFVec3d field or event specifies a three-dimensional (3D) vector. An MFVec3d field or event specifies zero or more 3D vectors. SFVec3d's and MFVec3d's are represented as a 3-tuple of double-precision floating point values (see [5.3.4 SFDouble and MFDouble](#)). The allowable form for a double-precision floating point number is defined in the specific encoding.

The default value of an uninitialized SFVec3d field is (0 0 0). The default value of an MFVec3d field is the empty list.

### 5.3.19 SFVec3f and MFVec3f

The SFVec3f field or event specifies a three-dimensional (3D) vector. An MFVec3f field or event specifies zero or more 3D vectors. SFVec3f's and MFVec3f's are represented as a 3-tuple of single-precision floating point values (see [5.3.5 SFFloat and MFFloat](#)). The allowable form for a single-precision floating point number is defined in the specific encoding.

The default value of an uninitialized SFVec3f field is (0 0 0). The default value of an MFVec3f field is the empty list.

### 5.3.20 SFVec4d and MFVec4d

The SFVec4d field or event specifies a three-dimensional (3D) homogeneous vector. An MFVec4d field or event specifies zero or more 3D homogeneous vectors. SFVec4d's and MFVec4d's are represented as a 4-tuple of double-precision floating point values (see [5.3.4 SFDouble and MFDouble](#)). The allowable form for a double-precision floating point number is defined in the specific encoding.

The default value of an uninitialized SFVec4d field is (0 0 0 1). The default value of an MFVec4d field is the empty list.

### 5.3.21 SFVec4f and MFVec4f

The SFVec4f field or event specifies a three-dimensional (3D) homogeneous vector. An MFVec4f field or event specifies zero or more 3D homogeneous vectors. SFVec4f's and MFVec4f's are represented as a 4-tuple of single-precision floating point values (see [5.3.5 SFFloat and MFFloat](#)). The allowable form for a single-precision floating point number is defined in the specific encoding.

The default value of an uninitialized SFVec4f field is (0 0 0 1). The default value of an MFVec4f field is the empty list.







## Extensible 3D (X3D) Part 1: Architecture and base components

# 26 Humanoid Animation (H-Anim HAnim) component

## 26.1 Introduction

### 26.1.1 Name

The name of this component is "H-Anim HAnim". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

### 26.1.2 Overview

This clause describes the Humanoid Animation (H-Anim HAnim) component of this part of ISO/IEC 19775. [Table 26.1](#) provides links to the major topics in this clause. The H-Anim HAnim component of X3D defines the node bindings and other specifics for implementing ISO/IEC 19774 (see [2.\[19774\]](#)) within X3D.

**Table 26.1 — Topics**

- [26.1 Introduction](#)
  - [26.1.1 Name](#)
  - [26.1.2 Overview](#)
- [26.2 Concepts](#)
- [26.3 Node reference](#)
  - [26.3.1 HAnimDisplacer](#)
  - [26.3.2 HAnimHumanoid](#)
  - [26.3.3 HAnimJoint](#)
  - [26.3.4 HAnimMotion](#)
  - [26.3.5 HAnimSegment](#)
  - [26.3.6 HAnimSite](#)
- [26.4 Support levels](#)
- [Table 26.1 — Topics](#)
- [Table 26.2 — H-anim component support levels](#)

## 26.2 Concepts

This component maps the functionality defined in [ISO/IEC 19774](#) to a set of X3D nodes. The semantics for these nodes are as specified therein.

## 26.3 Node reference

### 26.3.1 HAnimDisplacer

```
HAnimDisplacer : X3DGeometricPropertyNode {
  MFInt32 [in,out] coordIndex [] [0,∞) or -1
  SFString [in,out] description ""
  MFVec3f [in,out] displacements []
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFString [in,out] name ""
  SFFloat [in,out] weight 0.0 (-∞,∞)
}
```

Applications may need to alter the shape of individual segments. At the most basic level, this is done by writing to the *point* field of the node derived from [X3DCoordinateNode](#) that is found in the *coord* field of the [HAnimSegment](#) node.

In some cases, the application may need to be able to identify specific groups of vertices within an HAnimSegment.

EXAMPLE The application may need to know which vertices within the skull HAnimSegment comprise the left eyebrow.

It may also require "hints" as to the direction in which each vertex should move. That information is stored in a node called an HAnimDisplacer. The HAnimDisplacers for a particular HAnimSegment are stored in the *displacers* field of that HAnimSegment.

The description of each field shall be as described in [ISO/IEC 19774](#).

### 26.3.2 HAnimHumanoid

```
HAnimHumanoid : X3DChildNode, X3DBoundedObject {
  SFVec3f [in,out] center 0 0 0 (-∞,∞)
  SFString [in,out] description ""
  SFBool [in,out] bboxDisplav FALSE
  SFBool [in,out] visible TRUE
  MFString [in,out] info []
  MFVec3f [in,out] jointBindingPositions [] (-∞,∞)
  MFRotation [in,out] jointBindingRotations [] (-∞,∞)[[-1,1]
  MFVec3f [in,out] jointBindingScales [] (0,∞)
  MFNode [in,out] ioints [] [HAnimJoint]
  SFInt32 [in,out] loa -1 [-1,4]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFNode [in,out] motions [] [HAnimMotion]
  MFBool [in,out] motionsEnabled []
  SFString [in,out] name ""
  SFRotation [in,out] rotation 0 0 1 0 (-∞,∞)[[-1,1]
  SFVec3f [in,out] scale 1 1 1 (0,∞)
  SFRotation [in,out] scaleOrientation 0 0 1 0 (-∞,∞)[[-1,1]
  MFNode [in,out] segments [] [HAnimSegment]
  MFNode [in,out] sites [] [HAnimSite]
  SFString [in,out] skeletalConfiguration "BASIC"
  MFNode [in,out] skeleton [] [HAnimJoint, HAnimSite]
  MFNode [in,out] skin [] [IndexedFaceSet, X3DGroupingNodeGroup, Transform, Shape]
  SFNode [in,out] skinBindingCoords NULL [X3DCoordinateNode]
  SFNode [in,out] skinBindingNormals NULL [X3DNormalNode]
  SFNode [in,out] skinCoord NULL [X3DCoordinateNode]
  SFNode [in,out] skinNormal NULL [X3DNormalNode]
  SFVec3f [in,out] translation 0 0 0 (-∞,∞)
  SFString [in,out] version ""
```

```

MFNode [in,out] viewpoints [] [HAnimSite]
SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}

```

The HAnimHumanoid node is used to store human-readable data such as author and copyright information, as well as to store references to the [HAnimJoint](#), [HAnimMotion](#), [HAnimSegment](#), and [HAnimSite](#) nodes in addition to serving as a container for the entire humanoid. Thus, it serves as an essential node for moving the humanoid through its environment.

~~The description of each field shall be as described in~~ Each field is described in [ISO/IEC 19774](#).

### 26.3.3 HAnimJoint

```

HAnimJoint : X3DGroupingNode {
MFNode [in] addChildren [HAnimJoint,HAnimSegment,HAnimSite]
MFNode [in] removeChildren [HAnimJoint,HAnimSegment,HAnimSite]
SFVec3f [in,out] center 0 0 0 (-∞,∞)
MFNode [in,out] children [] [HAnimJoint,HAnimSegment,HAnimSite]
SFString [in out] description ""
MFNode [in,out] displacers [] [HAnimDisplacer]
SFBool [in out] bboxDisblav FALSE
SFBool [in out] visible TRUE
SFRotation [in,out] limitOrientation 0 0 1 0 (-∞,∞)[-1,1]
MFVec3f [in,out] llimit [] (-∞,∞)
SFNode [in,out] metadata NULL [X3DMetadataObject]
SFString [in,out] name ""
SFRotation [in,out] rotation 0 0 1 0 (-∞,∞)[-1,1]
SFVec3f [in,out] scale 1 1 1 (0,∞)
SFRotation [in,out] scaleOrientation 0 0 1 0 (-∞,∞)[-1,1]
MFInt32 [in,out] skinCoordIndex []
MFFloat [in,out] skinCoordWeight []
MFFloat [in,out] stiffness [0 0 0] [0,1]
SFVec3f [in,out] translation 0 0 0 (-∞,∞)
MFVec3f [in,out] ulimit [] (-∞,∞)
SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}

```

Each joint in the body is represented by an HAnimJoint node, which is used to define the relationship of each body segment to its immediate parent.

An HAnimJoint may only be a child of another HAnimJoint node or a child within the *skeleton* field in the case of the HAnimJoint used as a humanoid root (*i.e.*, an HAnimJoint may not be a child of an [HAnimSegment](#)).

The HAnimJoint node is also used to store other joint-specific information. In particular, a joint *name* is provided so that applications can identify each HAnimJoint node at run-time. The HAnimJoint node may contain hints for inverse-kinematics systems that wish to control the ~~H-Anim~~ [HAnim](#) figure. These hints include the upper and lower joint limits, the orientation of the joint limits, and a stiffness/resistance value.

NOTE These limits are not enforced by any mechanism within the scene graph of the humanoid, and are provided for information purposes only. Use of this information and enforcement of the joint limits is up to the application.

Humanoid authors and tools are free to implement the HAnimJoint node however they choose. In particular, they may choose to use a single polygonal mesh to represent a humanoid, rather than having a separate [IndexedFaceSet](#) for each body segment. In such a case, an HAnimJoint would be responsible for moving the vertices that correspond to a particular body segment and all the segments descended from it.

~~The description of each field shall be as described in~~ Each field is described in [ISO/IEC 19774](#).



## 26.3.4 HAnimMotion

```

HAnimMotion : X3DChildNode {
  SFString [in,out] channels      ""
  MFBool   [in,out] channelsEnabled []
  SFTime   [out]   cycleTime
  SFString [in,out] description  ""
  SFTime   [out]   elapsedTime   (0,∞)
  SFBool   [in,out] enabled      TRUE
  SFInt32  [out]   frameCount    [0,∞)
  SFTime   [in,out] frameDuration 0.1 (0,∞)
  SFInt32  [in,out] frameIncrement 1 (-∞,∞)
  SFInt32  [in,out] frameIndex   0 (0,∞)
  SFString [in,out] joints       ""
  SFInt32  [in,out] loa          -1 [-1,4]
  SFBool   [in,out] loop         false
  SFNode   [in,out] metadata     NULL [X3DMetadataObject]
  SFBool   [in]   next
  SFBool   [in]   previous
  MFFloat  [in,out] values       [] (-∞,∞)
}

```

HAnimMotion is used for motion animation of Humanoid characters. Raw motion data, for example, motion capture data, details the number of frames, the frame display time, and the parameter values for the motion from each channel at each frame.

The description of each field shall be as described in Each field is described in [ISO/IEC 19774](#).

## 26.3.5 HAnimSegment

```

HAnimSegment : X3DGroupingNode {
  MFNode [in]   addChildren      [X3DChildNode]
  MFNode [in]   removeChildren  [X3DChildNode]
  SFVec3f [in,out] centerOfMass 0 0 0 (-∞,∞)
  MFNode [in,out] children      [] [X3DChildNode]
  SFNode [in,out] coord         NULL [X3DCoordinateNode]
  SFString [in,out] description ""
  MFNode [in,out] displacers    [] [HAnimDisplacer]
  SFBool [in,out] bboxDisplav  FALSE
  SFBool [in,out] visible       TRUE
  SFFloat [in,out] mass         0 [0,∞)
  SFNode [in,out] metadata     NULL [X3DMetadataObject]
  MFFloat [in,out] momentsOfInertia [0 0 0 0 0 0 0 0] [0,∞)
  SFString [in,out] name
  SFVec3f []   bboxCenter      0 0 0 (-∞,∞)
  SFVec3f []   bboxSize        -1 -1 -1 [0,∞) or -1 -1 -1
}

```

Each body segment is stored in an HAnimSegment node. The HAnimSegment node is a grouping node that will typically contain either a number of [Shape](#) nodes or perhaps [Transform](#) nodes that position the body part within its coordinate system as defined in [ISO/IEC 19774](#). The use of LOD nodes is recommended if the geometry of the HAnimSegment is complex.

The description of each field shall be as described in Each field is described in [ISO/IEC 19774](#).

## 26.3.6 HAnimSite

```

HAnimSite : X3DGroupingNode {
  MFNode [in]   addChildren      [X3DChildNode]
  MFNode [in]   removeChildren  [X3DChildNode]
  SFVec3f [in,out] center        0 0 0 (-∞,∞)
  MFNode [in,out] children      [] [X3DChildNode]
  SFString [in,out] description ""
  SFBool [in,out] bboxDisplav  FALSE
  SFBool [in,out] visible       TRUE
  SFNode [in,out] metadata     NULL [X3DMetadataObject]
  SFString [in,out] name
  SFRotation [in,out] rotation  0 0 1 0 (-∞,∞)[-1,1]
  SFVec3f [in,out] scale        1 1 1 (0,∞)
  SFRotation [in,out] scaleOrientation 0 0 1 0 (-∞,∞)[-1,1]
}

```

```

SFVec3f [in,out] translation 0 0 0 (-∞,∞)|[-1,1]
SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}
    
```

An HAnimSite node serves three purposes. The first is to define an "end effector" location that can be used by an inverse kinematics system. The second is to define an attachment point for accessories such as jewelry and clothing. The third is to define a location for a virtual camera in the reference frame of an HAnimSegment (such as a view "through the eyes" of the humanoid for use in multi-user worlds).

The description of each field shall be as described in Each field is described in [ISO/IEC 19774](#).

## 26.4 Support levels

The **H-Anim HAnim** component provides 3 levels of support as specified in [Table 26.2](#).

**Table 26.2 — Humanoid animation (**H-Anim HAnim**) component support levels**

Level	Prerequisites	Nodes/Features	Support
1	Core 1 Grouping 1 Geometry3D 2 Shape 1 Texturing 1 Navigation 2	HAnimHumanoid skeleton support	
		HAnimDisplacer	All fields fully supported.
		HAnimHumanoid	All fields fully supported except <i>skin</i> , <i>skinCoord</i> , <i>skinNormal</i> , <i>skinBindingCoords</i> , <i>skinBindingNormals</i> , <i>motions</i> and <i>motionsEnabled</i> fields.
		HAnimJoint	All fields fully supported except <i>skinCoordIndex</i> and <i>skinCoordWeight</i> fields.
		HAnimSegment	All fields fully supported.
		HAnimSite	All fields fully supported.
2	Core 1 Grouping 1 Geometry3D 2 Shape 1 Texturing 1 Navigation 2	HAnimHumanoid skin support	
		HAnimDisplacer	All fields fully supported.

		HAnimHumanoid	All fields fully supported except <i>motions</i> and <i>motionsEnabled</i> fields.
		HAnimJoint	All fields fully supported.
		HAnimSegment	All fields fully supported.
		HAnimSite	All fields fully supported.
<b>3</b>	Core 1 Grouping 1 Geometry3D 2 Shape 1 Texturing 1 Navigation 2	Motion animation support	
		HAnimDisplacer	All fields fully supported.
		HAnimHumanoid	All fields fully supported.
		HAnimJoint	All fields fully supported.
		HAnimMotion	All fields fully supported.
		HAnimSegment	All fields fully supported.
		HAnimSite	All fields fully supported.





# Extensible 3D (X3D) Part 1: Architecture and base components

## Annex E

(normative)

### Immersive profile

---



#### E.1 General

This annex defines the X3D components which comprise the immersive profile. This includes not only the nodes which shall be supported but also which fields in the supported nodes may be ignored.

This profile is targeted towards:

- implementing immersive virtual worlds with complete navigational and environmental sensor control and
- implementing the functionality within the X3D architectural framework analogous to that specified in ISO/IEC 14772-1 for the VRML base profile of that standard (see [2.14772-1](#)).

#### E.2 Topics

[Table E.1](#) provides links to the major topics in this annex.

**Table E.1 — Topics**

- |   |
|---|
| <ul style="list-style-type: none"><li>• <a href="#">E.1 General</a></li><li>• <a href="#">E.2 Topics</a></li><li>• <a href="#">E.3 Component support</a></li><li>• <a href="#">E.4 Conformance criteria</a></li><li>• <a href="#">E.5 Node set</a></li><li>• <a href="#">E.6 Other limitations</a></li><li>• <a href="#">Table E.1 — Topics</a></li></ul> |
|---|

- [Table E.2 — Components and levels](#)
- [Table E.3 — Nodes for conforming to the Immersive profile](#)
- [Table E.4 — Other limitations](#)

## E.3 Component support

[Table E.2](#) lists the components and their levels which shall be supported in the Immersive profile. Tables E.2 and E.3 describe limitations on required support for nodes and fields contained within these components.

**Table E.2 — Components and levels**

Component	Level	Reference
Core	2	<a href="#">7.5 Support levels</a>
Time	1	<a href="#">8.5 Support levels</a>
Networking	3	<a href="#">9.5 Support levels</a>
Grouping	2	<a href="#">10.5 Support levels</a>
Rendering	3	<a href="#">11.5 Support levels</a>
Shape	2	<a href="#">12.5 Support levels</a>
Geometry3D	4	<a href="#">13.4 Support levels</a>
Geometry2D	1	<a href="#">14.4 Support levels</a>
Text	1	<a href="#">15.5 Support levels</a>
Sound	1	<a href="#">16.5 Support levels</a>
Lighting	2	<a href="#">17.5 Support levels</a>
Texturing	3	<a href="#">18.5 Support levels</a>
Interpolation	2	<a href="#">19.5 Support levels</a>
Pointing device sensor	1	<a href="#">20.5 Support levels</a>
Key device sensor	2	<a href="#">21.5 Support levels</a>
Environmental sensor	2	<a href="#">22.5 Support levels</a>
Navigation	2	<a href="#">23.4 Support levels</a>
Environmental effects	2	<a href="#">24.5 Support levels</a>

Scripting	1	<a href="#">29.5 Support levels</a>
Event utilities	1	<a href="#">30.5 Support levels</a>

## E.4 Conformance criteria

Conformance to this profile shall include conformance criteria defined by the specifications for those components and levels listed in [Table E.2](#).

In [Table E.2](#) and [Table E.3](#), the first column defines the item for which conformance is being defined. In some cases, general limits are defined but are later overridden in specific cases by more restrictive limits. The second column defines the requirements for a X3D file conforming to the Immersive profile; if a X3D file contains any items that exceed these limits, it may not be possible for a X3D browser conforming to the Immersive profile to successfully parse that X3D file. The third column defines the minimum complexity for a X3D scene that a X3D browser conforming to the Immersive profile shall be able to present to the user. The word "ignore" in the minimum browser support column refers only to the display of the item; in particular, *set\_* events to ignored inputOutput fields shall still generate corresponding *\_changed* events.

## E.5 Node set

[Table E.3](#) lists the nodes which shall be supported in the Immersive profile and specifies any fields in these nodes for which this profile requires less than full support.

**Table E.3 — Nodes for conforming to the Immersive profile**

Item	X3D File Limit	Minimum Browser Support
Anchor	No restrictions.	Full support.
Appearance	No restrictions.	<i>fillProperties</i> not supported.
AudioClip	30 second uncompressed PCM WAV.	30 second uncompressed PCM WAV.
Background	No restrictions.	One <i>skyColor</i> , one <i>groundColor</i> , panorama images as per <i>ImageTexture</i> .
Billboard	Restrictions as for all groups.	Full support except as for all groups.
BooleanFilter	No restrictions.	Full support.

BooleanSequencer	No restrictions.	Full support.
BooleanToggle	No restrictions.	Full support.
BooleanTrigger	No restrictions.	Full support.
Box	No restrictions.	Full support.
Collision	Restrictions as for all groups.	Full support except as for all groups. Any navigation behaviour acceptable when collision occurs.
Color	15,000 colours.	15,000 colours.
ColorInterpolator	Restrictions as for all interpolators.	Full support except as for all interpolators.
ColorRGBA	15,000 colours.	15,000 colours. Alpha component optionally supported.
Cone	No restrictions.	Full support.
Coordinate	15,000 points.	15,000 points.
CoordinateInterpolator	15,000 coordinates per <i>keyValue</i> . Restrictions as for all interpolators.	15,000 coordinates per <i>keyValue</i> . Support as for all interpolators.
Cylinder	No restrictions.	Full support.
CylinderSensor	No restrictions.	Full support.
DirectionalLight	No restrictions.	Not scoped by parent Group or Transform.
ElevationGrid	16,000 heights.	16,000 heights.
Extrusion	(# <i>crossSection</i> points) × (# <i>spine</i> points) ≤	(# <i>crossSection</i> points) × (# <i>spine</i> points) ≤ 2,500.



	2,500.	
Fog	No restrictions.	Full support.
FontStyle	No restrictions.	If the values of the text aspects character set, <i>family</i> , <i>style</i> cannot be simultaneously supported, the order of precedence shall be: 1) character set 2) <i>family</i> 3) <i>style</i> . Browser shall display all characters in Table 2 (Basic Latin) and Table 3 (Latin-1 Supplement) of ISO/IEC 10646-1 (see <a href="#">2.[110646-1]</a> ).
Group	Restrictions as for all groups.	Full support except as for all groups.
ImageTexture	JPEG ( <a href="#">2.[JPEG]</a> ) and PNG ( <a href="#">2.[115948]</a> ) format. Restrictions as for PixelTexture.	JPEG ( <a href="#">2.[JPEG]</a> ) and PNG ( <a href="#">2.[115948]</a> ) format. Support as for PixelTexture.
IndexedFaceSet	10 vertices per face. 5000 faces. Less than 15,000 indices.	10 vertices per face. 5000 faces. 15,000 indices in any index field.
IndexedLineSet	15,000 total vertices. 15,000 indices in any index field.	15,000 total vertices. 15,000 indices in any index field.
IndexedTriangleFan	15,000 total vertices. 15,000 indices in any index field.	15,000 total vertices. 15,000 indices in any index field.
IndexedTriangleSet	15,000 total vertices. 15,000 indices in any index field.	15,000 total vertices. 15,000 indices in any index field.
IndexedTriangleStripSet	15,000 total vertices. 15,000 indices in any index	15,000 total vertices. 15,000 indices in any index field.

	field.	
Inline	No restrictions.	Full support.
IntegerSequencer	No restrictions.	Full support.
IntegerTrigger	No restrictions.	Full support.
KeySensor	No restrictions.	Full support.
LineProperties	No restrictions.	Full support.
LineSet	15,000 total vertices.	15,000 total vertices.
LoadSensor	No restrictions.	Full support.
LOD	Restrictions as for all groups.	At least first 4 <i>level/range</i> combinations interpreted, and support as for all groups.
Material	No restrictions.	Full support.
MetadataBoolean	No restrictions.	Full support.
MetadataDouble	No restrictions.	Full support.
MetadataFloat	No restrictions.	Full support.
MetadataInteger	No restrictions.	Full support.
MetadataSet	No restrictions.	Full support.
MetadataString	No restrictions.	Full support.
MovieTexture	MPEG1-Systems and MPEG1-Video formats (see <a href="#">2.[114496-1]</a> ).	MPEG1-Systems and MPEG1-Video formats (see <a href="#">2.[114496-1]</a> ). Display one active movie texture.
		At least two textures displayed

MultiTexture	No restrictions.	per node with any number specified. Full support.
MultiTextureCoordinate	15,000 coordinates.	15,000 coordinates.
MultiTextureTransform	Restrictions as for all groups.	<i>addChildren</i> optionally supported. <i>removeChildren</i> optionally supported. Otherwise, full support except as for all groups.
NavigationInfo	No restrictions.	Full support.
Normal	15,000 normals	15,000 normals
NormalInterpolator	15,000 normals per <i>keyValue</i> . Restrictions as for all interpolators.	15,000 normals per <i>keyValue</i> . Support as for all interpolators.
OrientationInterpolator	Restrictions as for all interpolators.	Full support except as for all interpolators.
PixelTexture	512 width. 512 height.	512 width. 512 height. Display fully transparent and fully opaque pixels.
PlaneSensor	No restrictions.	Full support.
PointLight	No restrictions.	<i>radius</i> optionally supported. Linear attenuation.
PointSet	5000 points.	5000 points.
Polyline2D	5000 vertices.	5000 vertices.
Polypoint2D	5000 points.	5000 points.
PositionInterpolator	Restrictions as for all interpolators.	Full support except as for all interpolators.
ProximitySensor	No restrictions.	Full support.

Rectangle2D	No restrictions.	Full support.
ScalarInterpolator	Restrictions as for all interpolators.	Full support except as for all interpolators.
Script	25 fields of each access type.	25 fields of each access type. No scripting language support required.
Shape	No restrictions.	Full support.
Sound	No restrictions.	2 active sounds.
Sphere	No restrictions.	Full support.
SphereSensor	No restrictions.	Full support.
SpotLight	No restriction	<i>beamWidth</i> optionally supported. <i>radius</i> optionally supported. Linear attenuation.
StringSensor	100 characters per string. 100 strings.	Full support. 100 characters per string. 100 strings.
Switch	Restrictions as for all groups.	Full support except as for all groups.
Text	100 characters per string. 100 strings.	100 characters per string. 100 strings.
TextureCoordinate	15,000 coordinates.	15,000 coordinates.
TextureCoordinateGenerator	No restrictions.	Full support.
TextureTransform	No restrictions.	Full support.
TimeSensor	No restrictions.	<i>pause</i> optionally supported. <i>isPaused</i> optionally supported. <i>resumeTime</i> optionally supported.
TimeTrigger	No restrictions.	Full support.

TouchSensor	No restrictions.	Full support.
TriangleFanSet	15,000 coordinates.	Full support.
TriangleSet	15,000 coordinates.	Full support.
TriangleSet2D	15,000 coordinates.	Full support.
TriangleStripSet	15,000 coordinates.	Full support.
Transform	Restrictions as for all groups.	Full support except as for all groups.
Viewpoint	No restrictions.	Full support.
VisibilitySensor	No restrictions.	Always visible.
WorldInfo	No restrictions.	Full support.

## E.6 Other limitations

[Table E.4](#) specifies limitations unrelated to nodes which are imposed by the Immersive profile.

**Table E.4 — Other limitations**

Item	X3D File Limit	Minimum Browser Support
All groups	500 children.	500 children. Ignore <i>bboxCenter</i> and <i>bboxSize</i> .
All interpolators	1000 key-value pairs.	1000 key-value pairs.
All lights	8 simultaneous lights.	8 simultaneous lights.
Names for DEF/PROTO/field	50 utf8 octets.	50 utf8 octets.
All <i>url</i> fields	10 URLs.	10 URLs. URN's ignored. Support 'http', 'file', and 'ftp' protocols. Support relative URLs where relevant.

PROTO/ EXTERNPROTO	30 fields of each access type.	30 fields of each access type.
EXTERNPROTO	n/a	URL references X3D files conforming to the current X3D profile/component configuration
PROTO definition nesting depth	5 levels.	5 levels.
SFBool	No restrictions.	Full support.
SFColor	No restrictions.	Full support.
SFColorRGBA	No restrictions.	Full support.
SFDouble	No restrictions.	Full support. Range $\pm 1e\pm 12$ . Precision $1e-7$ .
SFFloat	No restrictions.	Full support.
SFImage	512 width. 512 height.	512 width. 512 height.
SFInt32	No restrictions.	Full support.
SFNode	No restrictions.	Full support.
SFRotation	No restrictions.	Full support.
SFString	30,000 utf8 octets.	30,000 utf8 octets.
SFTime	No restrictions.	Full support.
SFVec2d	15,000 values.	15,000 values.
SFVec2f	15,000 values.	15,000 values.
SFVec3d	15,000 values.	15,000 values.
SFVec3f	15,000 values.	15,000 values.
MFCColor	15,000 values.	15,000 values.
MFCColorRGBA	15,000 values.	15,000 values.
MFDouble	1000 values.	1000 values.
MFFloat	1,000 values.	1,000 values.
MFInt32	20,000 values.	20,000 values.

MFNode	500 values.	500 values.
MFRotation	1,000 values.	1,000 values.
MFString	30,000 utf8 octets per string, 10 strings.	30,000 utf8 octets per string, 10 strings.
MFTime	1,000 values.	1,000 values.
MFVec2d	15,000 values.	15,000 values.
MFVec2f	15,000 values.	15,000 values.
MFVec3d	15,000 values.	15,000 values.
MFVec3f	15,000 values.	15,000 values.







## Extensible 3D (X3D) Part 1: Architecture and base components

### 6 Conformance

---



#### 6.1 General

##### 6.1.1 Topics

This clause addresses conformance of X3D files, X3D generators and X3D browsers. The topics covered in this clause are shown in [Table 6.1](#).

**Table 6.1 — Topics**

- [6.1 General](#)
  - [6.1.1 Topics](#)
  - [6.1.2 Objectives](#)
  - [6.1.3 Scope](#)
- [6.2 Conformance](#)
  - [6.2.1 Conformance of X3D files](#)
  - [6.2.2 Conformance of X3D generators](#)
  - [6.2.3 Conformance of X3D browsers](#)
- [6.3 Minimum support requirements](#)
  - [6.3.1 Minimum support requirements for generators](#)
  - [6.3.2 Minimum support requirements for browsers](#)

##### 6.1.2 Objectives

The primary objectives of the specifications in this clause are:

- a. to promote interoperability by eliminating arbitrary subsets of, or extensions to, ISO/IEC 19775;
- b. to promote uniformity in the development of conformance tests;
- c. to promote consistent results across X3D browsers;
- d. to facilitate automated test generation.

## 6.1.3 Scope

Conformance is defined for X3D files and for X3D browsers. For X3D generators, conformance guidelines are presented for enhancing the likelihood of successful interoperability.

A concept of *base profile conformance* is defined to ensure interoperability of X3D generators and X3D browsers. Base profile conformance is based on a set of limits and minimal requirements. Base profile conformance is intended to provide a functional level of reasonable utility for X3D generators while limiting the complexity and resource requirements of X3D browsers. Base profile conformance may not be adequate for all uses of X3D.

This clause addresses the X3D data stream and implementation requirements. Implementation requirements include the latitude allowed for X3D generators and X3D browsers. This clause does not directly address the environmental, performance, or resource requirements of the generator or browser.

This clause does not define the application requirements or dictate application functional content within a X3D file.

The scope of this clause is limited to rules for the open interchange of X3D content.

## 6.2 Conformance

### 6.2.1 Conformance of X3D files

An X3D file is *syntactically correct* according to this part of ISO/IEC 19775 if the following conditions are met:

- a. The X3D file contains as its first element an X3D header statement (see [7.2.5.2 Header](#) statement) specifying the version to which this file conforms. Versions and associated content are specified in [Annex L Version content](#).
- b. All entities contained therein match the functional specification of entities of this part of ISO/IEC 19775 that correspond to the version specified in the header statement. The X3D file shall obey the relationships defined in the formal grammar and all other syntactic requirements.
- c. The sequence of entities in the X3D file obeys the relationships specified in this part of ISO/IEC 19775 for the version specified in the header statement producing the structure specified in the part of ISO/IEC 19775 for the version specified in the header statement.
- d. All field values in the X3D file obey the relationships specified in this part of ISO/IEC 19775 for the version specified in the header statement producing the structure specified in this part of ISO/IEC 19775 for the version specified in the header statement.
- e. No nodes appear in the X3D file other than those specified for the applicable profile as specified in this part of ISO/IEC 19775 unless specified in a COMPONENT statement, are required for the encoding technique, or are those defined by the PROTO or EXTERNPROTO entities should such be available in the profile.

- f. No nodes or fields appear in the X3D file other than those defined as part of the version specified in the header statement.
- g. The X3D file is encoded according to the rules of [ISO/IEC 19776](#).
- h. It does not contain behaviour described as undefined in this part of ISO/IEC 19775.

## 6.2.2 Conformance of X3D generators

A X3D generator is conforming to ISO/IEC 19775 if all X3D files that are generated are syntactically correct and meet the requirements state in [6.2.1 Conformance of X3D files](#).

A X3D generator conforms to a profile if it can be configured such that all X3D files generated conform to that profile.

## 6.2.3 Conformance of X3D browsers

An X3D browser is conforming if:

- a. It is able to read any X3D file that conforms to the profiles and components supported by that browser for the version(s) support by that browser.
- b. It implements the functionality specified for all abstract interfaces, insofar as they are made available in concrete nodes derived from those interfaces, within the latitude defined for the specified profile, components, and version and as allowed in this clause.
- c. It presents the graphical and audio characteristics of the X3D nodes in any X3D file that conforms to the applicable profile, components, and version, within the latitude defined for the specified profile, components, and version and as allowed in this clause.
- d. It correctly handles user interaction and generation of events as specified for the applicable profile, components, and version, within the latitude defined for the specified profile, components, and version and as allowed in this clause.
- e. It satisfies the minimum support requirements for browsers for the applicable profile as enumerated in the table of minimum support requirements for that profile.

## 6.3 Minimum support requirements

### 6.3.1 Minimum support requirements for generators

There is no minimum complexity which is required of (or appropriate for) X3D generators. Any compliant set of nodes of arbitrary complexity may be generated, as appropriate to represent application content, and which produce only the nodes allowed by the applicable profile, components, and version.

### 6.3.2 Minimum support requirements for browsers

Each profile defines the minimum complexity which shall be supported by an X3D browser in support of that profile. Browser implementations may choose to support

greater limits but may not reduce the limits described for the applicable profile. When the X3D file contains nodes which exceed the limits implemented by the browser, the results are undefined. The words "optionally supported" in the minimum browser support column refers only to the display of the item; in particular, *set\_* events to ignored inputOutput fields shall still generate corresponding *\_changed* events. Where latitude is specified in a table of minimum support requirements for a particular node, full support is required for other aspects of that node.





## Extensible 3D (X3D) Part 1: Architecture and base components

### 27 NURBS component



#### 27.1 Introduction

##### 27.1.1 Name

The name of this component is "NURBS". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 27.1.2 Overview

This subclause describes the Non-uniform Rational B-Spline (NURBS) component of this part of ISO/IEC 19775. [Table 27.1](#) provides links to the major topics in this subclause.

**Table 27.1 — Topics**

- [27.1 Introduction](#)
  - [27.1.1 Name](#)
  - [27.1.2 Overview](#)
- [27.2 Concepts](#)
  - [27.2.1 Overview of NURBS](#)
  - [27.2.2 NURBS-related nodes](#)
  - [27.2.3 Common geometry fields and correctness](#)
  - [27.2.4 Tessellation strategies](#)
  - [27.2.5 Trimmed NURBS](#)
- [27.3 Abstract types](#)
  - [27.3.1 X3DNurbsControlCurveNode](#)
  - [27.3.2 X3DNurbsSurfaceGeometryNode](#)
  - [27.3.3 X3DParametricGeometryNode](#)
- [27.4 Node reference](#)
  - [27.4.1 Contour2D](#)
  - [27.4.2 ContourPolyline2D](#)
  - [27.4.3 CoordinateDouble](#)
  - [27.4.4 NurbsCurve](#)

- [27.4.5 NurbsCurve2D](#)
- [27.4.6 NurbsOrientationInterpolator](#)
- [27.4.7 NurbsPatchSurface](#)
- [27.4.8 NurbsPositionInterpolator](#)
- [27.4.9 NurbsSet](#)
- [27.4.10 NurbsSurfaceInterpolator](#)
- [27.4.11 NurbsSweptSurface](#)
- [27.4.12 NurbsSwungSurface](#)
- [27.4.13 NurbsTextureCoordinate](#)
- [27.4.14 NurbsTrimmedSurface](#)
- [27.5 Support levels](#)
- [Figure 27.1 — NurbsCurve](#)
- [Figure 27.2 — NurbsPatchSurface](#)
- [Figure 27.3 — NurbsSweptSurface](#)
- [Figure 27.4 — NurbsSwungSurface](#)
- [Figure 27.5 — NurbsTrimmedSurface](#)
- [Table 27.1 — Topics](#)
- [Table 27.2 — NURBS component support levels](#)

## 27.2 Concepts

### 27.2.1 Overview of NURBS

Non-uniform Rational B-Splines (NURBS) provide a convenient and efficient manner to generate curved lines and surfaces which can be smooth at any viewing distance. Since these surfaces are generated parametrically, only a small amount of data need be provided for describing complex surfaces.

### 27.2.2 NURBS-related nodes

The characteristics of a NURBS surfaces and curves are defined according to the mathematical definitions for Non-Uniform Rational B-Spline geometry.

There are many construction techniques including:

- a. special cases of NURBS surfaces such as sphere, cylinder or Bezier surfaces;
- b. Extrusion/swept surfaces, constructed given a spine curve and a cross-section curve either or both of which can be NURBS curves;
- c. surfaces of revolution, constructed given a circle/arc and a NURBS cross-section curve;
- d. skinned surfaces constructed from a set of curves;
- e. Gordon surfaces interpolating two sets of curves;
- f. Coons patches, a bi-cubic blended surface constructed from four border curves;
- g. Surfaces interpolating a set of points.

For this standard, it is assumed that creation of such surfaces is only a construction step at authoring time and that the surface will be represented as one of the [X3DParametricGeometryNode](#) nodes for X3D run-time delivery.

### 27.2.3 Common geometry fields and correctness

Background information on NURBS and some implementation strategies are described in [[NURBS](#)].

NURBs require input to be specified using control points, weights, knots and the order. Each of these inputs are defined using separate fields of the appropriate data type.

The control points and the corresponding weight values are held in separate fields. This separation also allows independent animation of the *controlPoint* fields using a [CoordinateInterpolator](#) node.

All nodes that use NURBs principles use the same field names (or u/v variations on them for the surface case). Those field names shall be interpreted as follows:

*order* defines the order of curve. From a mathematical point of view, the curve is defined by a polynomial of the degree  $order-1$ . The value of *order* shall be greater than or equal to 2. An implementation may limit order to a certain number. If it does so, then a warning shall be generated and the surface not displayed. An implementation shall at least support orders 2,3 and 4. The number of control points shall be at least equal to the order of the curve. The order defines the number of adjacent control points that influence a given control point.

*controlPoint* defines the [X3DCoordinateNode](#) instance that provides the source of coordinates used to control the curve or surface. Depending on the weight value and the order, this piecewise linear curve is approximated by the resulting parametric curve. The number of control points shall be equal to or greater than the order. A closed B-Spline curve can be specified by repeating the limiting control points, specifying a periodic knot vector, and setting the *closed* field to `TRUE`. If the last control point is not identical to the first or there exists a non-unitary value of weight within  $(order-1)$  control points of the seam, the *closed* field is ignored.

A *weight* value that shall be greater than zero is assigned to each *controlPoint*. The ordering of the values is equivalent to the ordering of the control point values. The number of values shall be identical to the number of control points. If the length of the weight vector is 0, the default weight 1.0 is assumed for each control point, thus defining a non-Rational curve. If the number of weight values is less than the number of control points, all weight values shall be ignored and a value of 1.0 shall be used.

*knots* defines the knot vector. The number of knots shall be equal to the number of control points plus the order of the curve. The order shall be non-decreasing. Within the knot vector there may not be more than  $order-1$  consecutive knots of equal value. If the length of a knot vector is 0 or not the exact number required ( $numcontrolPoint + order$ ), a default uniform knot vector is computed.

### 27.2.4 Tessellation strategies

Because low-level real-time rendering systems currently can handle only planar



triangles, a NURBS surface needs to be broken down (*i.e.*, tessellated) into a set of triangles approximating the true surface.

Tessellation can be done in different coordinate spaces:

- a. Tessellation in object space and the internal computation of the equivalent to an X3D [IndexedFaceSet](#).
- b. Transforming the control vertices to screen space, and tessellation in screen space

There are different methods to determine tessellation points on the surface:

- c. fixed tessellation based on a absolute number of subdivisions;
- d. adaptive tessellation based on chord length;
- e. adaptive tessellation based on the angle between two triangles;
- f. view dependent tessellation, fine tessellation near silhouette edges.

This standard does not specify which method is used to tessellate the surface. However, the implementation shall render the NURBS such that the approximation produces a rendered image in which the edges of the tessellation can not be perceived.

NOTE: Tessellation in screen space requires the ability to pass already transformed vertices for rendering. This requires the application to already light the vertices (see [17 Lighting component](#)) and pass the resulting color and specular RGB values for each vertex of a triangle.

To avoid cracks at the junction of two surfaces, tessellation values of a whole set of surfaces can be specified in a [NurbsSet](#).

## 27.2.5 Trimmed NURBS

The trimming curve specifies a NURBS-curve that limits the NURBS surface in order to create NURBS surfaces that contain holes or have smooth boundaries. Trimming curves are curves in the parametric space of the surface.

A trimming region is defined by a set of closed trimming loops in the parameter space of a surface. When a loop is oriented counter-clockwise, the area within the loop is retained, and the part outside is discarded. When the loop is oriented clockwise, the area within the loop is discarded, and the rest is retained. Loops may be nested, but a nested loop must be oriented oppositely from the loop that contains it. The outermost loop must be oriented counter-clockwise. Clockwiseness is determined by viewing the parametric surface from the side defined by the cross-product between the  $u$  and  $v$  axes of the parametric space.

A trimming loop consists of a connected sequence of NURBS curves and piecewise linear curves. The last point of every curve in the sequence shall be the same as the first point of the next curve, and the last point of the last curve shall be the same as the first point of the first curve. Self intersecting curves are not allowed.

## 27.3 Abstract types

### 27.3.1 *X3DNurbsControlCurveNode*

```

X3DNurbsControlCurveNode : X3DNode {
  MFVec2d [in,out] controlPoint [] (-∞,∞)
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}

```

The *X3DNurbsControlCurveNode* abstract node type is the base type for all node types that provide control curve information in 2D space.

The control points are defined in 2D coordinate space and interpreted according to the descendent node type as well as the user of this node instance.

### 27.3.2 X3DNurbsSurfaceGeometryNode

```

X3DNurbsSurfaceGeometryNode : X3DParametricGeometryNode {
  SFNode [in,out] controlPoint NULL [X3DCoordinateNode]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFNode [in,out] texCoord NULL [X3DTextureCoordinateNode|NurbsTextureCoordinate]
  SFInt32 [in,out] uTessellation 0 (-∞,∞)
  SFInt32 [in,out] vTessellation 0 (-∞,∞)
  MFDouble [in,out] weight [] (0,∞)
  SFBool [] solid TRUE
  SFBool [] uClosed FALSE
  SFInt32 [] uDimension 0 [0,∞)
  MFDouble [] uKnot [] (-∞,∞)
  SFInt32 [] uOrder 3 [2,∞)
  SFBool [] vClosed FALSE
  SFInt32 [] vDimension 0 [0,∞)
  MFDouble [] vKnot [] (-∞,∞)
  SFInt32 [] vOrder 3 [2,∞)
}

```

The *X3DNurbsSurfaceGeometryNode* represents the abstract geometry type for all types of NURBS surfaces.

*uDimension* and *vDimension* define the number of control points in the u and v dimensions.

*uOrder* and *vOrder* define the order of the surface in the u and v dimensions.

*uKnot* and *vKnot* define the knot values of the surface in the u and v dimensions.

*uClosed* and *vClosed* define whether or not the specific dimension is to be evaluated as a closed surface along the u and v directions, respectively.

*controlPoint* defines a set of control points of dimension *uDimension* × *vDimension*. This set of points defines a mesh where the points do not have a uniform spacing.

*uDimension* points define a polyline in u-direction followed by further u-polylines with the v-parameter in ascending order. The number of control points shall be equal or greater than the order. A closed surface shall be specified by repeating the limiting control points and setting the *closed* field to `TRUE`. If the *closed* field is set to `FALSE`, the implementation shall not be required to smoothly blend the edges of the surface in that dimension into a continuous surface. A closed surface in either the u-dimension or the v-dimension shall be specified by repeating the limiting control points for that dimension and setting the respective *uClosed* or *vClosed* field to `TRUE`. If the last control point is not identical with the first control point, the field is ignored. If either the *uClosed* or the *vClosed* field is set to `FALSE`, the implementation shall not be required to smoothly blend the edges of the surface in that dimension into a continuous surface.

The control vertex corresponding to the control point P[i,j] on the control grid is:

```

P[i,j].x = controlPoint[i + (j × uDimension)].x
P[i,j].y = controlPoint[i + (j × uDimension)].y
P[i,j].z = controlPoint[i + (j × uDimension)].z
P[i,j].w = weight[ i + (j × uDimension)]

```

where  $0 \leq i < uDimension$  and  
 $0 \leq j < vDimension$ .

For an implementation subdividing the surface in a equal number of subdivision steps, tessellation values **could** **might** be interpreted in the following way:

- a. if a tessellation value is greater than 0, the number of tessellation points is:

$$tessellation + 1;$$

- b. if a tessellation value is smaller than 0, the number of tessellation points is:

$$-tessellation \times (u/v)dimension + 1;$$

- c. if a tessellation value is 0, the number of tessellation points is:

$$(2 \times (u/v)dimension) + 1.$$

For implementations doing tessellations based on chord length, tessellation values less than zero are interpreted as the maximum chord length deviation in pixels. Implementations doing fully automatic tessellation may ignore the tessellation hint parameters.

*texCoord* provides additional information on how to generate texture coordinates. By default, texture coordinates in the unit square (or cube for 3D coordinates) are generated automatically from the parametric subdivision. A [NurbsTextureCoordinate](#) node or simply a [TextureCoordinate](#) node can then be used to compute a texture coordinate given a u/v parameter of the [X3DParametricGeometryNode](#). The NurbsTextureCoordinate also supports non-animated surfaces to specify a "chord length"-based texture coordinate parametrization.

The *solid* field determines whether the surface is visible when viewed from the inside. [11.2.3 Common geometry fields](#) provides a complete description of the *solid* field. When *solid*=TRUE is used, the surface shall be visible only from the side that appears ccw (counter-clockwise) on the screen, assuming a surface's quads would be rendered in this order:

```
point(u , v );
point(u-1, v );
point(u-1, v-1);
point(u , v-1);
```

where u is the parameter generating successive points along the u dimension, and v is the parameter generating successive points along the v dimension.

### 27.3.3 X3DParametricGeometryNode

```
X3DParametricGeometryNode : X3DGeometryNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

The *X3DParametricGeometryNode* abstract node type is the base type for all geometry node types that are created parametrically and use control points to describe the final shape of the surface. How the control points are described and interpreted shall be a property of the individual node type.

## 27.4 Node reference

### 27.4.1 Contour2D

```
Contour2D : X3DNode {
  MFNode [in]  addChildren    [NurbsCurve2D|ContourPolyline2D]
  MFNode [in]  removeChildren [NurbsCurve2D|ContourPolyline2D]
  MFNode [in,out] children    [] [NurbsCurve2D|ContourPolyline2D]
  SFNode [in,out] metadata    NULL [X3DMetadataObject]
}
```

The Contour2D node groups a set of curve segments to a composite contour. The children shall form a closed loop with the first point of the first child repeated as the last point of the last child, and the last point of a segment repeated as the first point of the consecutive one. The segments shall be defined by concrete nodes that implement the [X3DNurbsControlCurveNode](#) abstract type nodes, and shall be enumerated in the child field in consecutive order according to the topology of the contour.

The 2D coordinates used by the node shall be interpreted to lie in the (u, v) coordinate space defined by the NURBS surface.

### 27.4.2 ContourPolyline2D

```
ContourPolyline2D : X3DNurbsControlCurveNode {
  SFNode [in,out] metadata    NULL [X3DMetadataObject]
  MFVec2d [in,out] controlPoint [] (-∞, ∞)
}
```

The ContourPolyline2D node defines a piecewise linear curve segment as a part of a trimming contour in the u,v domain of a surface.

The *controlPoint* field specifies the end points of each segment of the piecewise linear curve.

ContourPolyline2D nodes are used as children of the [Contour2D](#) group.

### 27.4.3 CoordinateDouble

```
CoordinateDouble : X3DCoordinateNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFVec3d [in,out] point   [] (-∞,∞)
}
```

CoordinateDouble is a node type derived from [X3DCoordinateNode](#) that allows the definition of 3D coordinates in double precision floating point values.

### 27.4.4 NurbsCurve

```
NurbsCurve : X3DParametricGeometryNode {
  SFNode [in,out] controlPoint NULL [X3DCoordinateNode]
  SFNode [in,out] metadata    NULL [X3DMetadataObject]
  SFInt32 [in,out] tessellation 0 (-∞,∞)
  MFDouble [in,out] weight     [] (0,∞)
  SFBool [] closed            FALSE
  MFDouble [] knot             [] (-∞,∞)
  SFInt32 [] order             3 [2,∞)
}
```

The NurbsCurve node is a geometry node defining a parametric curve in 3D space (see [Figure 27.1](#))

The *tessellation* field gives a hint to the curve tessellator by setting an absolute number

of subdivision steps. These values shall be greater than or equal to the *Order* field. A value of 0 indicates that the browser choose a suitable tessellation. Interpretation of values below 0 is implementation dependent.

For an implementation subdividing the **curvesurface** into an equal number of subdivision steps, tessellation values are interpreted as follows:

- a. if a tessellation value is greater than 0, the number of tessellation points is:

$$\textit{tessellation} + 1;$$

- b. if a tessellation value is smaller than 0, the number of tessellation points is:

$$-\textit{tessellation} \times (\textit{number of control points}) + 1;$$

- c. if a tessellation value is 0, the number of tessellation points is:

$$(2 \times (\textit{number of control points}) + 1).$$

For implementations doing tessellations based on chord length, tessellation values less than zero are interpreted as the maximum chord length deviation in pixels.

Implementations doing fully automatic tessellation may ignore the tessellation hint parameters.

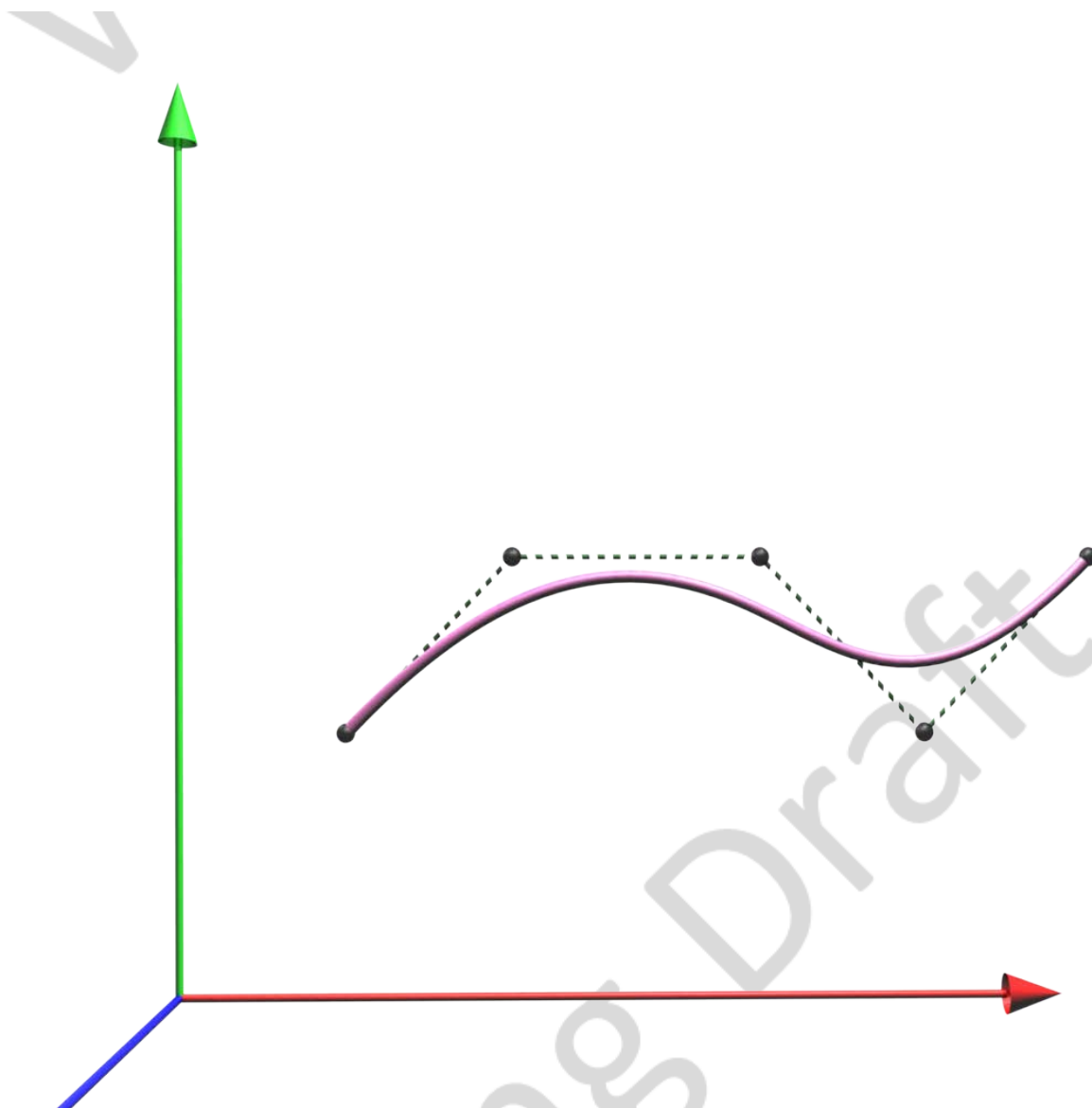


Figure 27.1 — NurbsCurve

## 27.4.5 NurbsCurve2D

```

NurbsCurve2D : X3DNurbsControlCurveNode {
  MFVec2d [in,out] controlPoint [] (-∞,∞)
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFInt32 [in,out] tessellation 0 (-∞,∞)
  MFDouble [in,out] weight [] (0,∞)
  SFBool [] closed FALSE
  MFDouble [] knot [] (-∞,∞)
  SFInt32 [] order 3 [2,∞)
}

```

The NurbsCurve2D node defines a trimming segment that is part of a trimming contour in the  $u,v$  domain of the surface.

NurbsCurve2D nodes are used as children of the [Contour2D](#) group.

## 27.4.6 NurbsOrientationInterpolator

```

NurbsOrientationInterpolator : X3DChildNode {
  SFfloat [in] set_fraction (-∞,∞)
  SFNode [in,out] controlPoint NULL [X3DCoordinateNode]
  MFDouble [in,out] knot [] (-∞,∞)
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}

```

```

SFInt32 [in,out] order      3 (2,∞)
MFDouble [in,out] weight    [] (-∞,∞)
SFRotation [out] value_changed
}

```

NurbsOrientationInterpolator specifies a 3D NURBS curve using the same fields as described for the [NurbsCurve](#) node.

The field *set\_fraction* has the same meaning as in the [NurbsPositionInterpolator](#).

Sending a *set\_fraction* input computes a 3D position on the curve, from which a tangent to the curve at that position is calculated. The tangent direction shall be oriented to point along the curve from the first knot value towards the last value. This orientation value shall be then sent by *value\_changed*. Given the same definition for control points, knots, order and weights, and the same value for *set\_fraction* the orientation interpolator shall generate the orientation of the tangent of the curve at the same position as the NurbsPositionInterpolator.

## 27.4.7 NurbsPatchSurface

```

NurbsPatchSurface : X3DNurbsSurfaceGeometryNode {
  SFNode [in,out] controlPoint NULL [X3DCoordinateNode]
  SFNode [in,out] metadata     NULL [X3DMetadataObject]
  SFNode [in,out] texCoord     NULL [X3DTextureCoordinateNode|NurbsTextureCoordinate]
  SFInt32 [in,out] uTessellation 0 (-∞,∞)
  SFInt32 [in,out] vTessellation 0 (-∞,∞)
  MFDouble [in,out] weight      [] (0,∞)
  SFBool [] solid              TRUE
  SFBool [] uClosed            FALSE
  SFInt32 [] uDimension        0 [0,∞)
  MFDouble [] uKnot            [] (-∞,∞)
  SFInt32 [] uOrder            3 [2,∞)
  SFBool [] vClosed            FALSE
  SFInt32 [] vDimension        0 [0,∞)
  MFDouble [] vKnot            [] (-∞,∞)
  SFInt32 [] vOrder            3 [2,∞)
}

```

The NurbsPatchSurface node is a contiguous NURBS surface patch. [Figure 27.2](#) shows an example of a NurbsPatchSurface node:



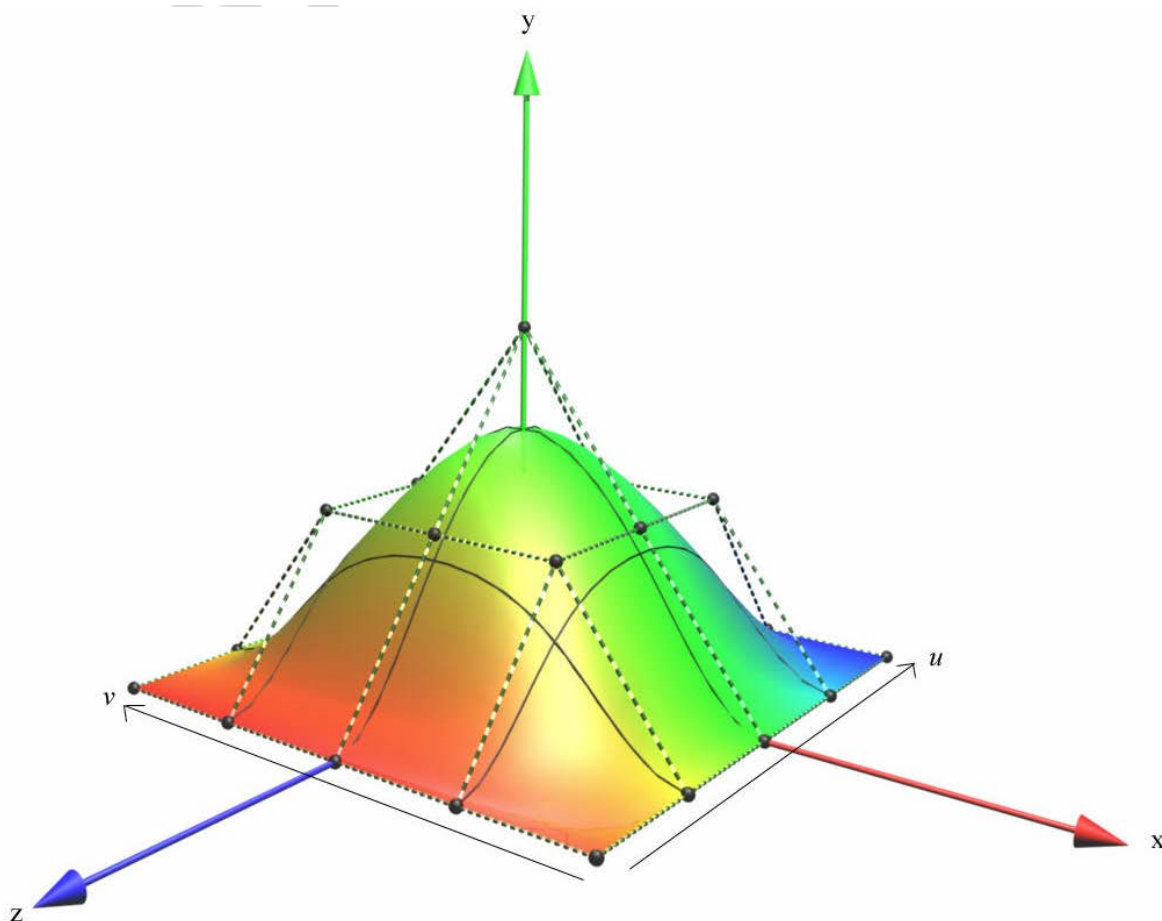


Figure 27.2 — NurbsPatchSurface

## 27.4.8 NurbsPositionInterpolator

```

NurbsPositionInterpolator : X3DChildNode {
  SFFloat [in]  set_fraction  (-∞,∞)
  SFNode  [in,out] controlPoint  NULL [X3DCoordinateNode]
  MFDouble [in,out] knot        [] (-∞,∞)
  SFNode  [in,out] metadata     NULL [X3DMetadataObject]
  SFInt32 [in,out] order        3  (2,∞)
  MFDouble [in,out] weight      [] (-∞,∞)
  SFVec3f [out]  value_changed
}

```

NurbsPositionInterpolator describes a 3D NURBS curve as specified in [27.4.4 NurbsCurve](#).

The fields *set\_fraction* and *value\_changed* have the same meaning as specified in [19.4.6 PositionInterpolator](#).

Sending a *set\_fraction* input computes a 3D position on the curve, which is sent by *value\_changed*. The *set\_fraction* value is used as the input value for the tessellation function. Thereby, the *knot* corresponds to the *key* field of a conventional interpolator node; *i.e.*, if the *set\_fraction* value is within  $[0,1]$  and the knot vector within  $[0,2]$ , only half of the curve is computed.

## 27.4.9 NurbsSet

```

NurbsSet : X3DChildNode, X3DBoundedObject {
  MFNode [in]  addGeometry      [X3DNurbsSurfaceGeometryNode]
}

```

```

MFNode [in]  removeGeometry      [X3DNurbsSurfaceGeometryNode]
MFNode [in,out] geometry        [] [X3DNurbsSurfaceGeometryNode]
SFNode [in,out] metadata        NULL [X3DMetadataObject]
SFFloat [in,out] tessellationScale 1.0 (0,∞)
SFVec3f []   bboxCenter         0 0 0 (-∞,∞)
SFVec3f []   bboxSize           -1 -1 -1 [0,∞) or -1 -1 -1
}

```

The NurbsSet node groups a set of Nurbs surface nodes to a common group for rendering purposes only. This informs the browser that the set of Nurbs surfaces shall be treated as a unit during tessellation to enforce tessellation continuity along borders. The *tessellationScale* parameter is scaling the tessellation values in lower level Nurbs surface nodes. A set of Nurbs surfaces that use a matching set of *controlPoint* along the borders shall result in a common tessellation stepping.

The geometry represented in the children of this node shall not be directly rendered. It is an informational node only. Surfaces not represented elsewhere in the transformation hierarchy shall not be rendered.

The bounds information is provided for optimization purposes only. A browser may choose to use this information about when to apply trimming or smooth tessellation between patches based on the bounds information (EXAMPLE only smooth when the viewer is within the bounds).

## 27.4.10 NurbsSurfaceInterpolator

```

NurbsSurfaceInterpolator : X3DChildNode {
SFVec2f [in]  set_fraction      (-∞,∞)
SFNode [in,out] controlPoint    NULL [X3DCoordinateNode]
SFNode [in,out] metadata        NULL [X3DMetadataObject]
MFDouble [in,out] weight        [] (-∞,∞)
SFVec3f [out] position_changed
SFVec3f [out] normal_changed
SFInt32 []   uDimension          0 [0,∞)
MFDouble []   uKnot              [] (-∞,∞)
SFInt32 []   uOrder              3 [2,∞)
SFInt32 []   vDimension          0 [0,∞)
MFDouble []   vKnot              [] (-∞,∞)
SFInt32 []   vOrder              3 [2,∞)
}

```

NurbsSurfaceInterpolator describes a 3D NURBS surface as specified in [27.4.7 NurbsPatchSurface](#).

Sending a *set\_fraction* input computes a 3D position and normal on the surface for the given *u* and *v* coordinates. The computed position on the surface shall be sent by *position\_changed*, and the computed normal shall be sent by *normal\_changed*.

Normals generated by *normal\_changed* events shall point from the ccw (counter-clockwise) side of the surface, assuming the order of surface quads is as specified for *X3DNurbsSurfaceGeometryNode*.

## 27.4.11 NurbsSweptSurface

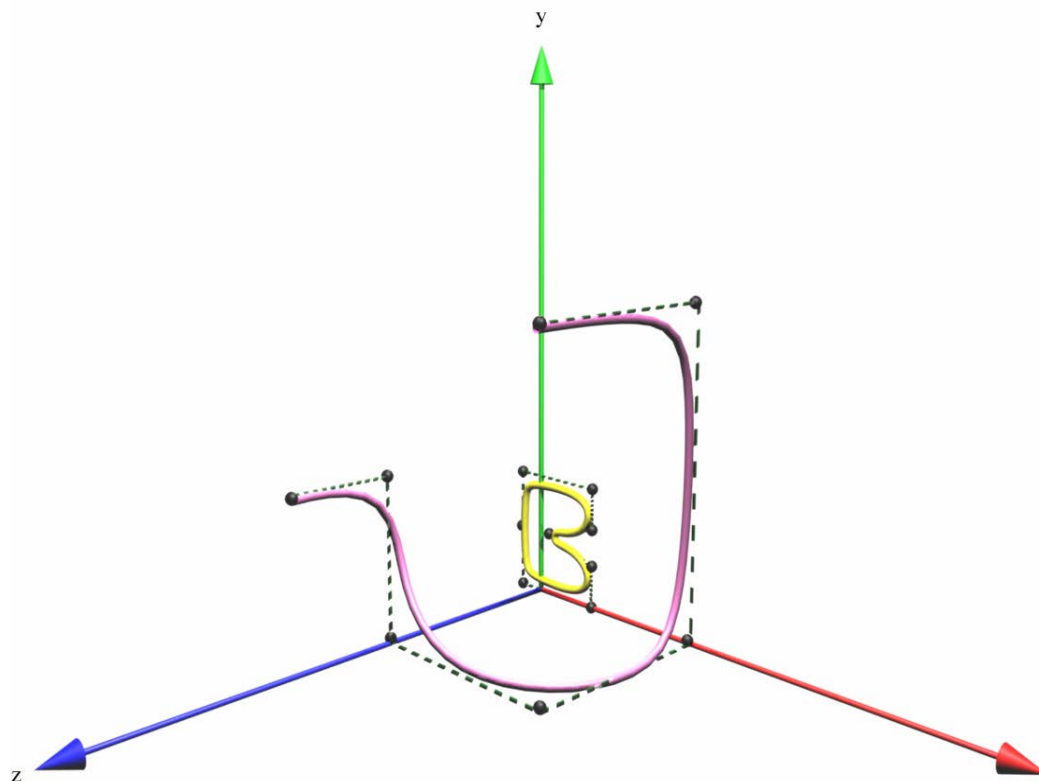
```

NurbsSweptSurface : X3DParametricGeometryNode {
SFNode [in,out] crossSectionCurve NULL [X3DNurbsControlCurveNode]
SFNode [in,out] metadata          NULL [X3DMetadataObject]
SFNode [in,out] trajectoryCurve   NULL [NurbsCurve]
SFBool []   ccw                   TRUE
SFBool []   solid                  TRUE
}

```

NurbsSweptSurface describes a generalized surface that defines a path in 2D space and constant cross section that may be 2D or 3D of the path as illustrated in [Figure 27.3](#). Conceptually it is the NURBS equivalent of the [Extrusion](#) node (see [13.3.5 Extrusion](#))

but permits the use of non-closed cross sections.



**Figure 27.3 — NurbsSweptSurface**

The *solid* and *ccw* fields are defined as specified in [11.2.3 Common geometry fields](#). To have the polygons' normals facing away from the axis, the trajectory curve should be oriented so that it is moving counterclockwise when looking down the  $-Y$  axis, thus defining a concept of "inside" and "outside".

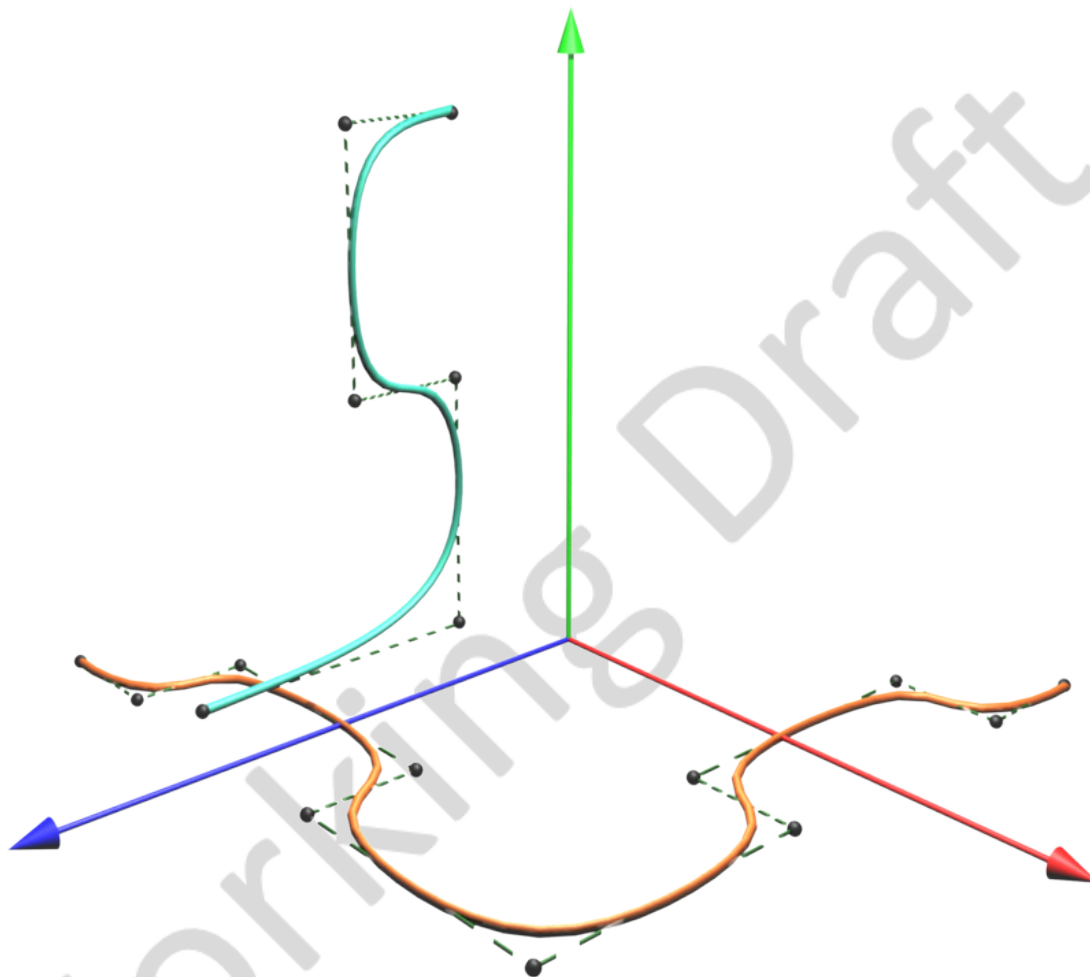
With *solid* `TRUE` and *ccw* `TRUE`, the cylinder is visible from the outside. Changing *ccw* to `FALSE` makes it visible from the inside.

## 27.4.12 NurbsSwungSurface

```
NurbsSwungSurface : X3DParametricGeometryNode {
  SFNode [in,out] metadata      NULL [X3DMetadataObject]
  SFNode [in,out] profileCurve  NULL [X3DNurbsControlCurveNode]
  SFNode [in,out] trajectoryCurve NULL [X3DNurbsControlCurveNode]
  SFBool []      ccw            TRUE
  SFBool []      solid          TRUE
}
```

NurbsSwungSurface describes a generalized surface that defines a path and constant cross section of the path as illustrated in [Figure 27.4](#).





**Figure 27.4 — NurbsSwungSurface**

The profile curve is a 2D curve in the yz-plane that describes the cross-sectional shape of the object.

The trajectory curve is a 2d curve in the xz-plane that describes the path over which to trace the cross-section.

The *solid* and *ccw* fields are defined in [11.2.3 Common geometry fields](#). To have the normals of the polygons facing away from the axis, the trajectory curve should be oriented so that it is moving counterclockwise when looking down the  $-Y$  axis, thus defining a concept of "inside" and "outside".

With *solid* `TRUE` and *ccw* `TRUE`, the cylinder is visible from the outside. Changing *ccw* to `FALSE` specifies that the cylinder is visible from the inside.

### 27.4.13 NurbsTextureCoordinate

```
NurbsTextureCoordinate : X3DNode {
  MFVec2f [in,out] controlPoint [] (-∞,∞)
  SFNode [in,out] metadata NULL [X3DMetadataObject]
```

```

MFFloat [in,out] weight [] (0,∞)
SFInt32 [] uDimension 0 [0,∞)
MFDouble [] uKnot [] (-∞,∞)
SFInt32 [] uOrder 3 [2,∞)
SFInt32 [] vDimension 0 [0,∞)
MFDouble [] vKnot [] (-∞,∞)
SFInt32 [] vOrder 3 [2,∞)
}

```

The `NurbsTextureCoordinate` node is a NURBS surface existing in the parametric domain of its surface host specifying the mapping of the texture onto the surface.

The parameters are as specified in [X3DNurbsSurfaceGeometryNode](#) with the exception that the control points are specified in (u, v) coordinates.

The tessellation process generates 2D texture coordinates. If a `NurbsTextureCoordinate` is undefined, texture coordinates are computed by the client on the basis of parametric step size. Conventional vertex parameters do not apply on NURBS surfaces because triangles are only available after polygonalization. However, the conventional texture transform may be used.

`NurbsTextureCoordinate` nodes are accessed through the `texCoord` field of a node derived from [X3DNurbsSurfaceGeometryNode](#). A `NurbsTextureCoordinate` node separately encountered is ignored.

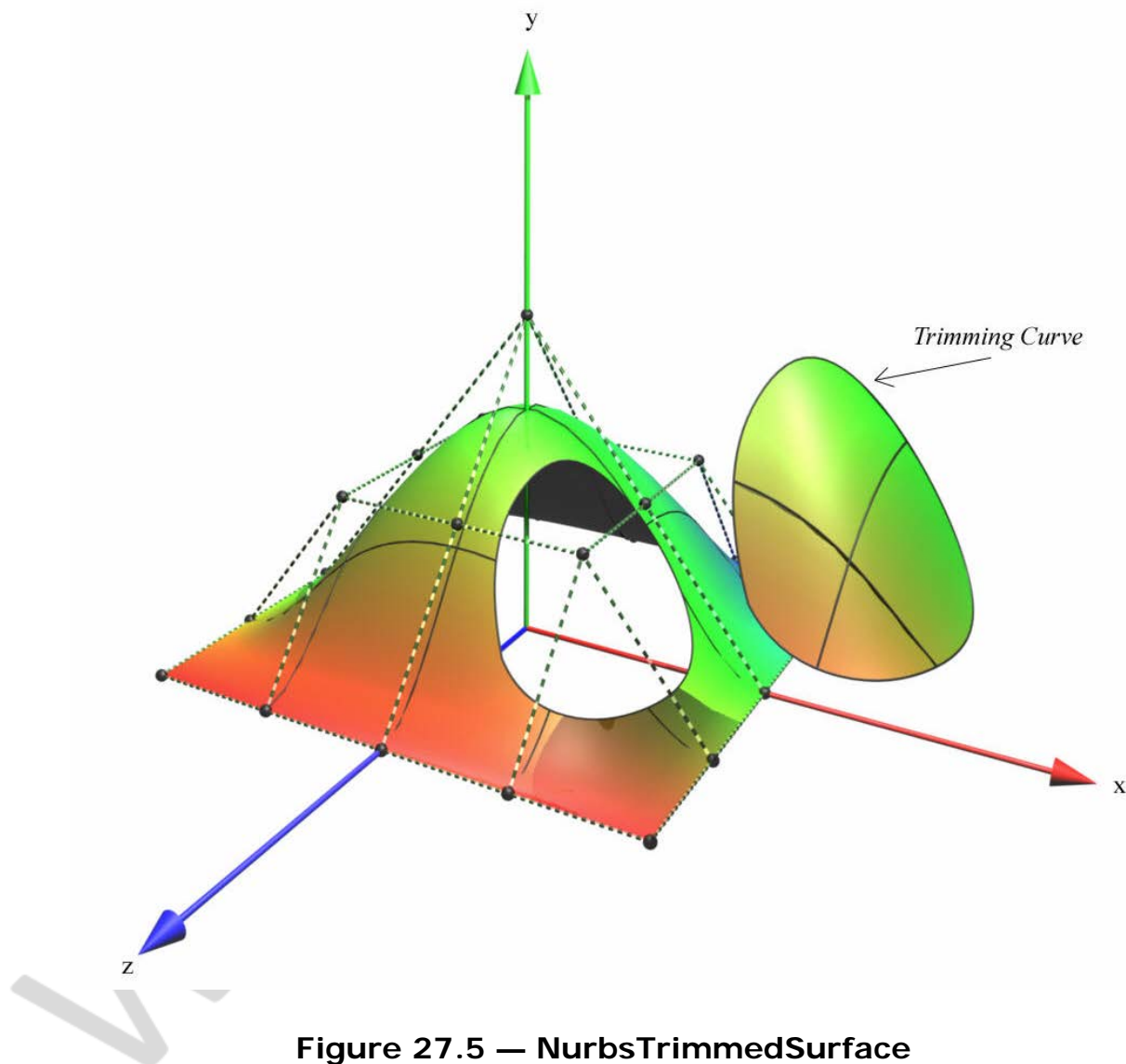
## 27.4.14 NurbsTrimmedSurface

```

NurbsTrimmedSurface : X3DNurbsSurfaceGeometryNode {
  MFNode [in] addTrimmingContour [Contour2D]
  MFNode [in] removeTrimmingContour [Contour2D]
  SFNode [in,out] controlPoint NULL [X3DCoordinateNode]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFNode [in,out] texCoord NULL [X3DTextureCoordinateNode|NurbsTextureCoordinate]
  MFNode [in,out] trimmingContour [] [Contour2D]
  SFInt32 [in,out] uTessellation 0 (-∞,∞)
  SFInt32 [in,out] vTessellation 0 (-∞,∞)
  MFDouble [in,out] weight [] (0,∞)
  SFBool [] solid TRUE
  SFBool [] uClosed FALSE
  SFInt32 [] uDimension 0 [0,∞)
  MFDouble [] uKnot [] (-∞,∞)
  SFInt32 [] uOrder 3 [2,∞)
  SFBool [] vClosed FALSE
  SFInt32 [] vDimension 0 [0,∞)
  MFDouble [] vKnot [] (-∞,∞)
  SFInt32 [] vOrder 3 [2,∞)
}

```

The `NurbsTrimmedSurface` node defines a NURBS surface (see [27.4.7 NurbsPatchSurface](#)) that is trimmed by a set of trimming loops. The outermost trimming loop shall be defined in a counterclockwise direction. An example of a `NurbsTrimmedSurface` node is shown in [Figure 27.5](#).



**Figure 27.5 — NurbsTrimmedSurface**

The *trimmingContour* field, if specified, shall contain a set of [Contour2D](#) (see [27.4.1 Contour2D](#)) nodes. Trimming loops shall be processed as described for the [Contour2D](#) node. If no trimming contours are defined, the [NurbsTrimmedSurface](#) node shall have the same semantics as the [NurbsPatchSurface](#) node.

## 27.5 Support levels

The Non-uniform Rational B-spline (NURBS) component provides four levels of support as specified in [Table 27.2](#). Level 1 provides basic NURBS support. Level 2 adds the ability to ensure controlled tessellation along the boundaries between two NURBS surfaces. Level 3 adds specialized NURBS nodes. Level 4 adds trimmed NURBS surfaces.

**Table 27.2 — NURBS component support levels**

Level	Prerequisites	Nodes/Features	Support
	Core 1 Grouping 1		

<b>1</b>	Shape 1 Interpolator 1 Texturing 1		
		<i>X3DNurbsControlCurveNode</i> (abstract)	n/a
		<i>X3DNurbsSurfaceGeometryNode</i> (abstract)	n/a
		<i>X3DParametricGeometryNode</i> (abstract)	n/a
		CoordinateDouble	All fields fully supported.
		NurbsCurve	All fields fully supported.
		NurbsOrientationInterpolator	All fields fully supported.
		NurbsPatchSurface	All fields fully supported.
		NurbsPositionInterpolator	All fields fully supported.
		NurbsSurfaceInterpolator	All fields fully supported.
		NurbsTextureCoordinate	All fields fully supported.
<b>2</b>	Core 1 Grouping 1 Shape 1 Interpolator 1 Texturing 1		
		All Level 1 NURBS nodes	As supported in Level 1.
		NurbsSet	All fields fully supported.
<b>3</b>	Core 1 Grouping 1 Shape 1 Interpolator 1 Texturing 1		
		All Level 2 NURBS nodes	As supported in Level 2.



		NurbsCurve2D	All fields fully supported.
		ContourPolyline2D	All fields fully supported.
		NurbsSweptSurface	All fields fully supported.
		NurbsSwungSurface	All fields fully supported.
<b>4</b>	Core 1 Grouping 1 Shape 1 Interpolator 1 Texturing 1		
		All Level 3 NURBS nodes	As supported in Level 3.
		Contour2D	All fields fully supported.
		NurbsTrimmedSurface	All fields fully supported.





## Extensible 3D (X3D) Part 1: Architecture and base components

### Annex F

(normative)

### Full profile

---



#### ● F.1 General

This annex defines the X3D components which comprise the Full profile. This includes not only the nodes which shall be supported but also which fields in the supported nodes may be ignored.

The Full profile of X3D is comprised of all features of the standard.

#### ● F.2 Topics

[Table F.1](#) provides links to the major topics in this annex.

**Table F.1 — Topics**

- [F.1 General](#)
- [F.2 Topics](#)
- [F.3 Component support](#)
- [F.4 Conformance criteria](#)
- [F.5 Node set](#)
- [F.6 Other limitations](#)
  
- [Table F.1 — Topics](#)
- [Table F.2 — Components and levels](#)
- [Table F.3 — Nodes for conforming to the Full profile](#)
- [Table F.4 — Other limitations](#)

## F.3 Component support

[Table F.2](#) lists the components and their levels which shall be supported in the Full profile. Tables F.2 and F.3 describe limitations on required support for nodes and fields contained within these components.

**Table F.2 — Components and levels**

Component	Level	Reference
Core	2	<a href="#">7.5 Support levels</a>
Time	2	<a href="#">8.5 Support levels</a>
Networking	3	<a href="#">9.5 Support levels</a>
Grouping	3	<a href="#">10.5 Support levels</a>
Rendering	5	<a href="#">11.5 Support levels</a>
Shape	4	<a href="#">12.5 Support levels</a>
Geometry3D	4	<a href="#">13.4 Support levels</a>
Geometry2D	2	<a href="#">14.4 Support levels</a>
Text	1	<a href="#">15.5 Support levels</a>
Sound	1	<a href="#">16.5 Support levels</a>
Lighting	3	<a href="#">17.5 Support levels</a>
Texturing	3	<a href="#">18.5 Support levels</a>
Interpolation	5	<a href="#">19.5 Support levels</a>
Pointing device sensor	1	<a href="#">20.5 Support levels</a>
Key device sensor	2	<a href="#">21.5 Support levels</a>
Environmental sensor	3	<a href="#">22.5 Support levels</a>
Navigation	3	<a href="#">23.4 Support levels</a>
Environmental effects	4	<a href="#">24.5 Support levels</a>
Geospatial	2	<a href="#">25.4 Support levels</a>
Humanoid animation	1	<a href="#">26.4 Support levels</a>
Non-uniform Rational B-Spline (NURBS)	4	<a href="#">27.5 Support levels</a>

Distributed interactive simulation	2	<a href="#">28.4 Support levels</a>
Scripting	1	<a href="#">29.5 Support levels</a>
Event utilities	1	<a href="#">30.5 Support levels</a>
Programmable shaders	1	<a href="#">31.5 Support levels</a>
CAD geometry	2	<a href="#">32.5 Support levels</a>
Texturing3D	2	<a href="#">33.5 Support levels</a>
Cube map environmental texturing	3	<a href="#">34.5 Support levels</a>
Layering component	1	<a href="#">35.5 Support levels</a>
Layout component	2	<a href="#">36.5 Support levels</a>
Rigid body physics component	2	<a href="#">37.5 Support levels</a>
Picking sensor component	3	<a href="#">38.5 Support levels</a>
Followers component	1	<a href="#">39.5 Support levels</a>
Particle systems component	3	<a href="#">40.5 Support levels</a>
Volume rendering component	4	<a href="#">41.5 Support levels</a>

## F.4 Conformance criteria

Conformance to this profile shall include conformance criteria defined by the specifications for those components and levels listed in [Table F.2](#).

In [Table F.3](#) and [Table F.4](#), the first column defines the item for which conformance is being defined. In some cases, general limits are defined but are later overridden in specific cases by more restrictive limits. The second column defines the requirements for a X3D file conforming to the Full profile; if a X3D file contains any items that exceed these limits, it may not be possible for a X3D browser conforming to the Full profile to successfully parse that X3D file. The third column defines the minimum complexity for a X3D scene that a X3D browser conforming to the Full profile shall be able to present to the user. Fields flagged as "not supported" may be supported by browsers which conform to the Full profile. The word "ignore" in the minimum browser support column refers only to the display of the item; in particular, *set\_* events to ignored inputOutput fields shall still generate corresponding *\_changed* events.

## F.5 Node set

[Table F.3](#) lists the nodes which shall be supported in the Full profile and specifies any fields in these nodes for which this profile requires less than full support.

**Table F.3 — Nodes for conforming to the Full profile**

Item	X3D File Limit	Minimum Browser Support
Anchor	No restrictions.	Full support
Appearance	No restrictions.	Full support.
Arc2D	No restrictions.	Full support.
ArcClose2D	No restrictions.	Full support.
AudioClip	30 second uncompressed PCM WAV.	30 second uncompressed PCM WAV.
Background	No restrictions.	Full support.
BallJoint	No restrictions.	Full support.
Billboard	Restrictions as for all groups.	Full support except as for all groups.
BlendedVolumeStyle	No restrictions.	Full support.
BooleanFilter	No restrictions.	Full support.
BooleanSequencer	No restrictions.	Full support.
BooleanToggle	No restrictions.	Full support.
BooleanTrigger	No restrictions.	Full support.
BoundaryEnhancementVolumeStyle	No restrictions.	Full support.
BoundedPhysicsModel	No restrictions.	Full support.
Box	No restrictions.	Full support.
CADAssembly	No	Full support.

	restrictions.	
CADFace	No restrictions.	Full support.
CADLayer	No restrictions.	Full support.
CADPart	No restrictions.	Full support.
CartoonVolumeStyle	No restrictions.	Full support.
Circle2D	No restrictions.	Full support.
ClipPlane	At least six planes.	Full support.
CollidableOffset	No restrictions.	Full support.
CollidableShape	No restrictions.	Full support.
Collision	Restrictions as for all groups.	Full support except as for all groups. Any navigation behaviour acceptable when collision occurs.
CollisionCollection	No restrictions.	Full support.
CollisionSensor	No restrictions.	Full support.
CollisionSpace	No restrictions.	Full support.
Color	15,000 colours.	15,000 colours.
ColorChaser	No restrictions.	Full support.
ColorDamper	No restrictions.	Full support.
ColorInterpolator	Restrictions as for all interpolators.	Full support except as for all interpolators.
	15,000	

ColorRGBA	colours.	15,000 colours.
ComposedCubeMapTexture	No restrictions.	Full support.
ComposedShader	No restrictions.	Full support.
ComposedTexture3D	No restrictions.	Full support.
ComposedVolumeStyle	No restrictions.	Full support.
Cone	No restrictions.	Full support.
ConeEmitter	No restrictions.	Full support.
Contact	No restrictions.	Full support.
Contour2D	No restrictions.	Full support.
ContourPolyline2D	1500 control points.	Order 30.
Coordinate	15,000 points.	15,000 points.
CoordinateChaser	No restrictions.	Full support.
CoordinateDamper	No restrictions.	Full support.
CoordinateDouble	15,000 points.	15,000 points.
CoordinateInterpolator	15,000 coordinates per <i>keyValue</i> . Restrictions as for all interpolators.	15,000 coordinates per <i>keyValue</i> . Support as for all interpolators.
CoordinateInterpolator2D	15,000 coordinates per <i>keyValue</i> . Restrictions as for all interpolators.	15,000 coordinates per <i>keyValue</i> . Support as for all interpolators.
Cylinder	No	Full support.



	restrictions.	
CylinderSensor	No restrictions.	Full support.
DirectionalLight	No restrictions.	Full support.
DISEntityManager	No restrictions.	Full support.
DISEntityTypeMapping	No restrictions.	Full support.
Disk2D	No restrictions.	Full support.
DoubleAxisHingeJoint	No restrictions.	Full support.
EaseInEaseOut	No restrictions.	Full support.
EdgeEnhancementVolumeStyle	No restrictions.	Full support.
ElevationGrid	16,000 heights.	16,000 heights.
EspduTransform	No restrictions.	Full support.
ExplosionEmitter	No restrictions.	Full support.
Extrusion	$(\#crossSection\ points) \times (\#spine\ points) \leq 2,500.$	$(\#crossSection\ points) \times (\#spine\ points) \leq 2,500.$
FillProperties	No restrictions.	Full support.
FloatVertexAttribute	No restrictions.	Full support.
Fog	No restrictions.	Full support.
FogCoordinate	15,000 coordinates.	15,000 coordinates.
		If the values of the text aspects character set,

FontStyle	No restrictions.	<i>family, style</i> cannot be simultaneously supported, the order of precedence shall be: 1) character set 2) <i>family</i> 3) <i>style</i> . Browser must display all characters in ISO 8859-1 character set (see <a href="#">2.1[8859-1]</a> ).
ForcePhysicsModel	No restrictions.	Full support.
GeneratedCubeMapTexture	No restrictions.	Full support.
GeoCoordinate	15,000 points.	15,000 points.
GeoElevationGrid	16,000 heights.	16,000 heights.
GeoLocation	Restrictions as for all groups.	Full support except as for all groups.
GeoLOD	Restrictions as for all groups.	Full support.
GeoMetadata	No restrictions.	Full support.
GeoOrigin	No restrictions.	Full support.
GeoPositionInterpolator	1000 key-value pairs.	1000 key-value pairs.
GeoProximitySensor	No restrictions.	Full support.
GeoTouchSensor	No restrictions.	Full support.
GeoTransform	Restrictions as for all groups.	Full support except as for all groups.
GeoViewpoint	No restrictions.	Full support.
Group	Restrictions as for all groups.	Full support except as for all groups.
HAnimDisplacer	No restrictions.	Full support.
	No	

HAnimHumanoid	restrictions.	Full support.
HAnimJoint	No restrictions.	Full support.
HAnimSegment	No restrictions.	Full support.
HAnimSite	No restrictions.	Full support.
ImageCubeMapTexture	No restrictions.	Full support.
ImageTexture	JPEG ( <a href="#">2.[JPEG]</a> ) and PNG ( <a href="#">2.[115948]</a> ) format. Restrictions as for PixelTexture.	JPEG ( <a href="#">2.[JPEG]</a> ) and PNG ( <a href="#">2.[115948]</a> ) format. Support as for PixelTexture.
ImageTexture3D	No restrictions.	Full support.
IndexedFaceSet	10 vertices per face. 5000 faces. Less than 15,000 indices.	10 vertices per face. 5000 faces. 15,000 indices in any index field.
IndexedLineSet	15,000 total vertices. 15,000 indices in any index field.	15,000 total vertices. 15,000 indices in any index field.
IndexedQuadSet	No restrictions.	Full support.
IndexedTriangleFanSet	15,000 total vertices. 15,000 indices in any index field.	15,000 total vertices. 15,000 indices in any index field.
IndexedTriangleSet	15,000 total vertices. 15,000 indices in any index field.	15,000 total vertices. 15,000 indices in any index field.
	15,000 total vertices.	15,000 total vertices.

IndexedTriangleStripSet	15,000 indices in any index field.	15,000 indices in any index field.
Inline	No restrictions.	Full support except as for all groups.
IntegerSequencer	No restrictions.	Full support.
IntegerTrigger	No restrictions.	Full support.
IsoSurfaceVolumeData	Minimum dimensions: 512 width, 512 height, 512 depth.	Full support.
KeySensor	No restrictions.	Full support.
Layer	No restrictions.	Full support.
LayerSet	At least six layers.	Full support.
Layout	No restrictions.	Full support.
LayoutGroup	No restrictions.	Full support.
LayoutLayer	No restrictions.	Full support.
LinePickSensor	No restrictions.	Full support.
LineProperties	No restrictions.	Full support.
LineSet	15,000 total vertices.	15,000 total vertices.
LoadSensor	No restrictions.	Full support.
LocalFog	No restrictions.	Full support.
LOD	Restrictions as for all groups.	At least first 4 <i>level/range</i> combinations interpreted, and support

		as for all groups.
Material	No restrictions.	Full support
Matrix3VertexAttribute	No restrictions.	Full support.
Matrix4VertexAttribute	No restrictions.	Full support.
MetadataBoolean	No restrictions.	Full support.
MetadataDouble	No restrictions.	Full support.
MetadataFloat	No restrictions.	Full support.
MetadataInteger	No restrictions.	Full support.
MetadataSet	No restrictions.	Full support.
MetadataString	No restrictions.	Full support.
MotorJoint	No restrictions.	Full support.
MovieTexture	MPEG1-Systems and MPEG1-Video formats (see <a href="#">2.[114496-1]</a> ).	MPEG1-Systems and MPEG1-Video formats (see <a href="#">2.[114496-1]</a> ). Display one active movie texture.
MultiTexture	No restrictions.	At least two textures displayed per node with any number specified. Full support.
MultiTextureCoordinate	15,000 coordinates.	15,000 coordinates.
MultiTextureTransform	Restrictions as for all groups.	<i>addChildren</i> optionally supported. <i>removeChildren</i> optionally supported. Otherwise, full support except as for all groups.

NavigationInfo	No restrictions.	Full support.
Normal	15,000 normals	15,000 normals.
NormalInterpolator	15,000 normals per <i>keyValue</i> . Restrictions as for all interpolators.	15,000 normals per <i>keyValue</i> . Support as for all interpolators.
NurbsCurve	1500 control points.	Order 30.
NurbsCurve2D	1500 control points	Order 30.
NurbsOrientationInterpolator	Restrictions as for all interpolators.	Full support except as for all interpolators.
NurbsPatchSurface	1500 control points.	Order 30.
NurbsPositionInterpolator	Restrictions as for all interpolators.	Full support except as for all interpolators.
NurbsSet	No restrictions.	Full support.
NurbsSurfaceInterpolator	Restrictions as for all interpolators.	Full support except as for all interpolators.
NurbsSweptSurface	1500 control points.	Order 30.
NurbsSwungSurface	1500 control points.	Order 30.
NurbsTextureCoordinate	1500 control points.	Order 30.
NurbsTrimmedSurface	1500 control points. 10 contours.	Order 30. 10 contours.
OrientationChaser	No restrictions.	Full support.
OrientationDamper	No restrictions.	Full support.

OrientationInterpolator	Restrictions as for all interpolators.	Full support except as for all interpolators.
OrthoViewpoint	No restrictions.	Full support.
PackagedShader	No restrictions.	Full support.
ParticleSystem	No restrictions.	Full support.
PickableGroup	No restrictions.	Full support.
PixelTexture	512 width. 512 height.	512 width. 512 height. Display fully transparent and fully opaque pixels.
PixelTexture3D	No restrictions.	Full support.
PlaneSensor	No restrictions.	Full support.
PointEmitter	No restrictions.	Full support.
PointLight	No restrictions.	Full support.
PointPickSensor	No restrictions.	Full support.
PointSet	5000 points.	5000 points.
Polyline2D	15,000 vertices.	15,000 vertices.
PolylineEmitter	No restrictions.	Full support.
Polypoint2D	5000 points.	5000 points.
PositionChaser	No restrictions.	Full support.
PositionChaser2D	No restrictions.	Full support.
PositionDamper	No restrictions.	Full support.



PositionDamper2D	No restrictions.	Full support.
PositionInterpolator	Restrictions as for all interpolators.	Full support except as for all interpolators.
PositionInterpolator2D	Restrictions as for all interpolators.	Full support except as for all interpolators.
PrimitivePickSensor	No restrictions.	Full support.
ProgramShader	No restrictions.	Full support.
ProjectionVolumeStyle	No restrictions.	Full support.
ProximitySensor	No restrictions.	Full support.
QuadSet	No restrictions.	Full support.
ReceiverPDU	No restrictions.	Full support.
Rectangle2D	No restrictions.	Full support.
RigidBody	No restrictions.	Full support.
RigidBodyCollection	No restrictions.	Full support.
ScalarChaser	No restrictions.	Full support.
ScalarDamper	No restrictions.	Full support.
ScalarInterpolator	Restrictions as for all interpolators.	Full support except as for all interpolators.
ScreenFontStyle	No restrictions.	Full support.
ScreenGroup	No restrictions.	Full support.
		25 fields of each access

Script	25 fields of each access type.	type. ECMAScript and Java required.
SegmentedVolumeData	Minimum dimensions: 512 width, 512 height, 512 depth.	Full support.
ShadedVolumeStyle	No restrictions.	Full support.
ShaderPart	No restrictions.	Full support.
ShaderProgram	No restrictions.	Full support.
Shape	No restrictions.	Full support.
SignalPDU	No restrictions.	Full support
SilhouetteEnhancementVolumeStyle	No restrictions.	Full support.
SingleAxisHingeJoint	No restrictions.	Full support.
SliderJoint	No restrictions.	Full support.
Sound	No restrictions.	Full support
Sphere	No restrictions.	Full support.
SphereSensor	No restrictions.	Full support.
SplinePositionInterpolator	No restrictions.	Full support.
SplinePositionInterpolator2D	No restrictions.	Full support.
SplineScalarInterpolator	No restrictions.	Full support.
SpotLight	No restrictions.	Full support.

SquadOrientationInterpolator	No restrictions.	Full support.
StaticGroup	No restrictions.	Full support.
StringSensor	100 characters per string. 100 strings.	Full support. 100 characters per string. 100 strings.
SurfaceEmitter	No restrictions.	Full support.
Switch	Restrictions as for all groups.	Full support except as for all groups.
TexCoordChaser2D	No restrictions.	Full support.
TexCoordDamper2D	No restrictions.	Full support.
Text	100 characters per string. 100 strings.	100 characters per string. 100 strings.
TextureBackground	No restrictions.	All fields fully supported. All texture node types supported in texture fields.
TextureCoordinate	15,000 coordinates.	15,000 coordinates.
TextureCoordinate3D	15,000 coordinates.	15,000 coordinates.
TextureCoordinate4D	15,000 coordinates.	15,000 coordinates.
TextureCoordinateGenerator	No restrictions.	Full support.
TextureProperties	No restrictions.	Full support.
TextureTransform	No restrictions.	Full support.
TextureTransform3D	No restrictions.	Full support.
TextureTransformMatrix3D	No restrictions.	Full support.

TimeSensor	No restrictions.	Full support.
TimeTrigger	No restrictions.	Full support.
TouchSensor	No restrictions.	Full support.
Transform	Restrictions as for all groups.	Full support except as for all groups.
TransformSensor	No restrictions.	Full support.
TransmitterPDU	No restrictions.	Full support.
TriangleFanSet	15,000 coordinates.	Full support.
TriangleSet	15,000 coordinates.	Full support.
TriangleSet2D	15,000 coordinates.	Full support.
TriangleStripSet	15,000 coordinates.	Full support.
TwoSidedMaterial	No restrictions.	Full support.
UniversalJoint	No restrictions.	Full support.
Viewpoint	No restrictions.	Full support
ViewpointGroup	No restrictions.	Full support.
Viewport	No restrictions.	Full support.
VisibilitySensor	No restrictions.	Always visible.
VolumeData	Minimum dimensions: 512 width, 512 height, 512 depth.	Full support.
	No	

VolumeEmitter	restrictions.	Full support.
VolumePickSensor	No restrictions.	Full support.
WindPhysicsModel	No restrictions.	Full support.
WorldInfo	No restrictions.	Full support.

## F.6 Other limitations

[Table F.4](#) specifies other aspects of X3D functionality which are supported by this profile. Note that general items refer only to those specific nodes listed in [Table F.3](#).

**Table F.4 — Other limitations**

Item	X3D File Limit	Minimum Browser Support
All groups	500 children	500 children. Ignore <i>bboxCenter</i> and <i>bboxSize</i> .
All interpolators	1000 key-value pairs	1000 key-value pairs.
All lights	8 simultaneous lights	8 simultaneous lights.
Names for DEF/PROTO/field	50 utf8 octets	50 utf8 octets.
All <i>url</i> fields	10 URLs	10 URLs. URN's ignored. Support `http`, `file`, and `ftp` protocols. Support relative URLs where relevant.
Top-level fields	20 fields	20 fields
Top-level functions	20 functions	20 functions
PROTO/EXTERNPROTO	30 fields of each access type	30 fields of each access type
EXTERNPROTO	n/a	URL references to X3D files conforming to the current profile/component configuration.
PROTO definition nesting depth	5 levels	5 levels.

SFBool	No restrictions	Full support.
SFColor	No restrictions	Full support.
SFColorRGBA	No restrictions	Full support.
SFDouble	No restrictions	Full support. Range $\pm 1e\pm 12$ . Precision $1e-7$ .
SFFloat	No restrictions.	Full support.
SFImage	512 width. 512 height.	512 width. 512 height.
SFInt32	No restrictions.	Full support.
SFMatrix3d	No restrictions.	Full support.
SFMatrix3f	No restrictions.	Full support.
SFMatrix4d	No restrictions.	Full support.
SFMatrix4f	No restrictions.	Full support.
SFNode	No restrictions.	Full support.
SFRotation	No restrictions.	Full support.
SFString	30,000 utf8 octets.	30,000 utf8 octets.
SFTime	No restrictions.	Full support.
SFVec2d	15,000 values.	15,000 values.
SFVec2f	15,000 values.	15,000 values.
SFVec3d	15,000 values.	15,000 values.
SFVec3f	15,000 values.	15,000 values.
SFVec4d	15,000 values.	15,000 values.
SFVec4f	15,000 values.	15,000 values.
MFCColor	15,000 values.	15,000 values.
MFCColorRGBA	15,000 values.	15,000 values.
MFDouble	1000 values.	1000 values.
MFFloat	1,000 values.	1,000 values.
MFInt32	20,000 values.	20,000 values.

MFMMatrix3d	256 values.	256 values.
MFMMatrix3f	256 values.	256 values.
MFMMatrix4d	256 values.	256 values.
MFMMatrix4f	256 values.	256 values.
MFNode	500 values.	500 values.
MFRotation	1,000 values.	1,000 values.
MFString	30,000 utf8 octets per string, 10 strings.	30,000 utf8 octets per string, 10 strings.
MFTime	1,000 values.	1,000 values.
MFVec2d	15,000 values.	15,000 values.
MFVec2f	15,000 values.	15,000 values.
MFVec3d	15,000 values.	15,000 values.
MFVec3f	15,000 values.	15,000 values.
MFVec4d	15,000 values.	15,000 values.
MFVec4f	15,000 values.	15,000 values.





## Extensible 3D (X3D) Part 1: Architecture and base components

### 7 Core component



## 7.1 Introduction

### 7.1.1 Name

The name of this component is "Core". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

### 7.1.2 Overview

This clause describes the Core component of this part of ISO/IEC 19775. The Core component supplies the base functionality for the X3D run-time system, including the abstract base node type, field types, the event model, and routing. [Table 7.1](#) lists the major topics in this clause.

**Table 7.1 — Topics**

- [7.1 Introduction](#)
  - [7.1.1 Name](#)
  - [7.1.2 Overview](#)
- [7.2 Concepts](#)
  - [7.2.1 Overview of the core](#)
  - [7.2.2 Bindable children nodes](#)
  - [7.2.3 Sensors](#)
  - [7.2.4 Metadata](#)
    - [7.2.4.1 Overview](#)
    - [7.2.4.2 Data types for metadata](#)
    - [7.2.4.3 Integration with X3D worlds](#)
    - [7.2.4.4 Assigning metadata to an entire X3D world](#)
  - [7.2.5 Abstract X3D structure](#)
    - [7.2.5.1 Organization](#)
    - [7.2.5.2 Header statement](#)
    - [7.2.5.3 PROFILE statement](#)
    - [7.2.5.4 COMPONENT statement](#)



- [7.2.5.5 UNIT statement](#)
- [7.2.5.6 META statement](#)
- [7.2.5.7 ROUTE statement](#)
- [7.2.5.8 PROTO statement](#)
- [7.2.5.9 EXTERNPROTO statement](#)
- [7.3 Abstract types](#)
  - [7.3.1 X3DBindableNode](#)
  - [7.3.2 X3DChildNode](#)
  - [7.3.3 X3DInfoNode](#)
  - [7.3.4 X3DMetadataObject](#)
  - [7.3.5 X3DNode](#)
  - [7.3.6 X3DPrototypeInstance](#)
  - [7.3.7 X3DSensorNode](#)
- [7.4 Node reference](#)
  - [7.4.1 MetadataBoolean](#)
  - [7.4.2 MetadataDouble](#)
  - [7.4.3 MetadataFloat](#)
  - [7.4.4 MetadataInteger](#)
  - [7.4.5 MetadataSet](#)
  - [7.4.6 MetadataString](#)
  - [7.4.7 WorldInfo](#)
- [7.5 Support levels](#)
- [Table 7.1 — Topics](#)
- [Table 7.2 — Core component support levels](#)

## 7.2 Concepts

### 7.2.1 Overview of the core

The Core component provides the minimum functionality required by all X3D-compliant implementations. The Core component supplies the following abstract constructs:

- a. X3D field types descended from the abstract type [X3DField](#);
- b. the base abstract node types [X3DNode](#) and [X3DPrototypeInstance](#);
- c. commonly used interfaces such as [X3DBindableNode](#) and [X3DUrlObject](#);
- d. the X3D event model and routing;
- e. abstract file structure; and
- f. prototyping.

The Core component is a prerequisite component for all other X3D components.

The Core component may be supported at a variety of [levels](#), allowing for a range of implementations that are conformant to the X3D architecture, object model and event model. For more information on these topics, see [4. Concepts](#).

## 7.2.2 Bindable children nodes

Several X3D nodes, such as [Background](#), [TextureBackground](#), [Fog](#), [NavigationInfo](#), and [Viewpoint](#) are bindable children nodes, inheriting from the abstract node type [X3DBindableNode](#). These nodes have the unique behaviour that only one of each type can be bound per layer (*i.e.*, affect the user's experience) at any instant in time. The browser shall maintain an independent, separate stack for each type of bindable node in each layer. If there is no [LayerSet](#) node defined, there shall be only one set of binding stacks that apply to all nodes in the scene graph. Each of these nodes includes a *set\_bind* inputOnly field and an *isBound* outputOnly field. The *set\_bind* inputOnly field is used to move a given node to and from its respective top of stack. A `TRUE` value sent to the *set\_bind* inputOnly field moves the node to the top of the stack; sending a `FALSE` value removes it from the stack. The *isBound* event is output when a given node is:

- a. moved to the top of the stack;
- b. removed from the top of the stack;
- c. pushed down from the top of the stack by another node being placed on top.

That is, *isBound* events are sent when a given node becomes, or ceases to be, the active node. The node at the top of the stack (the most recently bound node) is the active node for its type and is used by the browser to set the world state. If the stack is empty (*i.e.*, either the X3D file has no bindable nodes for a given type or the stack has been popped until empty), the default field values for that node type are used to set world state. The results are undefined if a multiply instanced (DEF/USE) bindable node is bound.

Bindable nodes only affect the binding stacks of the layer in which they are defined.

The following rules describe the behaviour of the binding stack for a node of type *<bindable node>*, ([Background](#), [TextureBackground](#), [Fog](#), [NavigationInfo](#), or [Viewpoint](#)):

- d. During read, the first encountered *<bindable node>* in each layer is bound by pushing it to the top of the *<bindable node>* stack for that layer. Nodes contained within files referenced by [Inline](#) nodes, within the strings passed to the `Browser.createX3DFromString()` method, or within X3D files passed to the `Browser.createX3DFromURL()` method (see [Part 2 of ISO/IEC 19775](#)) are not candidates for the first encountered *<bindable node>*. The first node within a locally defined prototype instance is a valid candidate for the first encountered *<bindable node>*. The first encountered *<bindable node>* sends an *isBound* `TRUE` event.
- e. When a *set\_bind* `TRUE` event is received by a *<bindable node>*,
  1. If it is not on the top of the stack: the current top of stack node sends an *isBound* `FALSE` event. The new node is moved to the top of the stack and becomes the currently bound *<bindable node>*. The new *<bindable node>* (top of stack) sends an *isBound* `TRUE` event.
  2. If the node is already at the top of the stack, this event has no effect.
- f. When a *set\_bind* `FALSE` event is received by a *<bindable node>* in the stack, it is removed from the stack. If it was on the top of the stack,
  1. it sends an *isBound* `FALSE` event;

2. the next node in the stack becomes the currently bound *<bindable node>* (i.e., pop) and issues an *isBound*<sub>TRUE</sub> event.
- g. If a *set\_bind*<sub>FALSE</sub> event is received by a node not in the stack, the event is ignored and *isBound* events are not sent.
- h. When a node replaces another node at the top of the stack, the *isBound*<sub>TRUE</sub> and *isBound*<sub>FALSE</sub> output events from the two nodes are sent simultaneously (i.e., with identical timestamps).
- i. If a bound node is deleted, it behaves as if it received a *set\_bind*<sub>FALSE</sub> event (see f above).

The results are undefined if a bindable node is bound and is the child of an [LOD](#), [Switch](#), or any node or prototype that disables its children. If a bindable node is bound that results in collision with geometry, the browser shall perform its self-defined navigation adjustments as if the user navigated to this point (see [23.3.2 Collision](#)).

## 7.2.3 Sensors

Sensors are nodes that generate events based on external inputs to the scene graph, such as user input, changes to the viewing environment, messages from the network or ticks of the system clock. X3D defines the following types of sensors:

- a. Pointing device sensors (see [20 Point device sensor component](#)),
- b. Environmental sensors (see [22 Environmental sensor component](#)),
- c. Key device sensors (see [21 Key device sensor component](#)),
- d. Load sensors (see [9 Networking component](#)),
- e. Time sensors (see [8 Time component](#)), and
- f. Picking sensors (see [38 Picking sensor component](#)).

Sensors are children nodes in the hierarchy and therefore may be parented by grouping nodes as described in [10.2.1 Grouping and children node types](#).

Each type of sensor defines when an event is generated. The state of the scene graph after several sensors have generated events shall be as if each event is processed separately, in order. If sensors generate events at the same time, the state of the scene graph will be undefined if the results depend on the ordering of the events.

It is possible to create dependencies between various types of sensors.

EXAMPLE A TouchSensor may result in a change to a VisibilitySensor node's transformation, which in turn may cause the VisibilitySensor node's visibility status to change.

For a detailed description of how dependencies among sensors are handled during execution, see [4.4.8.3 Execution model](#).

## 7.2.4 Metadata

### 7.2.4.1 Overview

Metadata is information that is associated with the objects of the X3D world but is not a

direct part of the world representation. This part of ISO/IEC 19775 defines an abstract interface [X3DMetadataObject](#) that identifies a node as containing metadata and metadata nodes that specify metadata values in various data types.

### 7.2.4.2 Data types for metadata

This part of ISO/IEC 19775 specifies four basic representation types: strings, single-precision and double-precision floating point values, booleans, and integers. Each piece of metadata has two additional strings that describe:

- a. the metadata standard (if any) from which the metadata specification emanates, and
- b. the identification for the particular piece of metadata being provided.

The [MetadataSet](#) node is provided to support cases when a specific set of metadata requires more than a single data type.

NOTE Since a metadata node is derived from [X3DNode](#), the metadata node may itself have metadata.

### 7.2.4.3 Integration with X3D worlds

The [X3DNode](#) abstract node type specifies an SFNode field *metadata* that may only be populated with nodes derived from [X3DMetadataObject](#). If the *metadata* field is empty, no metadata is associated with the node. Since all nodes in X3D are derived from [X3DNode](#), metadata may be placed anywhere in an X3D world.

Metadata is not included as part of the depiction of an X3D world. However, metadata nodes may have a DEF name and the values of the fields of a metadata node may be accessed by SAI services and can be accessed using routing.

The content of the value field of a metadata node is not interpreted by the X3D browser.

Metadata may also be attached to other X3D nodes by setting the metadata field of that node to a node derived from the [X3DMetadataNode](#) abstract node type.

### 7.2.4.4 Assigning metadata to an entire X3D world

The META statement (see [7.2.5.5 META statement](#)) may be used to assign metadata to the entire world. The content of the META statement is accessible using the SAI. If it is desired that metadata information that applies to the entire world be provided for access through routing, a [WorldInfo](#) node containing the metadata in its metadata field may be inserted as a root node.

## 7.2.5 Abstract X3D structure

### 7.2.5.1 Organization

An X3D world is conceptually defined as a sequence of statements organized conceptually as a file. The first item in the file is the Header statement. The second item in the file is the PROFILE statement. The PROFILE statement may be optionally followed

by one or more COMPONENT, UNIT and/or META statements in that order. There may be multiple of each of the COMPONENT, UNIT, and/or META statements. The remainder of the file consists of the other elements defined in this part of ISO/IEC 19775.

ROUTE statements are used to specify the pathways for allowed transmission of events. These statements link a field in one node to a field of the same field type in another node.

PROTO statements are used to specify new node types. Such statements assign a name to the new node type along with a declaration of the interface for the new node type. This is followed by a definition for the node type functionality.

EXTERNPROTO statements are used to specify an interface to PROTO or EXTERNPROTO statements located externally to the local file.

Any additional X3D content loaded into the scene via [Inline](#) nodes or scenes loaded using `createX3DFromStream`, `createX3DFromString`, or `createX3DFromUrl`, shall be declared as having a profile that has an equal or smaller set of required functionality; *i.e.*, there can be no components explicitly declared, or implied by the profile in that content, that requires functionality not declared in the original profile and component declarations for the containing scene. Any UNIT statements within the additional contained external X3D content are ignored and the units specified in the root file are used.

Although an X3D world is described as being contained in a file, the file may be conceptual and created dynamically during run-time as described in [Part 2 of ISO/IEC 19775](#).

### 7.2.5.2 Header statement

The Header statement is an encoding-dependent statement containing the following elements:

- a. identification of the standard being supported (for this standard, this is "X3D");
- b. version of the standard being supported (for this version of this part of ISO/IEC 19775, the version number is "3.2");
- c. identification of the character encoding being supported (for this standard, this shall be "UTF-8"), and
- d. optional comments that may apply to the file.

While the exact representation of this information is dependent on the encoding, this information shall always be stored as human-readable text.

### 7.2.5.3 PROFILE statement

Every X3D application shall declare a profile at the beginning of execution. This declaration tells the browser the exact set of components and their support levels that are required for the application to run, allowing for a browser to dynamically load the appropriate components if it so desires, and providing a mechanism for strict conformance should the browser choose to enforce it. If a browser supports the combination of declared profile and components (see [7.2.5.4 COMPONENT statement](#)),

it may proceed with presenting the world; otherwise, it shall fail.

The profile is declared via a PROFILE statement immediately following the Header statement at the top of the file. The form of the PROFILE statement is:

```
PROFILE <name>
```

where name is a string that does not contain whitespace.

The following profiles are defined in this standard:

- a. Core (see [A Core profile](#)),
- b. Interchange (see [B Interchange profile](#)),
- c. Interactive (see [C Interactive profile](#)),
- d. MPEG-4 interactive (see [D MPEG-4 interactive profile](#)),
- e. Immersive (see [E Immersive profile](#)),
- f. Full (see [F Full profile](#)), and
- g. CADInterchange (see [H CAD interchange profile](#)).

The profile name is implicitly qualified by the version number of the standard (see [7.2.5.2 Header statement](#)). Browsers shall use both the profile name and the version number to determine the specific characteristics of the profile.

#### 7.2.5.4 COMPONENT statement

X3D applications may explicitly declare additional components required for the application to run. This is useful for combining features that do not appear together in a predefined profile, such as adding humanoid animation support to the Immersive profile. If a browser supports the combination of declared profile and components, it may proceed with presenting the world; otherwise, it shall fail.

Declaring a component in a file shall only add support for nodes and functionality defined in that component at the requested support level. Nodes that are defined as part of the prerequisite components (see [4.5.3 Base Components](#)) shall not be automatically included. A user shall declare all components and levels for the nodes and/or functionality being used either explicitly through the use of the COMPONENT statement or implicitly through the PROFILE statement.

Components are declared by COMPONENT statements at the top of the file, immediately following the PROFILE statement but preceding any other content. The form of the component statement is:

```
COMPONENT <name> <level>
```

where <name> is a string that does not contain whitespace, and <level> is a positive integer.

If support for a component at the desired level is implied by the application's declared profile, the declaration for that component is unnecessary but may be included.

#### 7.2.5.5 UNIT statement



X3D applications may explicitly alter the initial base units within an X3D world by inserting UNIT statements defining the characteristics of the new default base units. At most one UNIT statement shall be provided for each base unit type. Only the UNIT statements in the root file apply to an X3D world. If no UNIT statements are provided, the initial base units as specified in [4.3.6 Standard units and coordinate system](#) shall apply.

UNIT statements contained in X3D files referenced by Inline nodes or contained in X3D files consisting of EXTERNPROTO bodies shall be used to align effected units to the base units of the root file before the referenced X3D file content is incorporated in the X3D world.

UNIT statements may only be contained in X3D worlds created for X3D version 3.3 or later (as specified in the Header statement). If a version of 3.2 or earlier is specified in conjunction with UNIT statements, the browser shall fail.

A change in a base unit is specified by UNIT statements at the top of the file preceding any element content but in the statement order specified in [7.2.5.1 Organization](#). The form of the UNIT statement is:

```
UNIT <category> <name> <conversionFactor>
```

where <category> is a string specifying one of the categories in [Table 4.2](#), <name> is a string that does not contain whitespace that provides a name for the new default base unit, and <conversion\_factor> is a positive double precision value that converts the new default base unit to the initial base unit specified in [Table 4.2](#). Direct modification of conversion factors for derived units is not allowed.

### 7.2.5.6 META statement

X3D applications may explicitly declare metadata about the world being defined. This is done by adding one or more META statements that contain such information. Such statements do not affect the scene graph but simply provide additional information in the world.

Metadata that applies to the entire file may be specified by META statements at the top of the file preceding any element content but in the statement order specified in [7.2.5.1 Organization](#). The form of the META statement is:

```
META <key> <data>
```

where <key> is a string that identifies the metadata and <data> is a string that defines the value for the metadata identified by <key>.

### 7.2.5.7 ROUTE statement

X3D applications specify connections between fields of one node to fields of other nodes using the ROUTE statement. See [4.4.8.2 Routes](#) for a general discussion of routes.

ROUTE statements may appear anywhere in the file and have the following form:

```
ROUTE <fromNodeName> <fromFieldName> <toNodeName> <toFieldName>
```

where <fromNodeName> identifies the node that will generate an event,

<fromFieldName> is the name of the field in the generating node from which the event will emanate, <toNodeName> identifies the node that will receive an event, and <toFieldName> identifies the field in the destination node that will receive the event.

### 7.2.5.8 PROTO statement

New node types may be defined by X3D applications through use of the PROTO statement as specified in [4.4.4 Prototype semantics](#).

PROTO statements may appear anywhere in the file and have the following form:

```
PROTO <protoName> <protoInterfaceDeclaration> <protoDefinition>
```

The <protoName> specifies the name for the new node type.

The <protoInterfaceDeclaration> specifies a list of field definitions. Each field definition specifies the data type, access type, and name for the field. For initializeOnly and inputOutput fields, the default value is also specified. See [4.4.4.2 PROTO interface declaration semantics](#) for details.

The <protoDefinition> consists of a list of nodes the first of which is used to specify the node type for the prototype. This list may instantiate other prototypes provided that the definitions of the referenced prototypes precede this PROTO statement. See [4.4.4.3 PROTO definition semantics](#) for details.

### 7.2.5.9 EXTERNPROTO statement

Externally defined new node types may be used by X3D applications by referencing their definition using the EXTERNPROTO statement as specified in [4.4.5 External prototype semantics](#).

EXTERNPROTO statements may appear anywhere in the file and have the following form:

```
EXTERNPROTO <externprotoName> <externprotoInterfaceDeclaration> <externprotoURL>
```

The <externprotoName> specifies the name for the new node type.

The <externprotoInterfaceDeclaration> specifies a list of field definitions. Each field definition specifies the data type, access type, and name for the field. The default value for initializeOnly and inputOutput field is derived as specified in [4.4.5.2 EXTERNPROTO interface semantics](#).

The <externprotoURL> specifies the location of the definition for the externally defined prototype. See [4.4.5.3 EXTERNPROTO URL semantics](#) for details.

## 7.3 Abstract types

### 7.3.1 X3DBindableNode

```
X3DBindableNode : X3DChildNode {
  SFBool [in] set_bind
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTime [out] bindTime
  SFBool [out] isBound
```



}

X3DBindableNode is the abstract **basenode** type for all bindable children nodes, including [Background](#), [TextureBackground](#), [Fog](#), [NavigationInfo](#) and [Viewpoint](#). For complete discussion of bindable behaviors, see [7.2.2 Bindable children nodes](#).

### 7.3.2 X3DChildNode

```
X3DChildNode : X3DNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This abstract node type indicates that the concrete nodes that are instantiated based on it may be used in *children*, *addChildren*, and *removeChildren* fields.

More details on the *children*, *addChildren*, and *removeChildren* fields can be found in [10.2.1 Grouping and children node types](#).

### 7.3.3 X3DInfoNode

```
X3DInfoNode : X3DChildNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This is the base node type for all nodes that contain only information without visual semantics.

### 7.3.4 X3DMetadataObject

```
X3DMetadataObject {
  SFString [in,out] name "" (Required)
  SFString [in,out] reference ""
}
```

This abstract interface is the basis for all metadata nodes. The interface is inherited by all metadata nodes.

The specification of a non-empty value for the *name* field is required.

The specification of the *reference* field is optional. If provided, it identifies the metadata standard or other specification that defines the *name* field. If the *reference* field is not provided or is empty, the meaning of the *name* field is considered implicit to the characters in the string.

### 7.3.5 X3DNode

```
X3DNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This abstract node type is the base type for all nodes **and node types** in the X3D system.

### 7.3.6 X3DPrototypeInstance

```
X3DPrototypeInstance : X3DNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This abstract node type is the base type for all prototype instances in the X3D system. Any user-defined nodes declared with `PROTO` or `EXTERNPROTO` are instantiated using this base type. An *X3DPrototypeInstance* may be placed anywhere in the scene graph where it is legal to place the first node declared within the prototype instance. For example, if the base type of first node is *X3DAppearanceNode*, that prototype may be instantiated anywhere in the scene graph that allows for an appearance node (EXAMPLE Shape).

### 7.3.7 X3DSensorNode

```
X3DSensorNode : X3DChildNode {
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFBool [out] isActive
}
```

This abstract node type is the base type for all sensors.

## 7.4 Node reference

### 7.4.1 MetadataBoolean

```
MetadataBoolean : X3DNode, X3DMetadataObject {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFString [in,out] name ""
  SFString [in,out] reference ""
  MFBool [in,out] value []
}
```

The metadata provided by this node is contained in the Boolean values of the *value* field.

### 7.4.2 MetadataDouble

```
MetadataDouble : X3DNode, X3DMetadataObject {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFString [in,out] name ""
  SFString [in,out] reference ""
  MFDouble [in,out] value []
}
```

The metadata provided by this node is contained in the double-precision floating point numbers of the *value* field.

### 7.4.3 MetadataFloat

```
MetadataFloat : X3DNode, X3DMetadataObject {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFString [in,out] name ""
  SFString [in,out] reference ""
  MFFloat [in,out] value []
}
```

The metadata provided by this node is contained in the single-precision floating point numbers of the *value* field.

### 7.4.4 MetadataInteger

```
MetadataInteger : X3DNode, X3DMetadataObject {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFString [in,out] name ""
  SFString [in,out] reference ""
  MFInt32 [in,out] value []
}
```

The metadata provided by this node is contained in the integers of the *value* field.

## 7.4.5 MetadataSet

```
MetadataSet : X3DNode, X3DMetadataObject {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFString [in,out] name ""
  SFString [in,out] reference ""
  MFNode [in,out] value [] [X3DMetadataObject]
}
```

The metadata provided by this node is contained in the metadata nodes of the *value* field.

## 7.4.6 MetadataString

```
MetadataString : X3DNode, X3DMetadataObject {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFString [in,out] name ""
  SFString [in,out] reference ""
  MFString [in,out] value []
}
```

The metadata provided by this node is contained in the strings of the *value* field.

## 7.4.7 WorldInfo

```
WorldInfo : X3DInfoNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFString [] info []
  SFString [] title ""
}
```

The WorldInfo node contains information about the world. This node is strictly for documentation purposes and has no effect on the visual appearance or behaviour of the world. The *title* field is intended to store the name or title of the world so that browsers can present this to the user (perhaps in the window border). Any other information about the world can be stored in the *info* field, such as author information, copyright, and usage instructions.

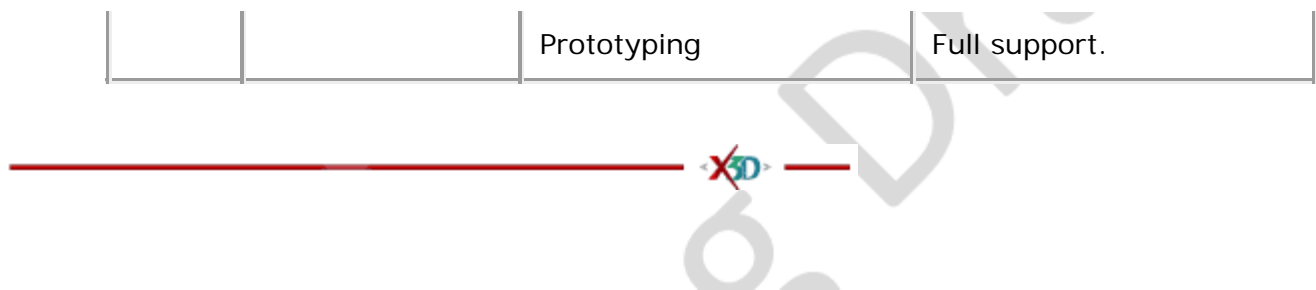
## 7.5 Support levels

The Core component provides two levels of support as specified in [Table 7.2](#). Level 1 provides the minimum basis for all profiles and components. Level 2 adds support for prototypes.

**Table 7.2 — Core component support levels**

Level	Prerequisites	Nodes/Features	Support
<b>1</b>	None		
		<i>X3DBindableNode</i> (abstract)	n/a
		<i>X3DChildNode</i> (abstract)	n/a
		<i>X3DField</i> (abstract)	n/a

		<i>X3DInfoNode</i> (abstract)	n/a
		<i>X3DMetadataObject</i> (abstract)	n/a
		<i>X3DNode</i> (abstract)	n/a
		<i>X3DPrototypeInstance</i> (abstract)	n/a
		<i>X3DSensorNode</i> (abstract)	n/a
		<i>X3DUriObject</i> (abstract)	n/a
		MetadataDouble	All fields are fully supported.
		MetadataFloat	All fields are fully supported.
		MetadataInteger	All fields are fully supported.
		MetadataSet	All fields are fully supported.
		MetadataString	All fields are fully supported.
		WorldInfo	All fields are fully supported.
		Statements: Header PROFILE COMPONENT UNIT META	Full support.
		Field types	All field types.
		Event model	As specified in <a href="#">4.4.8 Event Model</a> .
		Routing	Full support.
		Prototyping	Optionally supported.
<b>2</b>	None		
		All Level 1 Core objects	As supported in Level 1.





## Extensible 3D (X3D) Part 1: Architecture and base components

# 28 Distributed interactive simulation (DIS) component

## 28.1 Introduction

### 28.1.1 Name

The name of this component is "DIS". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

### 28.1.2 Overview

This clause describes the Distributed Interactive Simulation (DIS) component of this International Standard. [Table 28.1](#) provides links to the major topics in this clause.

**Table 28.1 — Topics**

- [28.1 Introduction](#)
  - [28.1.1 Name](#)
  - [28.1.2 Overview](#)
- [28.2 Concepts](#)
  - [28.2.1 Overview of DIS](#)
  - [28.2.2 Network communications](#)
  - [28.2.3 Common DIS fields](#)
- [28.3 Node reference](#)
  - [28.3.1 DISEntityManager](#)
  - [28.3.2 DISEntityTypeMapping](#)
  - [28.3.3 EspduTransform](#)
  - [28.3.4 ReceiverPdu](#)
  - [28.3.5 SignalPdu](#)
  - [28.3.6 TransmitterPdu](#)
- [28.4 Support levels](#)
- [Table 28.1 — Topics](#)
- [Table 28.2 — DIS component support levels](#)

## 28.2 Concepts

### 28.2.1 Overview of DIS

IEEE 1278 (see [2.\[IEEE1278\]](#)) is an IEEE communications standard for physically based distributed simulations. Known by the name Distributed Interactive Simulation (DIS), the standard defines the binary layout of a series of messages used to transmit simulation information. Often used by military applications, IEEE 1278 covers a wide range of data, including entity location, velocity, and orientation, and more obscure features such as electronic warfare and supply logistics. In addition to its original focus on military simulations, DIS is also used in civilian applications.

The DIS component consists of the following X3D nodes:

- [DISEntityManager](#),
- [DISEntityTypeMapping](#),
- [EspduTransform](#),
- [ReceiverPdu](#),
- [SignalPdu](#), and
- [TransmitterPdu](#).

Together, these nodes provide the means to send and receive DIS-compliant messages, called Protocol Data Units (PDUs), across the network. Together these nodes support seven DIS PDU message types: Collision, Detonate, Entity State, Fire, Receiver, Signal and Transmitter. Numerous other DIS PDUs are defined by the DIS protocol, but corresponding X3D mappings are not defined.

### 28.2.2 Network communications

DIS messages are typically transmitted on User Datagram Protocol (UDP) (see [\[UDP\]](#)) sockets. Multicast, unicast or broadcast transport mechanisms may be used for network communications. Each of the X3D DIS nodes communicates via a UDP socket, usually multicast-enabled, and uses it to read and/or write DIS messages. These messages can be used to communicate and to modify both position and orientation of virtual entities in the X3D scene among multiple hosts across the network. Each DIS implementation is responsible for managing sockets. New entities are registered by the DIS node to send/receive network updates. "Entities" are a high-level abstraction; in the case of a position update, the actual X3D scene-graph object modified may be a [Transform](#) node (as for [EspduTransform](#)), and the geometry for an animated entity is contained in the corresponding children.

### 28.2.3 Common DIS fields

The DIS nodes have a number of descriptive fields in common relating to the desired behavior of the DIS node. Common fields include message header and content information conforming to the DIS standard, network status, and configuration data needed to establish or modify network communications. Since nodes in the DIS

component can receive data from the network, these nodes are also sensors. Thus, these nodes implement the [X3DSensorNode](#) interface and include both *enabled* and *isActive* fields.

Common fields relating to description of the desired behavior of the DIS node are: *isActive*, *timestamp*, *networkMode*, *isStandAlone*, *isNetworkReader*, *isNetworkWriter*, *readInterval*, and *writeInterval*.

The *isActive* field indicates if the node has received a DIS message (when output as `TRUE`) or not (when output as `FALSE`). Since DIS entities can be considered inactive after some period of time (five seconds is specified as the default in [IEEE 1278](#)) either event may be received by listening nodes. An implementation may use a different value.

The *timestamp* field provides the time (SFTIME) at which the DIS message arrived, referenced to local system time.

The *networkMode* field indicates if the X3D DIS node is operating in one of three distinct ways: independently from the network, as a sender writing updates, or as a receiver reading updates.

- *networkMode* `standAlone` only connects dynamic behavior via local ROUTEs and does not send/receive PDUs to/from the network.
- *networkMode* `networkReader` reads messages at *readInterval* seconds from the network, which can modify fields in the node upon receipt. In this mode, the entity geometry in the DIS node (e.g., [EspduTransform](#)) acts as a remote copy of the entity that sent the PDUs.
- *networkMode* `networkWriter` sends messages at *writeInterval* seconds to the network. In this mode, the entity geometry in the DIS node (e.g., [EspduTransform](#)) acts as the master copy of the entity originating state updates.

Fields *isStandAlone*, *isNetworkReader*, and *isNetworkWriter* are respectively sent as appropriate `TRUE` or `FALSE` events during initialization of the DIS node and whenever *networkMode* is changed. These fields match the state of *networkMode*. One and only one of these three fields can be `TRUE` at any given time.

The *readInterval* field is a time in seconds between checking for receipt of DIS messages. Setting the *readInterval* to zero disables the reading of DIS messages. The *writeInterval* is a time in seconds between message transmissions by the node. Setting the *writeInterval* to zero disables the transmission of DIS messages by the node.

Common fields relating to standard identification of DIS entities are: *siteID*, *applicationID*, and *entityID*.

The *siteID* and *applicationID* fields are used to create the DIS PDU Simulation Address record. The intent for each simulation exercise is for each DIS site to be assigned a unique identifier, and each simulation application at a DIS site assigned an application identifier unique within that site. Both fields are 16-bit, unsigned numbers. A common practice is to assign the four octets of a participant's Internet Protocol (IP) host address to *siteID*. The *entityID* field further identifies the DIS entity that is the subject of the particular PDU (EXAMPLE an Entity State PDU to update the location and orientation of a particular simulation entity). The *entityID* is an unsigned, 16-bit number.



Each entity in a DIS application is assigned a triplet identifier (*siteID*, *applicationID* and *entityID* fields) that is unique across all entities in that application and in the particular exercise. The entity identifier triplet is valid for the duration of the exercise.

Common fields relating to DIS network communications are: *address*, *port*, *multicastRelayHost*, *multicastRelayPort*, *rtpHeaderExpected* and *rtpHeaderHeard*.

The *address* field identifies the multicast address for the message transmission (EXAMPLE "224.2.181.145" or "localhost"). The *port* field identifies the multicast port (EXAMPLE 62040) for sending or receiving DIS messages.

Fields *multicastRelayHost*, *multicastRelayPort*, *rtpHeaderExpected* and *rtpHeaderHeard* provide networking extensions to the IEEE DIS protocol (see [2.\[IEEE1278\]](#)) intended to make DIS more compatible with Internet conventions for unicast and multicast routing over wide-area networks (WANs). If wide-area multicast is needed but not available locally, the *multicastRelayHost* and *multicastRelayPort* fields are provided as a fallback server address and associated port, used for creating a unicast tunnel connection to a multicast-connected relay server. Field *rtpHeaderExpected* indicates that the Real Time Protocol (see [2.\[RFC1889\]](#)) header is expected to be prepended to the DIS PDU message to be sent or received by the node (when the field is set to `TRUE`). Field *rtpHeaderHeard* indicates that the RTP header has been prepended to the incoming DIS message.

The *geoSystem* field is used to define the spatial reference frame and is described in [25.2.3 Specifying a spatial reference frame](#).

The geometry of the nodes in children is to be specified in base length units in X3D coordinates relative to the location specified by the *geoCoords* field. The *geoCoords* field should be provided in the format described in [25.2.3 Specifying a spatial reference frame](#).

The *geoCoords* field can be used to dynamically update the geospatial location of the model; for example, **for example** an event **could** **might** be sent from a `GeoPositionInterpolator` node.

## 28.3 Node reference

### 28.3.2 DISEntityManager

```
DISEntityManager : X3DChildNode {
  SFString [in,out] address      "localhost"
  SFInt32  [in,out] applicationID 1      [0,65535]
  MFNode   [in,out] mapping      []      [DISEntityTypeMapping]
  SFNode   [in,out] metadata     NULL    [X3DMetadataObject]
  SFInt32  [in,out] port         0       [0,65535]
  SFInt32  [in,out] siteID       0       [0,65535]
  MFNode   [out]  addedEntities
  MFNode   [out]  removedEntities
}
```

A `DISEntityManager` node notifies content when new entities arrive or current entities leave.

The *mapping* field provides a mechanism for automatically creating an X3D model for a new entity arriving. If a new entity matches one of the nodes, an instance of the provided URL is created and added as a child to the [EspduTransform](#) specified in the

*addedEntities* field. See [28.3.2 DISEntityTypeMapping](#) for details on matching DIS parameters to URLs.

The *addedEntities* field contains any new entities added last frame. These will be EspduTransform nodes.

The *removedEntities* field contains any entities removed last frame, either from a timeout or from an explicit RemoveEntityPDU action. This will contain a reference to the EspduTransform node.

## 28.3.2 DISEntityTypeMapping

```
DISEntityTypeMapping : X3DInfoNode, X3DUrlObject {
  SFString [in,out] description ""
  SFBool [in,out] load TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTime [in,out] refresh 0.0 [0,∞)
  MFString [in,out] url [] [URI]
  SFInt32 [] category 0 [0,255]
  SFInt32 [] country 0 [0,65535]
  SFInt32 [] domain 0 [0,255]
  SFInt32 [] extra 0 [0,255]
  SFInt32 [] kind 0 [0,255]
  SFInt32 [] specific 0 [0,255]
  SFInt32 [] subcategory 0 [0,255]
}
```

A DISEntityTypeMapping node provides a mapping from DIS Entity type information to an X3D model. This model provides a visual and behavioral representation of the entity for usage in X3D simulations. The mappings are done by selecting the most specific record that fits the entity. A value of 0 is considered a wildcard. All fields after the first zero shall be zero as well.

The fields are checked in the following order: kind, domain, country, category, subcategory, specific, extra.

EXAMPLE Given an entity whose entity type record was: kind=1, domain=2, country=3, category=4, subcategory=5, specific=6, extra=7. If the mapping field of the DISEntityManager contained these nodes:

```
DISEntityTypeMapping {
  domain 1
  kind 2
  country 3
  url ["model-a.x3d"]
}
DISEntityTypeMapping {
  domain 1
  kind 2
  country 3
  category 4
  url ["model-b.x3d"]
}
```

Then, an entity using the second node with a url of "model-b.x3d" is used as its the most specific mapping.

## 28.3.3 EspduTransform

```
EspduTransform : X3DGroupingNode, X3DSensorNode {
  MFNode [in] addChildren
  MFNode [in] removeChildren
  SFFloat [in] set_articulationParameterValue0 (-∞,∞)
  SFFloat [in] set_articulationParameterValue1 (-∞,∞)
  SFFloat [in] set_articulationParameterValue2 (-∞,∞)
  SFFloat [in] set_articulationParameterValue3 (-∞,∞)
  SFFloat [in] set_articulationParameterValue4 (-∞,∞)
  SFFloat [in] set_articulationParameterValue5 (-∞,∞)
  SFFloat [in] set_articulationParameterValue6 (-∞,∞)
  SFFloat [in] set_articulationParameterValue7 (-∞,∞)
  SFString [in,out] address "localhost"
  SFInt32 [in,out] applicationID 1 [0,65535]
  SFInt32 [in,out] articulationParameterCount 0 [0,78]
  MFInt32 [in,out] articulationParameterDesignatorArray [] [0,255]
```

MFInt32	[in,out]	articulationParameterChangeIndicatorArray	[]	[0,255]
MFInt32	[in,out]	articulationParameterIdPartAttachedToArray	[]	[0,65535]
MFInt32	[in,out]	articulationParameterTypeArray	[]	
MFFloat	[in,out]	articulationParameterArray	[]	(-∞,∞)
SFVec3f	[in,out]	center	0 0 0	(-∞,∞)
MFNode	[in,out]	children	[]	
SFInt32	[in,out]	collisionType	0	[0,255]
SFInt32	[in,out]	deadReckoning	0	[0,255]
SFVec3f	[in,out]	detonationLocation	0 0 0	(-∞,∞)
SFVec3f	[in,out]	detonationRelativeLocation	0 0 0	(-∞,∞)
SFInt32	[in,out]	detonationResult	0	[0,255]
SFBool	[in,out]	bboxDisplay	FALSE	
SFBool	[in,out]	enabled	TRUE	
SFInt32	[in,out]	entityCategory	0	[0,255]
SFInt32	[in,out]	entityCountry	0	[0,65535]
SFInt32	[in,out]	entityDomain	0	[0,255]
SFInt32	[in,out]	entityExtra	0	[0,255]
SFInt32	[in,out]	entityID	0	[0,65535]
SFInt32	[in,out]	entityKind	0	[0,255]
SFInt32	[in,out]	entitySpecific	0	[0,255]
SFInt32	[in,out]	entitySubCategory	0	[0,255]
SFInt32	[in,out]	eventApplicationID	1	[0,65535]
SFInt32	[in,out]	eventEntityID	0	[0,65535]
SFInt32	[in,out]	eventNumber	0	[0,65535]
SFInt32	[in,out]	eventSiteID	0	[0,65535]
SFBool	[in,out]	fired1	FALSE	
SFBool	[in,out]	fired2	FALSE	
SFInt32	[in,out]	fireMissionIndex	0	[0,65535]
SFFloat	[in,out]	firingRange	0.0	(0,∞)
SFInt32	[in,out]	firingRate	0	[0,65535]
SFInt32	[in,out]	forceID	0	[0,255]
SFInt32	[in,out]	fuse	0	[0,65535]
SFVec3d	[in,out]	geoCoords	0 0 0	(-∞,∞)
SFVec3f	[in,out]	linearVelocity	0 0 0	(-∞,∞)
SFVec3f	[in,out]	linearAcceleration	0 0 0	(-∞,∞)
SFString	[in,out]	marking	""	
SFNode	[in,out]	metadata	NULL	[X3DMetadataObject]
SFString	[in,out]	multicastRelayHost	""	
SFInt32	[in,out]	multicastRelayPort	0	
SFInt32	[in,out]	munitionApplicationID	1	[0,65535]
SFVec3f	[in,out]	munitionEndPoint	0 0 0	(-∞,∞)
SFInt32	[in,out]	munitionEntityID	0	[0,65535]
SFInt32	[in,out]	munitionQuantity	0	[0,65535]
SFInt32	[in,out]	munitionSiteID	0	[0,65535]
SFVec3f	[in,out]	munitionStartPoint	0 0 0	(-∞,∞)
SFString	[in,out]	networkMode	"standAlone" ["standAlone"]	
			"networkReader"	
			"networkWriter"	
SFInt32	[in,out]	port	0	[0,65535]
SFTime	[in,out]	readInterval	0.1	[0,∞)
SFRotation	[in,out]	rotation	0 0 1 0	(-∞,∞) [-1,1]
SFVec3f	[in,out]	scale	1 1 1	(-∞,∞)
SFRotation	[in,out]	scaleOrientation	0 0 1 0	(-∞,∞) [-1,1]
SFInt32	[in,out]	siteID	0	[0,65535]
SFVec3f	[in,out]	translation	0 0 0	(-∞,∞)
SFBool	[in,out]	visible	TRUE	
SFInt32	[in,out]	warhead	0	[0,65535]
SFTime	[in,out]	writeInterval	1.0	[0,∞)
SFFloat	[out]	articulationParameterValue0_changed	0.0	(-∞,∞)
SFFloat	[out]	articulationParameterValue1_changed	0.0	(-∞,∞)
SFFloat	[out]	articulationParameterValue2_changed	0.0	(-∞,∞)
SFFloat	[out]	articulationParameterValue3_changed	0.0	(-∞,∞)
SFFloat	[out]	articulationParameterValue4_changed	0.0	(-∞,∞)
SFFloat	[out]	articulationParameterValue5_changed	0.0	(-∞,∞)
SFFloat	[out]	articulationParameterValue6_changed	0.0	(-∞,∞)
SFFloat	[out]	articulationParameterValue7_changed	0.0	(-∞,∞)
SFTime	[out]	collideTime	0	[0,∞)
SFTime	[out]	detonateTime	0	[0,∞)
SFTime	[out]	firedTime	0	[0,∞)
SFBool	[out]	isActive	FALSE	
SFBool	[out]	isCollided	FALSE	
SFBool	[out]	isDetonated	FALSE	
SFBool	[out]	isNetworkReader	FALSE	
SFBool	[out]	isNetworkWriter	FALSE	
SFBool	[out]	isRtpHeaderHeard	FALSE	
SFBool	[out]	isStandAlone	FALSE	
SFTime	[out]	timestamp	0	[0,∞)
SFVec3f	[]	bboxCenter	0 0 0	(-∞,∞)
SFVec3f	[]	bboxSize	-1 -1 -1	[0,∞) or -1 -1 -1
MFString	[]	geoSystem	["GD","WE"]	[see 25.2.3]
SFBool	[]	rtpHeaderExpected	FALSE	

EspduTransform is a [X3DGroupingNode](#) that can contain most nodes, and also implements the X3DBoundedObject interface. EspduTransform integrates functionality of the following DIS PDUs: EntityStatePDU, CollisionPDU, DetonationPDU, FirePDU, CreateEntity, and RemoveEntity. The following description identifies the fields of the EspduTransform node that are associated with the content of these PDUs.

As an *X3DGroupingNode*, EspduTransform has *addChildren* and *removeChildren* events

to permit modification to the subordinate structure of the scene graph. The *removeChildren* event removes nodes from the *EspduTransform*'s *children* field. Any nodes in the *removeChildren* event that are not in the *EspduTransform*'s *children* list are ignored. Adding a node to the *children* field will add that node to the *EspduTransform*'s set of children. Adding any node to the *EspduTransform*'s *children* field that is already in that child list is illegal. Adding any node to the *EspduTransform*'s *children* that is an ancestor of that grouping is illegal.

Fields in the *EspduTransform* node that were not previously described in [28.2.3 Common DIS fields](#) are: *translation*, *rotation*, *center*, *scale*, *scaleOrientation*, *bboxCenter*, *bboxSize*, *articulationParameterCount*, *articulationParameterDesignatorArray*, *articulationParameterChangeIndicatorArray*, *articulationParameterIdPartAttachedToArray*, *articulationParameterTypeArray*, *articulationParameterArray*, *set\_articulationParameterValue0*, *set\_articulationParameterValue1*, *set\_articulationParameterValue2*, *set\_articulationParameterValue3*, *set\_articulationParameterValue4*, *set\_articulationParameterValue5*, *set\_articulationParameterValue6*, *set\_articulationParameterValue7*, *articulationParameterValue0\_changed*, *articulationParameterValue1\_changed*, *articulationParameterValue2\_changed*, *articulationParameterValue3\_changed*, *articulationParameterValue4\_changed*, *articulationParameterValue5\_changed*, *articulationParameterValue6\_changed*, *articulationParameterValue7\_changed*, *marking*, *forceID*, *entityKind*, *entityDomain*, *entityCountry*, *entityCategory*, *entitySubCategory*, *entitySpecific*, *entityExtra*, *linearVelocity*, *linearAcceleration*, *deadReckoning*, *isCollided*, *collidedTime*, *eventApplicationID*, *eventSiteID*, *eventEntityID*, *collisionType*, *eventNumber*, *fired1*, *fired2*, *firedTime*, *munitionStartPoint*, *munitionEndPoint*, *fireMissionIndex*, *munitionApplicationID*, *munitionSiteID*, *munitionEntityID*, *warhead*, *fuse*, *munitionQuantity*, *firingRate*, *firingRange*, *isDetonated*, *detonateTime*, *detonationLocation*, *detonationRelativeLocation*, and *detonationResult*.

The Entity State PDU provides notification of a new position and orientation of an entity, which directly corresponds to the functionality of the X3D [Transform](#) node. The *translation* field corresponds to the new position in the DIS coordinate system. It is important to distinguish between the X3D coordinate system and the DIS coordinate system. If (x, y, z) are the coordinates of a point in the X3D coordinate system, corresponding DIS coordinates for the same point would be (x, -z, y). Note that only X3D coordinates are used by the X3D scene. The *EspduTransform* node internally performs all conversions to/from DIS coordinates when writing/reading DIS PDUs to/from the network.

The *rotation* field provides the rotation of the entity, where the rotation is performed relative to the value of the *center* field. The *scale* field provides scaling factors along the x, y, z axes, while the *scaleOrientation* field provides scaling factors for the specified *rotation*. The *translation*, *rotation*, *scale*, and *center* fields corresponds directly with functionality of the Transform node. The *bboxCenter* and *bboxSize* fields (of the [X3DBoundedObject](#) interface) specify the center and size, respectively, of a cube bounding the entity geometry contained in the *EspduTransform* grouping node, corresponding to the same fields of an X3D Transform node.

Articulation parameter *inputOnly* events and *outputOnly* events are provided for the *articulationParameters* array in order to enable simple routing of primary events of

interest into (and out of) the array. As an example, if eight articulation parameters were needed for an `EspduTransform` controlling the movable parts of a race-car model, each of these articulation parameters might be individually routed as necessary. Events into (and out of) articulationParameter subscripts [8] through [78] are accomplished either by a separate Script node mechanism, or else by complete routing/replacement using an `MFFloat` event.

The *articulationParameterCount* field (8-bit unsigned integer) indicates the number of parameters that are being used to describe articulation of various segments of the entity model. For example, the orientation of a turret together with the inclination of the gun for a tank entity may be described by two articulation parameters, or the orientation of various segments in a humanoid model may be provided by several articulation parameters. The maximum number of articulated parameter records in an Entity State PDU is constrained to 78 by the maximum length of a PDU.

For X3D authoring convenience in ROUTEing events to (or from) articulation parameters, the first eight articulation parameter values may be accessed by `accessType` `inputOnly/outputOnly` fields (*set\_articulationParameterValue0*, ..., *set\_articulationParameterValue7* and *articulationParameterValue0\_changed*, ..., *articulationParameterValue7\_changed*).

Fields *articulationParameterDesignatorArray*, *articulationParameterChangeIndicatorArray*, *articulationParameterIdPartAttachedToArray*, *articulationParameterTypeArray* are arrays that correspond to additional values provided in each articulation parameter record. Elements in these arrays correspond to each articulation parameter in sequential order.

- The Parameter Type Designator entries in the *articulationParameterDesignatorArray* indicate if the the parameter record is for an articulated or attached part. It is represented by an 8-bit enumeration.
- The Change Indicator entries in the *articulationChangeIndicatorArray* indicate the change of any parameter for the associated articulated part. This is specified by an 8-bit unsigned integer. The value is initially set to zero for each exercise and is sequentially incremented by one for each change in the articulation parameters. The proper indicator is updated automatically by an X3D DIS implementation upon receipt of a *set\_articulationParameterValue* event.
- The ID - Part Attached To entries in the *articulationParameterIdPartAttachedToArray* identify the articulated part to which this articulation parameter is attached. The value is specified by a 16-bit unsigned integer, and is set to zero if the articulated part is attached directly to the entity.
- The Parameter Type entries in the *articulationParameterTypeArray* are specified by 32-bit enumeration values.
- The Parameter Value entries in the *articulationParameterArray* are specified by a 64-bit field. The definition of the 64 bits is determined based on the type of parameter indicated above.

The *marking* field is a `SFString` value (with a maximum of 11 characters) corresponding to a selection from an enumerated set of markings in the DIS standard (for full compliance) or an arbitrary string for non-compliant applications using the `EspduTransform` node.



The *forceID* and *entityKind* fields are 8-bit identification enumerations. The *entityDomain* field (8-bit enumeration) identifies the domain of operation of the entity (e.g., subsurface, surface, land), except for munition entities. For munition entities, this field specifies the domain of the target. The *entityCountry* field (16-bit enumeration) specifies the country to which the design of the entity is attributed. The *entityCategory* field (8-bit enumeration) identifies the main category that describes the entity. The *entitySubCategory* field (8-bit enumeration) specifies a subcategory based on the identified category value. The *entitySpecific* field (8-bit enumeration) provides specific information about the entity based on the identified subcategory field. The *entityExtra* field (8-bit enumeration) provides additional information about the entity. The DIS specification also allows identification of an Alternative Entity Type containing the same fields (Entity Kind, Domain, Country, Category, Subcategory, Specific, Extra) as described above.

Enumeration values are provided directly, or in additional references, as specified by IEEE 1278 (see [2.\[IEEE1278\]](#)).

The *linearVelocity* and *linearAcceleration* fields provide the linear velocity and acceleration vectors, respectively, for dead reckoning calculations. The dead reckoning algorithm to be applied is identified in the *deadReckoning* field (8-bit enumeration).

The CollisionPDU is sent to notify an entity that a collision has occurred. The issuing entity is identified in the *entityID* field described in [28.2.3 Common DIS fields](#). The *isCollided* field is a Boolean value indicating if a collision (`TRUE`) has occurred. The *collideTime* field gives the time (SFTIME) at which the collision was determined to have occurred. In a CollisionPDU message, the *eventSiteID*, *eventApplicationID*, *eventEntityID* triplet uniquely identifies the entity colliding with the issuing entity (when known). The *collisionType* field (8-bit enumeration) identifies the type of collision that occurred.

The *eventNumber* field is set to one for each exercise and incremented by one for each fire event, collision event, or electromagnetic mission event.

The FirePDU notifies the simulation that an entity has fired a weapon. The firing entity is identified in the *entityID* field described in [28.2.3 Common DIS fields](#). Field *fired1* (set to `TRUE`) indicates the primary weapon was fired; field *fired2* (set to `TRUE`) indicates the entity's secondary weapon was fired. The *firedTime* field gives the time (SFTIME) at which the firing occurred. Fields *munitionStartPoint* and *munitionEndPoint* describe the path of the munition from firing weapon to detonation or impact. The *fireMissionIndex* field identifies the fire mission, if known. The *firingRange* field specifies the range (in meters) that an entity's fire control system has assumed in computing the fire control solution.

In a FirePDU message, the *EventSiteID*, *EventApplicationID*, *EventEntityID* triplet uniquely identifies the target entity, when known. For the FirePDU and DetonationPDU messages, the *munitionSiteID*, *munitionApplicationID*, *munitionEntityID* triplet uniquely identifies the munition entity (if known).

The FirePDU and DetonationPDU messages provide burst descriptor information in the *warhead* (16-bit enumeration), *fuse* (16-bit enumeration), *munitionQuantity* (16-bit unsigned integer), and *firingRate* (16-bit unsigned integer) fields.

The DetonationPDU provides notification that a munition has detonated or impacted so that other entities can determine possible damage from the detonation. The *detonated* field indicates if detonation has occurred (`TRUE`). The *detonateTime* field gives the time (SFTime) at which the detonation is determined to have occurred. This enables other entities to determine their position relative to the detonation at the time the detonation occurred.

The DetonationPDU provides the *detonationLocation* in world coordinates, as well as the *detonationRelativeLocation*, the location of the detonation or impact in the target entity's coordinate system. The *detonationResult* field (8-bit enumeration) provides information on the outcome of the detonation event.

The CreateEntityPDU notifies other entities of a new entity in the simulation. The *siteID*, *applicationID*, *entityID* triplet described in [28.2.3 Common DIS fields](#), uniquely identifies the new entity. A CreateEntityPdu is sent upon startup or creation of a new entity.

The RemoveEntityPDU notifies other entities of the removal of an entity from the simulation. The *siteID*, *applicationID*, *entityID* triplet described in [28.2.3 Common DIS fields](#), uniquely identifies the entity to be removed. A RemoveEntityPDU is sent upon shutdown or removal of an existing entity.

### 28.3.4 ReceiverPdu

```
ReceiverPdu : X3DNetworkSensorNode, X3DBoundedObject {
  SFString [in,out] address          "localhost"
  SFInt32  [in,out] applicationID    1          [0,65535]
  SFBool   [in,out] bboxDisplay     FALSE
  SFBool   [in,out] enabled         TRUE
  SFInt32  [in,out] entityID        0          [0,65535]
  SFVec3d  [in,out] geoCoords       0 0 0      (-∞,∞)
  SFNode   [in,out] metadata        NULL      [X3DMetadataObject]
  SFString [in,out] multicastRelayHost ""
  SFInt32  [in,out] multicastRelayPort 0
  SFString [in,out] networkMode     "standAlone" ["standAlone"|
                                     "networkReader"|
                                     "networkWriter"]
  SFInt32  [in,out] port            0          [0,65535]
  SFInt32  [in,out] radioID         0          [0,65535]
  SFFloat  [in,out] readInterval    0.1      [0,∞)
  SFFloat  [in,out] receivedPower   0.0       [0,∞)
  SFInt32  [in,out] receiverState   0          [0,65535]
  SFBool   [in,out] rtpHeaderExpected FALSE
  SFInt32  [in,out] siteID          0          [0,65535]
  SFInt32  [in,out] transmitterApplicationID 1 [0,65535]
  SFInt32  [in,out] transmitterEntityID 0 [0,65535]
  SFInt32  [in,out] transmitterRadioID 0 [0,65535]
  SFInt32  [in,out] transmitterSiteID 0 [0,65535]
  SFBool   [in,out] visible         TRUE
  SFInt32  [in,out] whichGeometry   1          [-1,∞)
  SFFloat  [in,out] writeInterval   1.0      [0,∞)
  SFBool   [out]  isActive         FALSE
  SFBool   [out]  isNetworkReader  FALSE
  SFBool   [out]  isNetworkWriter  FALSE
  SFBool   [out]  isRtpHeaderHeard FALSE
  SFBool   [out]  isStandAlone     FALSE
  SFTime   [out]  timestamp        0
  SFVec3f  []    bboxCenter        0 0 0      (-∞,∞)
  SFVec3f  []    bboxSize          -1 -1 -1 [0,∞) [0,∞) [0,∞) or -1 -1 -1
  MFString []    geoSystem         ["GD", "WE"] [see 25.2.3]
}
```

ReceiverPdu is an [X3DChildNode](#) node, and also implements the [X3DBoundedObject](#) interface. The ReceiverPdu transmits the state of radio frequency (RF) receivers modeled in the simulation. Fields in the ReceiverPdu node that were not previously described in Common DIS Fields are: *whichGeometry*, *radioID*, *receivedPower*, *receiverState*, *transmitterSiteID*, *transmitterApplicationID*, *transmitterEntityID*, and *transmitterRadioID*.

The *radioID* field (16-bit unsigned integer) identifies a particular radio within a given entity (*entityID*). The *radioID* is assigned sequentially, starting with 1. The combination of Entity ID and Radio ID uniquely identify a radio within a simulation exercise. The *receivedPower* field (32-bit floating point) indicates the RF power received, after applying any propagation loss and antenna gain. The field value is in units of decibel-milliwatts (dBm). The *receiverState* (16-bit enumeration) indicates if the receiver is currently idle or busy via one of the following enumerated values:

- 0 = off,
- 1 = on but not receiving, or
- 2 = on and receiving.

The *transmitterEntityID* (16-bit unsigned integer) identifies the transmitter entity that has emitted the signal being received. The *transmitterRadioID* field (16-bit unsigned integer) identifies the particular radio within the transmitter entity (*transmitterEntityID*).

The *transmitterSiteID* field (16-bit unsigned integer) provides the unique DIS site identifier for the transmitter entity. The *transmitterApplicationID* (16-bit unsigned integer) provides the application identifier for the transmitter entity that is unique within the DIS site (*transmitterSiteID*).

The *whichGeometry* field indicates to the rendering software what geometry to draw for the receiverPdu node: -1 for no geometry; 0 for text trace; 1 for default geometry for this node. Additional alternative geometry modes may optionally be supported by browsers. Lack of support for higher modes reverts to *whichGeometry* value of 1.

The *bboxCenter* and *bboxSize* fields (of the *X3DBoundedObject* interface) specify the center and size, respectively, of a cube bounding the display geometry (if any) for this node.

## 28.3.5 SignalPdu

```
SignalPdu : X3DNetworkSensorNode, X3DBoundedObject {
  SFString [in,out] address "localhost"
  SFInt32 [in,out] applicationID 1 [0,65535]
  MFInt32 [in,out] data [] [0,255]
  SFBool [in,out] bboxDisplay FALSE
  SFInt32 [in,out] dataLength 0 [0,65535]
  SFBool [in,out] enabled TRUE
  SFInt32 [in,out] encodingScheme 0 [0,65535]
  SFInt32 [in,out] entityID 0 [0,65535]
  SFVec3d [in,out] geoCoords 0 0 0 (-∞,∞)
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFString [in,out] multicastRelayHost ""
  SFInt32 [in,out] multicastRelayPort 0
  SFString [in,out] networkMode "standAlone" ["standAlone"
    "networkReader"
    "networkWriter"]
  SFInt32 [in,out] port 0 [0,65535]
  SFInt32 [in,out] radioID 0 [0,65535]
  SFFloat [in,out] readInterval 0.1 [0,∞)
  SFBool [in,out] rtpHeaderExpected FALSE
  SFInt32 [in,out] sampleRate 0 [0,65535]
  SFInt32 [in,out] samples 0 [0,65535]
  SFInt32 [in,out] siteID 0 [0,65535]
  SFInt32 [in,out] tdlType 0 [0,65535]
  SFBool [in,out] visible TRUE
  SFInt32 [in,out] whichGeometry 1 [-1,∞)
  SFFloat [in,out] writeInterval 1.0 [0,∞)
  SFBool [out] isActive
  SFBool [out] isNetworkReader
  SFBool [out] isNetworkWriter
  SFBool [out] isRtpHeaderHeard
  SFBool [out] isStandAlone
  SFTime [out] timestamp
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
```



```
MFString []   geoSystem   ["GD","WE"] [see 25.2.3]
}
```

SignalPdu is an [X3DChildNode](#) node, and also implements the [X3DBoundedObject](#) interface. Transmission of voice, audio or other data is communicated by issuing a Signal PDU from the SignalPdu node. Fields in the SignalPdu node that were not previously described in Common DIS Fields are: *radioID*, *encodingScheme*, *tdlType*, *sampleRate*, *dataLength*, *samples*, *data*, and *whichGeometry*.

The *radioID* field identifies a particular radio within a given entity (*entityID*). The *radioID* (16-bit unsigned integer) is assigned sequentially, starting with 1. The combination of Entity ID and Radio ID uniquely identify a radio within a simulation exercise. The *encodingScheme* field (16-bit enumeration) designates both an Encoding Class enumerated value (2 most significant bits):

- 0 = Encoded Voice;
- 1 = Raw Binary Data;
- 2 = Application-Specific Data;
- 3 = Database Index.

and an Encoding Type enumerated value (14 least significant bits):

- 1 = 8-bit mu-law;
- 2 = CVSD per MIL-STD-188-113;
- 3 = ADPCM per CCITT G.721;
- 4 = 16-bit linear PCM;
- 5 = 8-bit linear PCM;
- 6 = Vector Quantization.

The *tdlType* field (16-bit enumeration) specifies the Tactical Data Link (TDL) type as an enumerated value when the Encoding Class is *voice*, *raw binary*, *application-specific*, or *database index* representation of a TDL message. The field is set to zero when it is not representing a TDL message.

The *sampleRate* field (32-bit unsigned integer) gives either (1) the sample rate in samples per second if the Encoding Class is *encoded audio* or (2) the data rate in bits per second for data transmissions. If the Encoding Class is *database index*, *sampleRate* is set to zero.

The *samples* field (16-bit unsigned integer) gives the number of samples in the PDU if the Encoding Class is *encoded voice*; otherwise, the field is set to zero.

The *dataLength* field (16-bit unsigned integer) specifies the number of bits of digital voice audio or digital data being sent in the Signal PDU. If the Encoding Class is *database index*, the *dataLength* field is set to the value 96.

The *data* field specifies the audio or digital data conveyed by the radio transmission. The interpretation of the field depends on the value of the *encodingScheme* and *tdlType* fields. Refer to IEEE 1278 ([2.\[IEEE1278\]](#)) for details.

The *whichGeometry* field indicates to the rendering software what geometry to draw for the SignalPdu node: -1 for no geometry; 0 for text trace; 1 for default geometry for this node. Additional alternative geometry modes may optionally be supported by

browsers. Lack of support for higher modes reverts to *whichGeometry* value of 1.

The *bboxCenter* and *bboxSize* fields (of the *X3DBoundedObject* interface) specify the center and size, respectively, of a cube bounding the display geometry (if any) for this node.

## 28.3.6 TransmitterPdu

```

TransmitterPdu : X3DNetworkSensorNode, X3DBoundedObject {
  SFString [in,out] address "localhost"
  SFVec3f [in,out] antennaLocation 0 0 0 (-∞,∞)
  SFInt32 [in,out] antennaPatternLength 0 [0,65535]
  SFInt32 [in,out] antennaPatternType 0 [0,65535]
  SFInt32 [in,out] applicationID 1 [0,65535]
  SFInt32 [in,out] cryptoKeyID 0 [0,65535]
  SFInt32 [in,out] cryptoSystem 0 [0,65535]
  SFBool [in,out] bboxDisplay FALSE
  SFBool [in,out] enabled TRUE
  SFInt32 [in,out] entityID 0 [0,65535]
  SFInt32 [in,out] frequency 0
  SFVec3d [in,out] geoCoords 0 0 0 (-∞,∞)
  SFInt32 [in,out] inputSource 0 [0,255]
  SFInt32 [in,out] lengthOfModulationParameters 0 [0,255]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFInt32 [in,out] modulationTypeDetail 0 [0,65535]
  SFInt32 [in,out] modulationTypeMajor 0 [0,65535]
  SFInt32 [in,out] modulationTypeSpreadSpectrum 0 [0,65535]
  SFInt32 [in,out] modulationTypeSystem 0 [0,65535]
  SFString [in,out] multicastRelayHost ""
  SFInt32 [in,out] multicastRelayPort 0
  SFString [in,out] networkMode "standAlone" ["standAlone"]
  SFInt32 [in,out] port 0 [0,65535]
  SFFloat [in,out] power 0.0 [0,∞)
  SFInt32 [in,out] radioEntityTypeCategory 0 [0,255]
  SFInt32 [in,out] radioEntityTypeCountry 0 [0,65535]
  SFInt32 [in,out] radioEntityTypeDomain 0 [0,255]
  SFInt32 [in,out] radioEntityTypeKind 0 [0,255]
  SFInt32 [in,out] radioEntityTypeNomenclature 0 [0,255]
  SFInt32 [in,out] radioEntityTypeNomenclatureVersion 0 [0,65535]
  SFInt32 [in,out] radioID 0 [0,255]
  SFFloat [in,out] readInterval 0.1 [0,∞)
  SFVec3f [in,out] relativeAntennaLocation 0 0 0 (-∞,∞)
  SFBool [in,out] rtpHeaderExpected FALSE
  SFInt32 [in,out] siteID 0 [0,65535]
  SFFloat [in,out] transmitFrequencyBandwidth 0.0 (-∞,∞)
  SFInt32 [in,out] transmitState 0 [0,255]
  SFBool [in,out] visible TRUE
  SFInt32 [in,out] whichGeometry 1 [-1,∞)
  SFFloat [in,out] writeInterval 1.0 [0,∞)
  SFBool [out] isActive FALSE
  SFBool [out] isNetworkReader FALSE
  SFBool [out] isNetworkWriter FALSE
  SFBool [out] isRtpHeaderHeard FALSE
  SFBool [out] isStandAlone FALSE
  SFTime [out] timestamp 0
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) [0,∞) [0,∞) or -1 -1 -1
  MFString [] geoSystem ["GD","WE"] [see 25.2.3]
}

```

TransmitterPdu is an X3DChildNode node, and also implements the X3DBoundedObject interface. The TransmitterPdu provides detailed information about a radio transmitter. Fields in the TransmitterPdu node that were not previously described in Common DIS Fields are: *radioID*, *radioEntityTypeKind*, *radioEntityTypeDomain*, *radioEntityTypeCountry*, *radioEntityTypeCategory*, *radioEntityTypeNomenclature*, *radioEntityTypeNomenclatureVersion*, *transmitState*, *inputSource*, *antennaLocation*, *antennaPatternType*, *antennaPatternLength*, *frequency*, *transmitFrequencyBandwidth*, *power*, *modulationTypeSpreadSpectrum*, *modulationTypeMajor*, *modulationTypeDetail*, *modulationTypeSystem*, *lengthOfModulationParameters*, *cryptoSystem*, *cryptoKeyID*, *relativeAntennaLocation*, and *whichGeometry*.

The *radioID* field identifies a particular radio within a given entity (*entityID*). The *radioID* (16-bit unsigned integer) is assigned sequentially, starting with 1. The combination of Entity ID and Radio ID uniquely identify a radio within a simulation

exercise.

The radio entity type is described by a combination of fields: *radioEntityTypeKind*, *radioEntityTypeDomain*, *radioEntityTypeCountry*, *radioEntityTypeCategory*, *radioEntityTypeNomenclatureVersion*, and *radioEntityTypeNomenclature*. The *radioEntityTypeKind* is an 8-bit enumeration (e.g., value of 7 indicates Entity Kind of "Radio"). The *radioEntityTypeDomain* field (8-bit enumeration) designates the domain of operation of the radio enumerated value:

- 0 = other;
- 1 = land;
- 2 = air;
- 3 = surface;
- 4 = subsurface;
- 5 = space.

The *radioEntityTypeCountry* field (16-bit enumeration) identifies the country to which the design of the radio entity is attributed (see [2.\[IEEE1278\]](#)).

The *radioEntityTypeCategory* field (8-bit enumeration) specifies the main category describing the radio entity. The *radioEntityTypeNomenclature* (16-bit enumeration) specifies the nomenclature for a particular communications device. Nomenclatures are a combination of letters and/or numbers arranged in a specific sequence to provide a short method of identification. The *radioEntityTypeNomenclatureVersion* field designates the specific modification or individual unit type of a series and/or family of equipment.

The *transmitterState* field (8-bit enumeration) indicates the operational state of the transmitter entity enumerated value:

- 0 = off,
- 1 = on but not transmitting, or
- 2 = on and transmitting.

The *inputSource* (8-bit enumeration) specifies which position or data port in the entity utilizing the radio is providing the input audio or data being transmitted enumerated value:

- 0 = other,
- 1 = pilot,
- 2 = copilot,
- 3 = first officer,
- 4 = driver,
- 5 = loader,
- 6 = gunner,
- 7 = commander,
- 8 = digital data device, or
- 9 = intercom.

The *antennaLocation* field provides the location of the transmitter antenna in the DIS coordinate system.

NOTE IEEE 1278 (see [2.\[IEEE1278\]](#)) allocates 64-bit floating point values for the components of the antenna

location vector, whereas it is represented here as `SFVec3f` for maximum interoperability with X3D.

The *relativeAntennaLocation* field provides an offset from the location of the transmitter entity to simulate placement of antennas some distance from the transmitter equipment.

The *antennaPatternType* field (16-bit enumeration) indicates the type of representation for the radiation pattern from the antenna enumerated value:

- 0 = omnidirectional;
- 1 = beam;
- 2 = spherical harmonic.

The value of this field determines the interpretation of the Antenna Pattern Parameter field of the DIS Transmitter PDU. The *antennaPatternLength* field (16-bit unsigned integer) specifies the length of the Antenna Pattern Parameters field in octets (value is a multiple of 8).

The *frequency* field specifies the center frequency (in Hertz) being used by the radio for transmission. Note that IEEE 1278 allocates a 64-bit unsigned integer to represent frequency values, whereas it is limited to `SFInt32` in X3D. The *transmitFrequencyBandwidth* (32-bit floating point) identifies the bandpass of the transmitting radio entity. The *power* field (32-bit floating point) provides the average power (in units of dBm) of the radio entity transmission.

Information about the type of modulation used for radio transmission is represented in the Transmitter PDU by several fields. These fields identify the signal parameters that are used to determine whether two radios may interoperate. The *modulationTypeSpreadSpectrum* field indicates the spread spectrum technique or combination of techniques in use enumerated value:

- 0 = frequency hopping;
- 1 = pseudo-noise;
- 2 = time hopping;
- 3-15 are to be determined.

The *modulationTypeMajor* (16-bit enumeration) provides the major classification of the modulation type enumerated value:

- 0 = other;
- 1 = amplitude;
- 2 = amplitude and angle;
- 3 = angle;
- 4 = combination;
- 5 = pulse;
- 6 = unmodulated.

The *modulationTypeDetail* field (16-bit enumerations) contains detailed information depending on the Major Modulation Type (*modulationTypeMajor*). The *modulationTypeSystem* field (16-bit enumeration) specifies the interpretation of the modulation parameter field(s) in the Transmitter PDU (enumerated value):

- 0 = other;

- 1 = generic;
- 2 = HQ;
- 3 = HQII;
- 4 = HQIIA;
- 5 = SINCGARS;
- 6 = CCTT SINCGARS.

The *cryptoSystem* field (16-bit enumeration) identifies the crypto equipment used enumerated value:

- 0 = other;
- 1 = KY-28;
- 2 = KY-58;
- 3 = Narrow Spectrum Secure Voice (NSVE);
- 4 = Wide Spectrum Secure Voice (WSVE);
- 5 = SINCGARS ICOM.

The *cryptoKeyID* field (16-bit unsigned integer) indicates whether the crypto equipment is in the baseband encryption mode or the diphase encryption mode (high order bit of the 16-bit field) and provides the key identifier (lower order 15 bits). If the key identifiers of the transmitter and receiver match, they are considered to be using the same encryption key (note that this is not an actual crypto key).

The *whichGeometry* field indicates to the rendering software what geometry to draw for the TransmitterPdu node: -1 for no geometry; 0 for text trace; 1 for default geometry for this node. Additional alternative geometry modes may optionally be supported by browsers. Lack of support for higher modes reverts to *whichGeometry* value of 1.

The *bboxCenter* and *bboxSize* fields (of the X3DBoundedObject interface) specify the center and size, respectively, of a cube bounding the display geometry (if any) for this node.

## 28.4 Support levels

The DIS component provides two levels of support as specified in [Table 28.2](#).

**Table 28.2 — DIS component support levels**

Level	Prerequisites	Nodes/Features	Support
1	Core 1 Time 1 Grouping 3 Networking 3 Rendering 1 Shape 1 Geometry3D 1 Interpolator 1 Point device sensor 1 Navigation 1		

		EspduTransform	All fields fully supported.
		ReceiverPDU	All fields fully supported.
		SignalPDU	All fields fully supported.
		TransmitterPDU	All fields fully supported.
<b>2</b>	Core 1 Time 1 Grouping 3 Networking 3 Rendering 1 Shape 1 Geometry3D 1 Interpolator 1 Point device sensor 1 Navigation 1		
		All Level 1 DIS nodes.	All fields fully supported.
		DISEntityManager	All fields fully supported.
		DISEntityTypeMapping	All fields fully supported.





## Extensible 3D (X3D) Part 1: Architecture and base components

# Annex G Recommended navigation behaviours (informative)



## ● G.1 Introduction and table of contents

This annex describes basic X3D scene navigation recommended practice. This recommended practice describes a browser-independent standardized keyboard interface which implements X3D frequently used scene interactivity. Features that imply interactivity are fundamental in X3D. The author expects to be able to specify multiple viewpoints in a predictable sequence, the ability to point and select, and to enable continuous navigation within the scene. Likewise the interactor expects to be able to exercise scene functionality using predictable methods.

This recommended practice is intended to allow use of a core subset of the functionality of an X3D browser, not unnecessarily limit interactive functionality which may be provided by a browser.

[Table G.1](#) lists the major topics in this annex.

**Table G.1 — Topics**

- [G.1 Introduction and table of contents](#)
- [G.2 Select from multiple viewpoints](#)
- [G.3 Emulate pointing device](#)
- [G.4 Select or activate pointing device](#)
- [G.5 Disable/enable keyboard](#)
- [Table G.1 Topics](#)

## ● G.2 Select from multiple viewpoints

User navigation in X3D environments includes definition of multiple viewpoints. Where the user is allowed to freely select between viewpoints, typical controls allow simple



selection of:

- Home (Initial) ViewPoint,
- Last (Final) Viewpoint,
- Next Viewpoint in Sequence, and
- Previous Viewpoint in Sequence.

This annex recommends using the following keys:

HOME	Initial Viewpoint
PGDN	Next Viewpoint
PGUP	Previous Viewpoint
END	Final Viewpoint

## G.3 Emulate pointing device

The pointing device is used to control navigation through the scene. Where the user is allowed to interact using the pointing device, typical controls allow up/down/right/left pointing device movement to control movement of the viewpoint.

The objective is not to actually move the screen tracking cursor, but to allow navigation control as if the tracking cursor or pointer is moved under control of the pointing device.

This annex recommends using the following (arrow) keys to emulate relative tracking pointer movement as follows:

UP	Up
DOWN	Down
LEFT	Left
RIGHT	Right

Movement left/right/up/down refers to motion of the user's view while navigating.

Activation of these keys causes movement of the viewpoint according to currently selected navigation type:

WALK:	forward/backward/left/right
FLY:	forward/backward/left/right
EXAMINE:	orbit up/down/left/right around center of rotation with camera pointed at center of rotation

## G.4 Select or activate pointing device

The pointing device is used to provide a means of selecting of a scene element. Where the user is allowed to use this, the following action is recommended: activate pointing device (left mouse click).

This annex recommends using the following key:

ENTER	Left Mouse Click
-------	------------------

## G.5 Disable/enable keyboard

It is recommended that the browser provide a means for the author to enable and disable the keyboard.







## Extensible 3D (X3D) Part 1: Architecture and base components

### 8 Time component

#### 8.1 Introduction

##### 8.1.1 Name

The name of this component is "Time". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 8.1.2 Overview

This clause describes the Time component of this part of ISO/IEC 19775. This includes a definition of the TimeSensor node, the fundamental means for connecting the X3D world to the time base of the browser. [Table 8.1](#) links to the major topics in this clause.

**Table 8.1 — Topics**

- [8.1 Introduction](#)
  - [8.1.1 Name](#)
  - [8.1.2 Overview](#)
- [8.2 Concepts](#)
  - [8.2.1 Time model](#)
  - [8.2.2 Time origin](#)
  - [8.2.3 Discrete and continuous changes](#)
  - [8.2.4 Time-dependent nodes](#)
    - [8.2.4.1 Overview](#)
    - [8.2.4.2 Time cycles](#)
    - [8.2.4.3 Time activation](#)
    - [8.2.4.4 Pausing time](#)
- [8.3 Abstract types](#)
  - [8.3.1 X3DTimeDependentNode](#)
- [8.4 Node reference](#)
  - [8.4.1 TimeSensor](#)
- [8.5 Support levels](#)

- [Figure 8.1 — Examples of time-dependent node execution](#)
- [Table 8.1 — Topics](#)
- [Table 8.2 — Time component support levels](#)

## 8.2 Concepts

### 8.2.1 Time model

The browser controls the passage of time in a world by causing [TimeSensor](#) nodes to generate events as time passes. Specialized browsers or authoring applications may cause time to pass more quickly or slowly than in the real world, but typically the times generated by TimeSensor nodes will approximate "real" time. A world's creator should make no assumptions about how often a TimeSensor will generate events but can safely assume that each time event generated will have a timestamp greater than any previous time event.

### 8.2.2 Time origin

Time (0.0) is equivalent to 00:00:00 GMT January 1, 1970. Absolute times are specified in SFTIME or MFTIME fields as double-precision floating point numbers representing seconds. Negative absolute times are interpreted as happening before 1970.

Processing an event with timestamp  $t$  may only result in generating events with timestamps greater than or equal to  $t$ .

### 8.2.3 Discrete and continuous changes

This International Standard does not distinguish between discrete events (such as those generated by a [TouchSensor](#)) and events that are the result of sampling a conceptually continuous set of changes (such as the fraction events generated by a [TimeSensor](#)). An ideal X3D implementation would generate an infinite number of samples for continuous changes, each of which would be processed infinitely quickly.

Before processing a discrete event, all continuous changes that are occurring at the discrete event's timestamp shall behave as if they generate events at that same timestamp.

Beyond the requirements that continuous changes be up-to-date during the processing of discrete changes, the sampling frequency of continuous changes is implementation dependent. Typically, a TimeSensor affecting a visible (or otherwise perceptible) portion of the world will generate events once per *frame*, where a frame is a single rendering of the world or one time-step in a simulation.

### 8.2.4 Time-dependent nodes

#### 8.2.4.1 Overview

[AudioClip](#), [MovieTexture](#), and [TimeSensor](#) are examples of nodes that are of

[X3DTimeDependentNode](#) type and that activate, pause, resume, and deactivate instantiations of themselves at specified times. Each of these node types contains the inputOutput fields: *startTime*, *pauseTime*, *resumeTime*, *stopTime*, and *loop*, *elapsedTime*, *isActive*, and *isPaused*. The values of the inputOutput fields are used to determine when an instantiated node becomes active or inactive and enters or exits a paused state. Also, under certain conditions, these instantiated nodes ignore events to some of their inputOutput fields. A node ignores an input event by not accepting the new value and not generating an *xxx\_changed* event. An abstract time-dependent node type can be realized as any one of *AudioClip*, *MovieTexture*, or *TimeSensor*.

#### 8.2.4.2 Time cycles

Time-dependent nodes execute in cycles. A cycle is defined by field data within the node. If, at the end of a cycle, the value of *loop* is `FALSE`, execution is terminated (see below for events at termination). Conversely, if *loop* is `TRUE` at the end of a cycle, a time-dependent node continues execution into the next cycle. Thus, a time-dependent node with *loop* `TRUE` at the end of every cycle continues cycling forever if  $startTime \geq stopTime$ , or until *stopTime* if  $startTime < stopTime$ , or until the conditions to pause are set.

The *elapsedTime* outputOnly field delivers the current elapsed time since the *TimeSensor* was activated and running, cumulative in seconds and not counting any time while in a paused state.

#### 8.2.4.3 Time activation

The default values for each of the time-dependent nodes are specified such that any node with default values is already inactive and resumed (and, therefore, will generate no events upon loading). A time-dependent node can be defined such that it will be active upon reading by specifying *loop* `TRUE`. This use of a non-terminating time-dependent node should be used with caution since it incurs continuous overhead on the simulation.

A time-dependent node generates an *isActive* `TRUE` event when it becomes active and generates an *isActive* `FALSE` event when it becomes inactive. These are the only times at which an *isActive* event is generated. In particular, *isActive* events are not sent at each tick of a simulation.

A time-dependent node is inactive until its *startTime* is reached. When time *now* becomes greater than or equal to *startTime*, an *isActive* `TRUE` event is generated and the time-dependent node becomes active (*now* refers to the time at which the browser is simulating and displaying the virtual world). When a time-dependent node is read from a X3D file and the ROUTEs specified within the X3D file have been established, the node should determine if it is active and, if so, generate an *isActive* `TRUE` event and begin generating any other necessary events. However, if a node would have become inactive at any time before the reading of the X3D file, no events are generated upon the completion of the read.

An active time-dependent node will become inactive when *stopTime* is reached if  $stopTime > startTime$ . The value of *stopTime* is ignored if  $stopTime \leq startTime$ . Also, an active time-dependent node will become inactive at the end of the current cycle if

*loop* is `FALSE`. If an active time-dependent node receives a *set\_loop* `FALSE` event, execution continues until the end of the current cycle or until *stopTime* (if *stopTime* > *startTime*), whichever occurs first. The termination at the end of cycle can be overridden by a subsequent *set\_loop* `TRUE` event.

Any *set\_startTime* events to an active time-dependent node are ignored. Any *set\_stopTime* event where *stopTime* ≤ *startTime* sent to an active time-dependent node is also ignored. A *set\_stopTime* event where *startTime* < *stopTime* ≤ *now* sent to an active time-dependent node results in events being generated as if *stopTime* has just been reached. That is, final events, including an *isActive* `FALSE`, are generated and the node becomes inactive. The *stopTime\_changed* event will have the *set\_stopTime* value. Other final events are node-dependent (see [8.4.1 TimeSensor](#)).

A time-dependent node may be restarted while it is active by sending a *set\_stopTime* event equal to the current time (which will cause the node to become inactive) and a *set\_startTime* event, setting it to the current time or any time in the future. These events will have the same time stamp and should be processed as *set\_stopTime*, then *set\_startTime* to produce the correct behaviour.

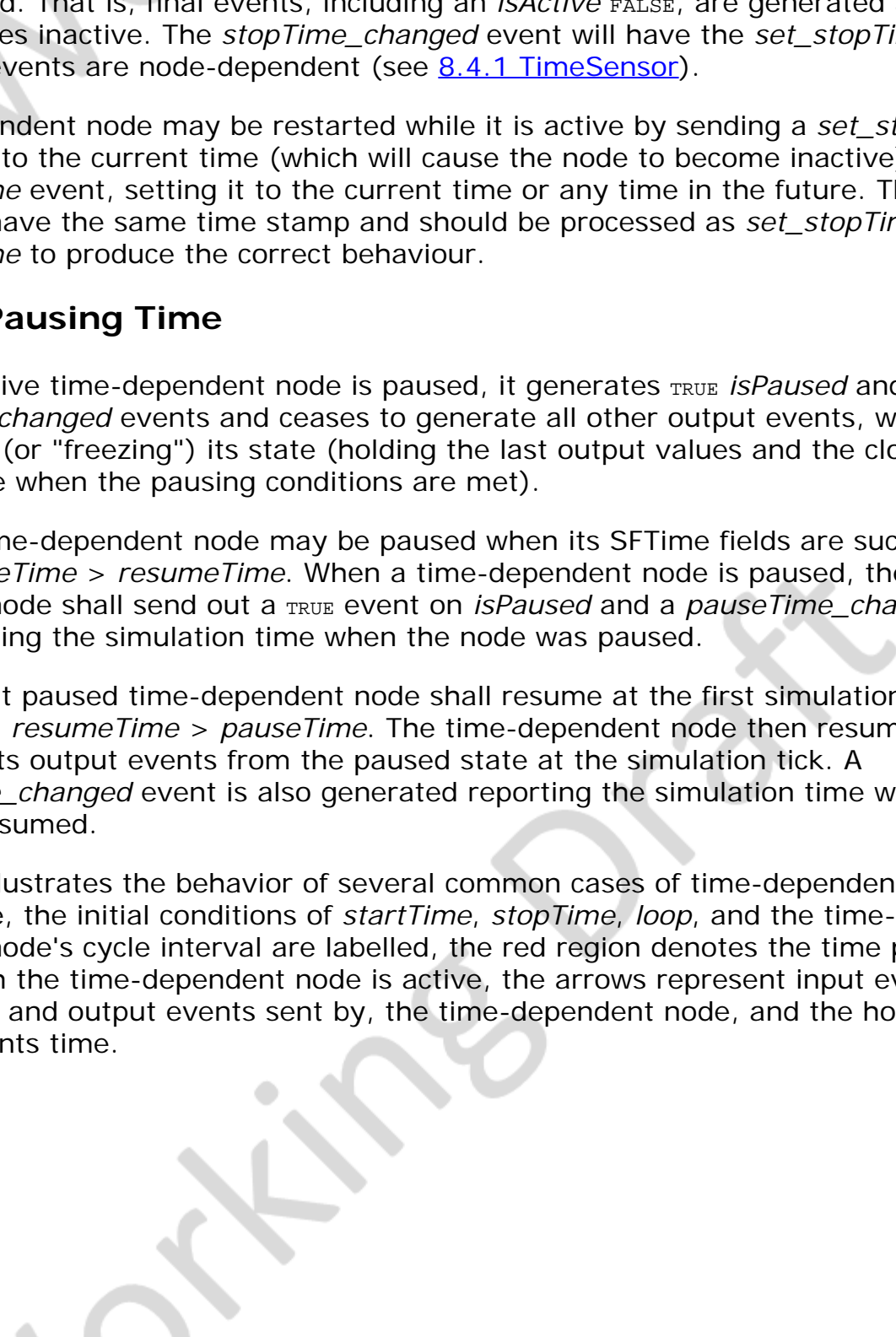
#### 8.2.4.4 Pausing Time

While an active time-dependent node is paused, it generates `TRUE` *isPaused* and *pauseTime\_changed* events and ceases to generate all other output events, while maintaining (or "freezing") its state (holding the last output values and the clock's internal time when the pausing conditions are met).

An active time-dependent node may be paused when its SFTIME fields are such that *now* ≥ *pauseTime* > *resumeTime*. When a time-dependent node is paused, the time-dependent node shall send out a `TRUE` event on *isPaused* and a *pauseTime\_changed* event reporting the simulation time when the node was paused.

An active but paused time-dependent node shall resume at the first simulation tick when *now* ≥ *resumeTime* > *pauseTime*. The time-dependent node then resumes generating its output events from the paused state at the simulation tick. A *resumeTime\_changed* event is also generated reporting the simulation time when the node was resumed.

[Figure 8.1](#) illustrates the behavior of several common cases of time-dependent nodes. In each case, the initial conditions of *startTime*, *stopTime*, *loop*, and the time-dependent node's cycle interval are labelled, the red region denotes the time period during which the time-dependent node is active, the arrows represent input events received by, and output events sent by, the time-dependent node, and the horizontal axis represents time.



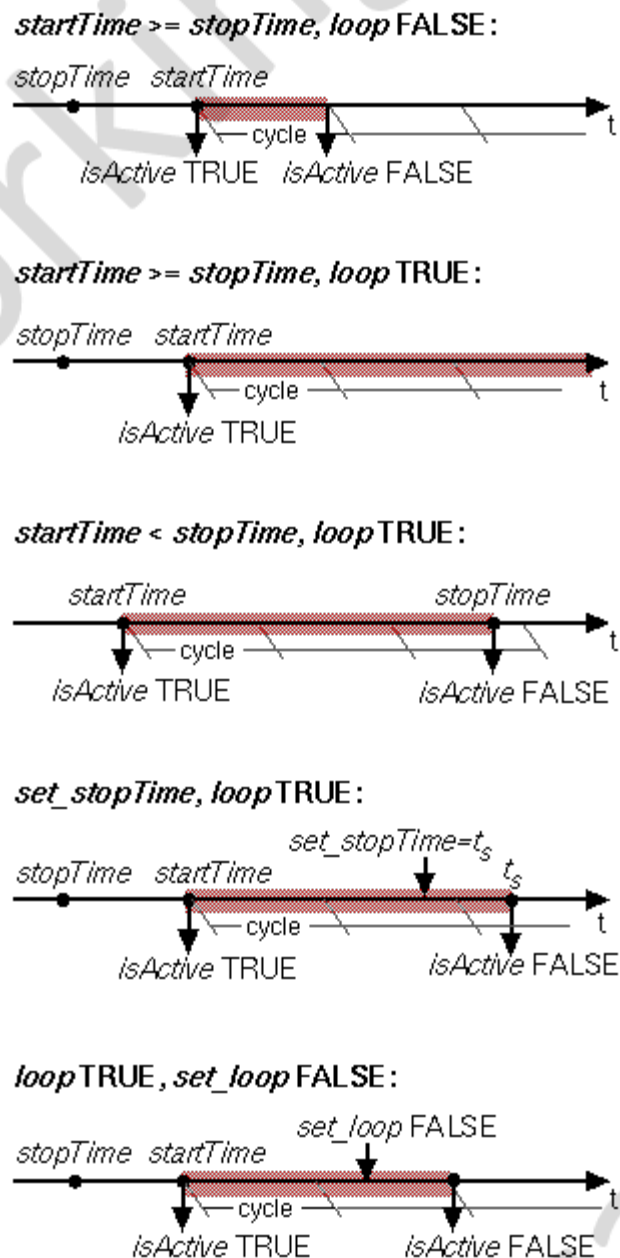


Figure 8.1 — Examples of time-dependent node execution

## 8.3 Abstract types

### 8.3.1 X3DTimeDependentNode

```

X3DTimeDependentNode : X3DChildNode {
  SFString [in,out] description ""
  SFBool [in,out] enabled FALSE
  SFBool [in,out] loop FALSE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTime [in,out] pauseTime 0 (-∞,∞)
  SFTime [in,out] resumeTime 0 (-∞,∞)
  SFTime [in,out] startTime 0 (-∞,∞)
  SFTime [in,out] stopTime 0 (-∞,∞)
  SFTime [out] elapsedTime
  SFBool [out] isActive
  SFBool [out] isPaused
}
    
```

This abstract node type is the base node type from which all time-dependent nodes are derived.

The description field specifies a textual description for intended purpose of the node. This information is beneficial for authoring, and may be used by optional browser-specific user interfaces that present users with more detailed information about active time-dependent behavior.

The *enabled* field enables and disables operation in a manner appropriate for the associated node.

See [8.2 Concepts](#) for a detailed discussion of fields in time-dependent nodes.

## 8.4 Node reference

### 8.4.1 TimeSensor

```
TimeSensor : X3DTimeDependentNode, X3DSensorNode {
  SFTIME [in,out] cycleInterval 1 (0,∞)
  SFString [in,out] description ""
  SFBool [in,out] enabled TRUE
  SFBool [in,out] loop FALSE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTIME [in,out] pauseTime 0 (-∞,∞)
  SFTIME [in,out] resumeTime 0
  SFTIME [in,out] startTime 0 (-∞,∞)
  SFTIME [in,out] stopTime 0 (-∞,∞)
  SFTIME [out] cycleTime
  SFTIME [out] elapsedTime
  SFFloat [out] fraction_changed
  SFBool [out] isActive
  SFBool [out] isPaused
  SFTIME [out] time
}
```

TimeSensor nodes generate events as time passes. TimeSensor nodes can be used for many purposes including:

- driving continuous simulations and animations;
- controlling periodic activities (e.g., one per minute);
- initiating single occurrence events such as an alarm clock.

The TimeSensor node contains two discrete outputOnly fields: *isActive* and *cycleTime*. The *isActive* outputOnly field sends `TRUE` when the TimeSensor node begins running, and `FALSE` when it stops running. The *cycleTime* outputOnly field sends a time event at *startTime* and at the beginning of each new cycle (useful for synchronization with other time-based objects). The remaining outputOnly fields generate continuous events. The *fraction\_changed* outputOnly field, an SFFloat in the closed interval  $[0,1]$ , sends the completed fraction of the current cycle. The *time* outputOnly field sends the absolute time for a given *simulation tick*.

If the *enabled* field is `TRUE`, the TimeSensor node is enabled and may be running. If a *set\_enabled FALSE* event is received while the TimeSensor node is running, the sensor performs the following actions:

- evaluates and sends all relevant outputs;
- sends a `FALSE` value for *isActive*;
- disables itself.



Input events on the fields of the TimeSensor node (e.g., *set\_startTime*) are processed and their corresponding outputOnly fields (e.g., *startTime\_changed*) are sent regardless of the state of the *enabled* field. The remaining discussion assumes *enabled* is `TRUE`.

The *loop*, *startTime*, *stopTime* and *isActive* fields and their effects on the TimeSensor node are discussed in detail in [8.2 Concepts](#). The "cycle" of a TimeSensor node lasts for *cycleInterval* seconds. The value of *cycleInterval* shall be greater than zero.

A *cycleTime* outputOnly field can be used for synchronization purposes such as sound with animation. The value of a *cycleTime* event will be equal to the time at the beginning of the current cycle. A *cycleTime* event is generated at the beginning of every cycle, including the cycle starting at *startTime*. The first *cycleTime* event for a TimeSensor node can be used as an alarm (single pulse at a specified time).

When a TimeSensor node becomes active, it generates an *isActive* = `TRUE` event and begins generating *time*, *fraction\_changed*, and *cycleTime* events which may be routed to other nodes to drive animation or simulated behaviours. The behaviour at read time is described below. The *time* event sends the absolute time for a given tick of the TimeSensor node ([SFTIME/MFTIME](#) fields and events represent the number of seconds since midnight GMT January 1, 1970).

*fraction\_changed* events output a floating point value in the closed interval [0, 1]. At *startTime* the value of *fraction\_changed* is 0. After *startTime*, the value of *fraction\_changed* in any cycle will progress through the range (0.0, 1.0]. At *startTime* +  $N \times \textit{cycleInterval}$ , for  $N = 1, 2, \dots$ , (i.e., at the end of every cycle), the value of *fraction\_changed* is 1.

Let *now* represent the time at the current simulation tick. Then the *time* and *fraction\_changed* output-only fields can then be computed as:

```
time = now
temp = (now - startTime) / cycleInterval
f = fractionalPart(temp)
if (f == 0.0 && now > startTime) fraction_changed = 1.0
else fraction_changed = f
```

where `fractionalPart(x)` is a function that returns the fractional part, (that is, the digits to the right of the decimal point), of a nonnegative floating point number.

A TimeSensor node can be set up to be active at read time by specifying *loop* `TRUE` (not the default) and *stopTime* less than or equal to *startTime* (satisfied by the default values). The *time* events output absolute times for each tick of the TimeSensor node simulation. The *time* events shall start at the first simulation tick greater than or equal to *startTime*. *time* events end at *stopTime*, or at *startTime* +  $N \times \textit{cycleInterval}$  for some positive integer value of  $N$ , or loop forever depending on the values of the other fields. An active TimeSensor node shall stop at the first simulation tick when  $\textit{now} \geq \textit{stopTime} > \textit{startTime}$ .

No guarantees are made with respect to how often a TimeSensor node generates time events, but a TimeSensor node shall generate events at least at every simulation tick. TimeSensor nodes are guaranteed to generate final *time* and *fraction\_changed* events. If *loop* is `FALSE` at the end of the  $N$ th *cycleInterval* and was `TRUE` at *startTime* +  $M \times \textit{cycleInterval}$  for all  $0 < M < N$ , the final *time* event will be generated with a value of ( $\textit{startTime} + N \times \textit{cycleInterval}$ ) or *stopTime* (if  $\textit{stopTime} > \textit{startTime}$ ),



whichever value is less. If *loop* is `TRUE` at the completion of every cycle, the final event is generated as evaluated at *stopTime* (if *stopTime* > *startTime*) or *never*.

An active TimeSensor node ignores *set\_cycleInterval* and *set\_startTime* events. An active TimeSensor node also ignores *set\_stopTime* events for *set\_stopTime* less than or equal to *startTime*. For example, if a *set\_startTime* event is received while a TimeSensor node is active, that *set\_startTime* event is ignored (the *startTime* field is not changed, and a *startTime\_changed* event is not generated). If an active TimeSensor node receives a *set\_stopTime* event that is less than the current time, and greater than *startTime*, it behaves as if the *stopTime* requested is the current time and sends the final events based on the current time (note that *stopTime* is set as specified in the field).

A TimeSensor read from a X3D file shall generate *isActive* `TRUE`, *time* and *fraction\_changed* events if the sensor is enabled and all conditions for a TimeSensor to be active are met.

## 8.5 Support levels

The Time component provides four levels of support as specified in [Table 8.2](#). Level 1 provides basic support for TimeSensor. Level 2 adds support for all of the fields of the TimeSensor node.

**Table 8.2 — Time component support levels**

Level	Prerequisites	Nodes/Features	Support
<b>1</b>	Core 1		
		<i>X3DTimeDependentNode</i> (abstract)	n/a
		TimeSensor	<i>pause</i> optionally supported. <i>isPaused</i> optionally supported. <i>resumeTime</i> optionally supported.
<b>2</b>	Core 1		
		Level 1 supported node	All fields as supported by Level 1.
		TimeSensor	All fields fully supported.





## Extensible 3D (X3D) Part 1: Architecture and base components

### 29 Scripting component

#### 29.1 Introduction

##### 29.1.1 Name

The name of this component is "Scripting". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 29.1.2 Overview

This clause describes the scripting component of this part of ISO/IEC 19775. This includes how [Script](#) nodes are used to effect changes in X3D worlds. [Table 29.1](#) provides links to the major topics in this clause.

**Table 29.1 — Topics**

- [29.1 Introduction](#)
  - [29.1.1 Name](#)
  - [29.1.2 Overview](#)
- [29.2 Concepts](#)
  - [29.2.1 Overview](#)
  - [29.2.2 Script execution](#)
  - [29.2.3 Initialize\(\) and shutdown\(\)](#)
  - [29.2.4 EventsProcessed\(\)](#)
  - [29.2.5 PrepareEvents\(\)](#)
  - [29.2.6 Scripts with direct outputs](#)
  - [29.2.7 Asynchronous scripts](#)
  - [29.2.8 Script languages](#)
  - [29.2.9 Event handling](#)
  - [29.2.10 Accessing fields and events](#)
- [29.3 Abstract types](#)
  - [29.3.1 X3DScriptNode](#)
- [29.4 Node reference](#)

### 29.4.1 Script

- [29.5 Support levels](#)
- [Table 29.1 — Topics](#)
- [Table 29.2 — Script component support levels](#)

## 29.2 Concepts

### 29.2.1 Overview

Authors often require that X3D worlds change dynamically in response to user inputs, external events, and the current state of the world. The proposition "if the vault is currently closed AND the correct combination is entered, open the vault" illustrates the type of problem which may need addressing. These kinds of decisions are expressed programmatically using the Scene Access Interface (SAI) specified in Part 2 of ISO/IEC 19775. The programmatic elements are provided internally from [Script](#) nodes (see [29.4.1 Script](#)) or externally from other application programs. These application programs are called *scripting environments*. In both cases, the scripting environment can receive events, process them, and send new events. Scripting environments can keep track of information between subsequent executions (*i.e.*, retaining internal state over time).

This clause describes the general mechanisms and semantics of all scripting access. [2. \[19775-2\]](#) defines a set of abstract scripting services and specific languages bound to those services. For internal scripting, event processing is performed by a program or script contained in (or referenced by) the Script node's *url* field. This program or script may be written in any programming language that the browser supports.

### 29.2.2 Script execution

A [Script](#) node is activated when it receives an event. The browser shall then execute the program in the Script node's *url* field (passing the program to an external interpreter if necessary). The program can perform a wide variety of actions including sending out events (and thereby changing the scene), performing calculations, and communicating with servers elsewhere on the Internet. A detailed description of the ordering of event processing is contained [4.4.8 Event model](#).

Script nodes may also be executed at initialization and shutdown as specified in [29.2.3 \*initialize\(\)\* and \*shutdown\(\)\*](#). Some scripting languages may allow the creation of separate processes from scripts, resulting in continuous execution (see [29.2.7 \*Asynchronous scripts\*](#)).

Script nodes receive events in timestamp order. Any events generated as a result of processing an event are given timestamps corresponding to the event that generated them. Conceptually, it takes no time for a Script node to receive and process an event, even though in practice it does take some amount of time to execute a Script.

When a *set\_url* event is received by a Script node that contains a script that has been previously initialized for a different URL, the *shutdown()* service of the current script is

called (see [29.2.3 \*initialize\(\)\* and \*shutdown\(\)\*](#)). Until the new script becomes available, the script shall behave as though it has no executable content. When the new script becomes available, the *Initialize()* service is invoked. The limiting case is when the URL contains inline code that can be immediately executed upon receipt of the *set\_url* event (EXAMPLE `ecmascript: protocol`). In this case, it can be assumed that the old code is unloaded and the new code loaded instantaneously, after any dynamic route requests have been performed.

### **29.2.3 *initialize()* and *shutdown()***

The scripting language binding may define an *initialize()* method. This method shall be invoked before the browser presents the world to the user and before any events are processed by any nodes in the same X3D file as the [Script](#) node containing this script. Events generated by the *initialize()* method shall have timestamps less than any other events generated by the Script node. This allows script initialization tasks to be performed prior to the user interacting with the world.

Likewise, the scripting language binding may define a *shutdown()* method. This method shall be invoked when the corresponding Script node is deleted or the world containing the Script node is unloaded or replaced by another world. This method may be used as a clean-up operation, such as informing external mechanisms to remove temporary files. No other methods of the script may be invoked after the *shutdown()* method has completed, though the *shutdown()* method may invoke methods or send events while shutting down. Events generated by the *shutdown()* method that are routed to nodes that are being deleted by the same action that caused the *shutdown()* method to execute will not be delivered. The deletion of the Script node containing the *shutdown()* method is not complete until the execution of its *shutdown()* method is complete.

### **29.2.4 *eventsProcessed()***

The scripting language binding may define an *eventsProcessed()* method that is called after one or more events are received. This method allows scripts that do not rely on the order of events received to generate fewer events than an equivalent script that generates events whenever events are received. If it is used in some other time-dependent way, *eventsProcessed()* may be nondeterministic, since different browser implementations may call *eventsProcessed()* at different times.

For a single event cascade, a given [Script](#) node's *eventsProcessed()* method shall be called at most once. Events generated from an *eventsProcessed()* method are given the timestamp of the last event processed.

### **29.2.5 *prepareEvents()***

The scripting language binding may define a *prepareEvents()* method that is called only once per timestamp. *prepareEvents()* is called before any ROUTE processing and allows a [Script](#) to collect any asynchronously generated data, such as input from a network queue or the results of calling field listeners, and generate events to be handled by the browser's normal event processing sequence as if it were a built-in sensor node.

### **29.2.6 Scripts with direct outputs**

[Script](#) nodes that have access to other nodes (via `SFNode` and `MNode` fields) and that have their `directOutput` field set to `TRUE` may directly post events to those nodes. They may also read the last value sent from any of the node's fields.

When setting a value in another node, implementations shall set values in other nodes by sending input events to the corresponding fields. These events shall be part of the current event cascade (see [4.4.8.3 Execution model](#)).

## 29.2.7 Asynchronous scripts

Some languages supported by X3D browsers may allow [Script](#) nodes to spontaneously generate events, allowing users to create Script nodes that function like new [X3DSensorNode](#) nodes. In these cases, the Script is generating the initial events that causes the event cascade, and the scripting language and/or the browser shall determine an appropriate timestamp for that initial event. Such events are then sorted into the event stream and processed like any other event, following all of the same rules including those for looping.

## 29.2.8 Script languages

The Script node's `url` field shall allow for both inline scripting and script reference via a URL. The MIME-type of the returned data defines the language type. Additionally, instructions can be included in-line using scripting language protocols as defined in [9.2.3 Scripting language protocols](#) for the specific language (from which the language type is inferred).

EXAMPLE The following [Script](#) node has one field named `start` and three different URL values specified in the `url` field: Java, ECMAScript, and inline ECMAScript:

```
Script {
  field SFBool start
  url [ "http://foo.com/fooBar.class",
        "http://foo.com/fooBar.js",
        "ecmascript:function start(value, timestamp) { ... }"
  ]
}
```

When a `start` event is received by the Script node, one of the scripts found in the `url` field is executed. The Java platform bytecode is the first choice, the ECMAScript code is the second choice, and the inline ECMAScript code the third choice.

A description of order of preference for multiple valued URL fields may be found in [9.3.2 X3DUrlObject](#).

## 29.2.9 Event handling

Events received by the [Script](#) node are passed to the appropriate scripting language method in the script. The method's name depends on the language type used. In some cases, it is identical to the name of the field; in others, it is a general callback method for all events (see the scripting language annexes for details). The method is passed two arguments: the event value and the event timestamp.

## 29.2.10 Accessing fields and events

The fields of a [Script](#) node are accessible from scripting language methods. Events can be routed to fields of Script nodes and the fields of Script nodes can be routed to fields of other nodes. Another Script node with access to this node can access the fields just like any other node (see [29.2.6 Scripts with direct outputs](#)).

It is recommended that user-defined field names defined in Script nodes follow the naming conventions described in [2.\[19775-2\]](#)

The field values can be read or written and are persistent across method call, and changes to a field can notify the node through its update method. See [5 Field type reference](#) for more information on field types.

## 29.3 Abstract types

### 29.3.1 X3DScriptNode (abstract)

```
X3DScriptNode : X3DChildNode, X3DURLObject {
  SFString [in,out] description ""
  SFBool [in,out] load TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTime [in,out] refresh 0.0 [0,∞)
  MFString [in,out] url [] [URI]
}
```

This abstract node type is the base type for all scripting nodes.

## 29.4 Node reference

### 29.4.1 Script

```
Script : X3DScriptNode {
  SFString [in,out] description ""
  SFBool [in,out] load TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTime [in,out] refresh 0.0 [0,∞)
  MFString [in,out] url [] [URI]
  SFBool [] directOutput FALSE
  SFBool [] mustEvaluate FALSE
  # And any number of:
  fieldType [in] fieldName
  fieldType [in,out] fieldName initialValue
  fieldType [out] fieldName
  fieldType [] fieldName initialValue
}
```

The Script node is used to program behaviour in a scene. Script nodes typically:

- signify a change or user action;
- receive events from other nodes;
- contain a program module that performs some computation;
- effect change somewhere else in the scene by sending events.

Each Script node has associated programming language code, referenced by the *url* field, that is executed to carry out the Script node's function. That code is referred to as the "script" in the rest of this description. Details on the *url* field can be found in [9.2.1 URLs](#).

Browsers are not required to support any specific language. Detailed information on scripting languages is described in [29.2 Concepts](#). Browsers supporting a scripting language for which a language binding is specified shall adhere to that language binding



(see [ISO/IEC 19777](#)).

Sometime before a script receives the first event it shall be initialized (any language-dependent or user-defined *initialize()* is performed). The script is able to receive and process events that are sent to it. Each event that can be received shall be declared in the Script node using the same syntax as is used in a prototype definition:

```
inputOnly type name
```

The *type* can be any of the standard X3D fields (as defined in [5 Field type reference](#)). *Name* shall be an identifier that is unique for this Script node.

The Script node is able to generate events in response to the incoming events. Each event that may be generated shall be declared in the Script node using the following syntax:

```
outputOnly type name
```

If the Script node's *mustEvaluate* field is `FALSE`, the browser may delay sending input events to the script until its outputs are needed by the browser. If the *mustEvaluate* field is `TRUE`, the browser shall send input events to the script as soon as possible, regardless of whether the outputs are needed. The *mustEvaluate* field shall be set to `TRUE` only if the Script node has effects that are not known to the browser (such as sending information across the network). Otherwise, poor performance may result.

Once the script has access to a X3D node (via an SFNode or MFNode value either in one of the Script node's fields or passed in as an attribute), the script is able to read the contents of that node's fields. If the Script node's *directOutput* field is `TRUE`, the script may also send events directly to any node to which it has access, and may dynamically establish or break routes. If *directOutput* is `FALSE` (the default), the script may only affect the rest of the world via events sent through its fields. The results are undefined if *directOutput* is `FALSE` and the script sends events directly to a node to which it has access.

A script is able to communicate directly with the X3D browser to get information such as the current time and the current world URL. This is strictly defined generally by the SAI services (see [Part 2 of ISO/IEC 19775](#)) and by the language bindings of the SAI (see [ISO/IEC 19777](#)) for the specific scripting language being used.

The location of the Script node in the scene graph has no affect on its operation.

EXAMPLE If a parent of a Script node is a [Switch](#) node with *whichChoice* set to "-1" (i.e., ignore its children), the Script node continues to operate as specified (i.e., it receives and sends events).

If the *refresh* field results in a new script getting loaded or the prior script getting reloaded, then all fields are re-initialized to their initially defined values, and the *initialize()* method is invoked, if provided, as defined in [29.2.3 initialize\(\) and shutdown\(\)](#).

**WARNING** Automatically reloading content can have security considerations and needs to be considered carefully.

## 29.5 Support levels

The Scripting component provides one level of support as specified in [Table 29.2](#).

**Table 29.2 — Scripting component support levels**

Level	Prerequisites	Nodes/Features	Support
1	Core 1		
		<i>X3ScriptNode</i> (abstract)	n/a
		Script node	All fields fully supported.







# Extensible 3D (X3D) Part 1: Architecture and base components

## Annex H

(normative)

### CADInterchange profile

---



#### H.1 General

This annex defines the X3D components that comprise the CADInterchange profile. This annex includes not only the nodes that shall be supported but also which fields in the supported nodes may be ignored.

This profile is targeted towards:

- Distillation of computer-aided design (CAD) data to downstream applications.
- Appropriately supporting Geometry and Appearance capabilities data for CAD.

#### H.2 Topics

[Table H.1](#) provides links to the major topics in this annex.

**Table H.1 — Topics**

- |  |
|--|
| <ul style="list-style-type: none"><li>• <a href="#">H.1 General</a></li><li>• <a href="#">H.2 Topics in this annex</a></li><li>• <a href="#">H.3 Component support</a></li><li>• <a href="#">H.4 Conformance criteria</a></li><li>• <a href="#">H.5 Node set</a></li><li>• <a href="#">H.6 Other limitations</a></li><br/><li>• <a href="#">Table H.1 — Topics</a></li><li>• <a href="#">Table H.2 — Components and levels</a></li><li>• <a href="#">Table H.3 — Nodes for conforming to the CADInterchange profile</a></li><li>• <a href="#">Table H.4 — Other limitations</a></li><li>• <a href="#">Table H.5 — Node set</a></li></ul> |
|--|

## H.3 Component support

[Table H.2](#) lists the components and their levels which shall be supported in the CADInterchange profile. Tables H.2 and H.3 describe limitations on required support for nodes and fields contained within these components.

**Table H.2 — Components and levels**

Component	Level	Reference
Core	1	<a href="#">7.5 Support levels</a>
Networking	2	<a href="#">9.5 Support levels</a>
Grouping	1	<a href="#">10.5 Support levels</a>
Rendering	4	<a href="#">11.5 Support levels</a>
Shape	2	<a href="#">12.5 Support levels</a>
Lighting	1	<a href="#">17.5 Support levels</a>
Texturing	2	<a href="#">18.5 Support levels</a>
Navigation	2	<a href="#">23.4 Support levels</a>
Shaders	1	<a href="#">31.5 Support levels</a>
CADGeometry	2	<a href="#">32.5 Support levels</a>

## H.4 Conformance criteria

Conformance to this profile shall include conformance criteria defined by the specifications for those components and levels listed in [Table H.2](#).

In Tables H.3 and H.4, the first column defines the item for which conformance is being defined. In some cases, general limits are defined but are later overridden in specific cases by more restrictive limits. The second column defines the requirements for an X3D file conforming to the CADInterchange profile; if an X3D file contains any items that exceed these limits, it may not be possible for an X3D browser conforming to the CADInterchange profile to successfully parse that X3D file. The third column defines the minimum complexity for an X3D scene that an X3D browser conforming to the CADInterchange profile shall be able to present to the user. Fields flagged as "not supported" may be supported by browsers which conform to the CADInterchange profile. The word "ignore" in the minimum browser support column refers only to the display of the item; in particular, *set\_* events to ignored inputOutput fields shall still generate corresponding *\_changed* events.

## H.5 Node set

[Table H.3](#) lists the nodes which shall be supported in the CADInterchange profile and specifies any fields in these nodes for which this profile requires less than full support.

**Table H.3 — Nodes for conforming to the CADInterchange profile**

Item	X3D File Limit	Minimum Browser Support
Anchor	No restrictions.	Full support.
Appearance	No restrictions.	<i>fillProperties</i> not supported.
CADAssembly	No restrictions.	Full support.
CADFace	No restrictions.	Full support.
CADLayer	No restrictions.	Full Support
CADPart	No restrictions.	Full support.
Billboard	No restrictions.	Can treat as just a grouping node, no runtime requirements
Collision	No restrictions.	Can treat as just a grouping node, no runtime requirements
Color	5,592,405 colours.	5,592,405 colours.
ColorRGBA	4,194,304 colours.	4,194,304 colours
Coordinate	16,777,216 points	16,777,216 points.
DirectionalLight	No restrictions.	Full support.
FragmentShader	No restrictions.	Full support.
Group	Restrictions as for all groups.	<i>addChilden</i> not supported. <i>removeChildren</i> not supported. Otherwise as for all groups.
ImageTexture	JPEG ( <a href="#">2. [JPEG]</a> ) and PNG ( <a href="#">2. [115948]</a> ) format.	JPEG ( <a href="#">2. [JPEG]</a> ) and PNG ( <a href="#">2. [115948]</a> ) format.
	5,592,405 total vertices.	5,592,405 total vertices.

IndexedLineSet	5,592,405 indices in any index field.	5,592,405 indices in any index field.
IndexedQuadSet	5,592,405 total faces. 5,592,405 indices in any index field.	5,592,405 total faces. 5,592,405 indices in any index field.
IndexedTriangleFanSet	5,592,405 total faces. 5,592,405 indices in any index field.	5,592,405 total faces. 5,592,405 indices in any index field.
IndexedTriangleSet	5,592,405 total faces. 5,592,405 indices in any index field.	5,592,405 total faces. 5,592,405 indices in any index field.
IndexedTriangleStripSet	5,592,405 total faces. 5,592,405 indices in any index field.	5,592,405 total faces. 5,592,405 indices in any index field.
Inline	No restrictions.	Optional support for <i>load</i> field. Other fields full support.
LineProperties	No restrictions.	Full support.
LineSet	5,592,405 total vertices.	5,592,405 total vertices.
LOD	No restrictions.	Runtime switching not required. An implementation can select one level and display
Material	No restrictions.	Full support.
MetadataBoolean	No restrictions.	Full support.
MetadataDouble	No restrictions.	Full support.
MetadataFloat	No restrictions.	Full support.
MetadataInteger	No restrictions.	Full support.
MetadataSet	No restrictions.	Full support.
MetadataString	No restrictions.	Full support.
MultiShader	No restrictions.	Full support.

MultiTexture	No restrictions.	At least one texture displayed per node with any number specified. Full support.
MultiTextureCoordinate	15,000 coordinates.	15,000 coordinates.
MultiTextureTransform	No restrictions.	Full support.
NavigationInfo	No restrictions.	<i>avatarSize</i> optionally supported. <i>speed</i> optionally supported. <i>type</i> optionally supported. <i>visibilityLimit</i> optionally supported.
Normal	5,592,405 normals	5,592,405 normals.
PixelTexture	512 width. 512 height.	512 width. 512 height. Display fully transparent and fully opaque pixels.
PointSet	5,592,405 points.	5,592,405 points.
QuadSet	5,592,405 total faces. 5,592,405 indices in any index field.	5,592,405 total faces. 5,592,405 indices in any index field.
Shader	No restrictions.	Full support.
ShaderAppearance	No restrictions.	Full support.
Shape	No restrictions.	Full support.
TextureCoordinate	5,592,405 coordinates.	5,592,405 coordinates.
TextureCoordinateGenerator	No restrictions.	Full support.
TextureTransform	No restrictions.	Full support.
Transform	Restrictions as for all groups.	<i>addChildren</i> not supported. <i>removeChildren</i> not supported. Otherwise, full support except as for all groups.
TriangleFanSet	5,592,405 triangles per fan. 5,592,405	5,592,405 triangles per fan. 5,592,405 total triangles.

	total triangles.	
TriangleSet	5,592,405 triangles	5,592,405 triangles
TriangleStripSet	5,592,405 triangles per strip. 5,592,405 total triangles	5,592,405 triangles per strip. 5,592,405 total triangles.
Viewpoint	No restrictions.	Full support.
VertexShader	No restrictions.	Full support.
WorldInfo	No restrictions.	Full support.

## H.6 Other limitations

[Table H.4](#) specifies other aspects of X3D functionality which are supported by this profile. Note that general items refer only to those specific nodes listed in [Table H.3](#).

**Table H.4 — Other limitations**

Item	X3D File Limit	Minimum Browser Support
All groups	16777216 children.	16777216 children.
All lights	8 simultaneous lights.	8 simultaneous lights.
Names for DEF/field	50 utf8 octets.	50 utf8 octets.
All <i>url</i> fields	10 URLs.	10 URLs. URN's ignored. Support "http", "file", and "ftp" protocols. Support relative URLs where relevant.
SFBool	No restrictions.	Full support.
SFColor	No restrictions.	Full support.
SFColorRGBA	No restrictions.	Full support.
SFDouble	No restrictions.	Full support. Range $\pm 1e\pm 12$ . Precision $1e-7$ .
SFFloat	No restrictions.	Full support.
SFImage	512 width. 512 height.	512 width. 512 height.

SFInt32	No restrictions.	Full support.
SFNode	No restrictions.	Full support.
SFRotation	No restrictions.	Full support.
SFString	30,000 utf8 octets.	30,000 utf8 octets.
SFTime	No restrictions.	Full support.
SFVec2d	15,000 values.	15,000 values.
SFVec2f	15,000 values.	15,000 values.
SFVec3d	15,000 values.	15,000 values.
SFVec3f	15,000 values.	15,000 values.
MFColor	15,000 values.	15,000 values.
MFColorRGBA	15,000 values.	15,000 values.
MFDouble	1000 values.	1000 values.
MFFloat	1,000 values.	1,000 values.
MFInt32	20,000 values.	20,000 values.
MFNode	500 values.	500 values.
MFRotation	1,000 values.	1,000 values.
MFString	30,000 utf8 octets per string, 10 strings.	30,000 utf8 octets per string, 10 strings.
MFTime	1,000 values.	1,000 values.
MFVec2d	15,000 values.	15,000 values.
MFVec2f	15,000 values.	15,000 values.
MFVec3d	15,000 values.	15,000 values.
MFVec3f	15,000 values.	15,000 values.







## Extensible 3D (X3D) Part 1: Architecture and base components

### 9 Networking component

---



#### 9.1 Introduction

##### 9.1.1 Name

The name of this component is "Networking". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 9.1.2 Overview

This clause describes the Networking component of this part of ISO/IEC 19775. This component defines the node types and other features used to access file-based and streaming resources on the World Wide Web. [Table 9.1](#) lists the major topics in this clause.

**Table 9.1 — Topics**

- [9.1 Introduction](#)
  - [9.1.1 Name](#)
  - [9.1.2 Overview](#)
- [9.2 Concepts](#)
  - [9.2.1 URLs](#)
  - [9.2.2 Relative URLs](#)
  - [9.2.3 Scripting language protocols](#)
  - [9.2.4 Browser options](#)
  - [9.2.5 IMPORT statement](#)
  - [9.2.6 EXPORT statement](#)
- [9.3 Abstract types](#)
  - [9.3.1 X3DNetworkSensorNode](#)
  - [9.3.2 X3DUriObject](#)
- [9.4 Node reference](#)
  - [9.4.1 Anchor](#)
  - [9.4.2 Inline](#)
  - [9.4.3 LoadSensor](#)

- [9.5 Support levels](#)
- [Table 9.1 — Topics](#)
- [Table 9.2 — Browser options](#)
- [Table 9.3 — Networking component support levels](#)

## 9.2 Concepts

### 9.2.1 URLs

A *URL* (Uniform Resource Locator), described in [2.\[RFC1738\]](#), is a form of Universal Resource Identifier (URI) that specifies a file located on a particular server and accessed through a specified protocol (such as `file:`, `http:` or `https:`). In this part of ISO/IEC 19775, the upper-case term URL refers to a Uniform Resource Locator, while the italicized lower-case version *url* refers to a field which may contain URLs or in-line encoded data.

Higher levels of this component extend the URL support of a browser to additional forms of URI, such as supporting *URNs* (Uniform Resource Name), which are another form of URI. A URN allows an abstract resolution mechanism to be invoked to locate a resource (see [2.\[RFC2141\]](#)). This allows a resource to be located on the local machine or a platform dependent resource to be located using the URN along with platform-specific identifiers.

For levels that support URNs, the *url* field shall also support the Web3D Consortium URN Namespace (see [2.\[RFC3541\]](#)) and also support the Universal Media Library that may be accessed using that namespace. A URN allows an abstract resolution mechanism to be invoked to locate a resource (see [2.\[RFC2141\]](#)). This allows a resource to be located on the local machine or a platform dependent resource to be located using the URN along with platform specific identifiers. A component extension can extend the URL support of a browser by supporting other URN naming schemes. More information on the *url* field may be found in [9.3.2 X3DUrlObject](#).

More general information on URLs is described in [2.\[RFC1738\]](#).

### 9.2.2 Relative URLs

Relative URLs are handled as described in [2.\[RFC1808\]](#). All name scopes (see [4.4.7 Run-time name scope](#)) maintain a base URI which is used for all relative URLs within that name scope. Whenever a node with a relative URL is defined, that node may only reference assets available within its name scope. It has no advance knowledge of how it may or may not be included by an Inline node or referenced by an external prototype instantiation. The base document for EXTERNPROTO statements or nodes that contain a URL field is:

- a. The X3D file in which an EXTERNPROTO is declared, namely the value of the *externprotoURL* field specified in [7.2.5.9 EXTERNPROTO statement](#).
- b. The X3D file in which the parent PROTO is declared, if the statement is inside the body of a prototype declaration, namely the value of the *protoDefinition* field

specified in [7.2.5.8 PROTO statement](#).

- c. The X3D file in which a Script is defined.
- d. Otherwise, the X3D file from which the statement is read.

### 9.2.3 Scripting language protocols

Components can add scripting support to an X3D browser. An example of this is the Scripting component which introduces a [Script](#) node. The Script node's *url* field may support custom protocols for the various scripting languages. For example, a script *url* prefixed with *ecmascript:* (or the deprecated *javascript:*) shall contain ECMAScript source, with line terminators allowed in the string. The details of each language protocol are defined in the parts of [ISO/IEC 19777](#), which define the bindings for each language. Browsers that conform to a profile that supports scripting are not required to support both the Java and ECMAScript scripting languages. Browsers shall adhere to the protocol defined in the corresponding part of [ISO/IEC 19777](#) for any scripting language that is supported.

EXAMPLE The following illustrates the use of mixing custom protocols and standard protocols in a single *url* field (order of precedence determines priority):

```
#X3D v3.0 utf8
Script {
  url [ "ecmascript: ...", # custom in-line ECMAScript code
        "http://bar.com/foo.js", # ECMAScript file reference
        "http://bar.com/foo.class" ] # Java platform bytecode file reference
}
```

The "..." represents in-line ECMAScript source code.

### 9.2.4 Browser options

X3D supports configuring the browser via a set of options. These options are values passed to the browser at start-up time that control its run-time operation. Browser options may be set as HTML PARAM values within an EMBED or OBJECT tag if the X3D browser is running as an embedded control within a World Wide Web browser, or through an application-specific mechanism such as a configuration file or system registry entry if the browser is running within some other containing application.

Support for browser options is not required but is strongly recommended. Some browsers may not support all available options, due to limitations in the underlying rendering system.

[Table 9.2](#) lists the available X3D Browser options.

**Table 9.2 — Browser options**

Name	Description	Type/valid range	Default
Antialiased	Render using hardware antialiasing if available	Boolean	False

Dashboard	Display browser navigation user interface	Boolean	Specified by bound NavigationInfo in content
EnableInlineViewpoints	Viewpoints from Inline nodes are included in list of viewpoints if made available by the Inline node.	Boolean	True
MotionBlur	Render animations with motion blur	Boolean	False
PrimitiveQuality	Render quality (tessellation level) for Box, Cone, Cylinder, Sphere	Low, Medium, High	Medium
QualityWhenMoving	Render quality while camera is moving	Low, Medium, High, Same (as while stationary)	Same
Shading	Specify shading mode for all objects	Wireframe, Flat, Gouraud, Phong	Gouraud
SplashScreen	Display browser splash screen on startup	Boolean	Implementation-dependent
TextureQuality	Quality of texture map display	Low, Medium, High	Medium

## 9.2.5 IMPORT statement

The IMPORT statement is used within an X3D file to specify nodes, which are defined within Inline files or programmatically created content, that are to be brought into the namespace of the containing file for the purposes of event routing. Once a node is imported, events may be sent to its fields via ROUTEs, or routed from any fields of the node which have output events. The IMPORT statement has the following components:

- a. The name of the Inline node that contains the node to be imported
- b. The name of the node to import
- c. An optional name to be used as an alias for the imported node within the run-time name scope, to help prevent name clashes within the parent scene containing the IMPORT statement.

The IMPORT statement has the following semantics:

- d. Once imported, events may be routed to or from the imported node in exactly the same manner as any node defined with DEF.
- e. Nodes imported into an X3D scene using the IMPORT statement may not be instanced via the USE statement.
- f. Only nodes that are exported from within the Inline via an EXPORT statement may be imported using a corresponding IMPORT statement.

The following example illustrates the use of the IMPORT statement (Classic VRML encoding syntax):

```
DEF I1 Inline {
  url "someurl.x3d"
}
. . .

IMPORT I1.rootTransform AS I1Root
DEF PI PositionInterpolator { ... }
ROUTE PI.value_changed TO I1Root.set_translation
```

In the above example, `rootTransform` is defined as a Transform node in the file `someurl.x3d` and exported via an EXPORT statement (see [4.4.6.3 EXPORT semantics](#)). The optional AS keyword defines an alias for `rootTransform` so that within the containing scene the node is referenced using the DEF name `I1Root`.

## 9.2.6 EXPORT statement

The EXPORT statement is used within an X3D file to specify nodes that may be imported into other scenes when Inlining that file. Only named nodes exported with an EXPORT statement are eligible to be imported into another file. The EXPORT statement has the following components:

- a. The DEF name of the node to be exported
- b. An optional name to be used as an alias for the exported node when importing it into other files

The EXPORT statement has the following semantics:

- c. Once imported into a containing scene, events may be routed to or from an exported node in exactly the same manner as any node defined with DEF.
- d. Exported nodes imported into a containing scene may not be instanced via the USE statement.
- e. Exportation may not be propagated across multiple files; that is, a node imported into one scene using the IMPORT statement may not then be further exported into another scene using the EXPORT statement.
- f. Nodes shall not be exported from the body of a PROTO declaration.

The following example illustrates the use of the EXPORT statement (Classic VRML encoding):

```
DEF T1 Transform {
  ...
}
. . .

EXPORT T1 AS rootTransform
```

In the above example, node `T1` is exported for use by other X3D scenes. The optional AS

keyword defines the exported name of `T1` as `rootTransform` (*i.e.*, other scenes may import the node only using the name `rootTransform`).

## 9.3 Abstract types

### 9.3.1 X3DNetworkSensorNode

```
X3DNetworkSensorNode : X3DSensorNode {
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFBool [out] isActive
}
```

This abstract node type is the basis for all sensors that generate events based on network activity.

### 9.3.2 X3DUrlObject

```
X3DUrlObject {
  SFString [in,out] description ""
  SFBool [in,out] load TRUE
  SFTime [in,out] refresh 0.0 [0,∞)
  MFString [in,out] url [] [URI]
}
```

This abstract interface is inherited by all nodes that contain data located on the World Wide Web, such as [AudioClip](#), [ImageTexture](#) and [Inline](#).

The *description* field specifies a textual description for the url asset. This information may be used by browser-specific user interfaces that wish to present users with more detailed information about the linked content.

The *load* field allows deferring when the Inline scene is read and displayed, in profiles that support that field. In profiles that do not support the *load* field, *url* content is loaded immediately.

The *refresh* field defines the interval in seconds that are necessary before an automatic reload of the current *url* asset is performed. If the preceding file loading fails or the *load* field is FALSE, no refresh is performed. If performed, a refresh attempts to reload the currently loaded entry of the url list. If a refresh fails to reload the currently loaded *url* entry, the browser retries the other entries in the *url* list.

**WARNING** Automatically reloading content can have security considerations and needs to be considered carefully.

All *url* fields can hold multiple string values. The strings in these fields indicate multiple locations to search for data in the order listed. If the browser cannot locate or interpret the data specified by the first location, it shall try the second and subsequent locations in order until a location containing interpretable data is encountered. X3D browsers only have to interpret a single string. If no interpretable locations are found, the node type defines the resultant default behaviour.

Each specified URL shall refer to a valid X3D file that contains a list of children nodes, prototypes and routes at the top level as described in [10.2.1 Grouping and children node types](#). The results are undefined if the URL refers to a file that is not **an X3D file, or if the X3D file contains an invalid scene.** a supported file type, or if the file contains

**invalid content.**

It shall be an error to specify a file in the URL field that has a set of component definitions that is not a subset of the components of the containing world. In addition, the components shall not be of a higher support level than those used by the containing world, either implicitly or through the PROFILE declaration or additional COMPONENT statements. When the world indicated by the *url* field requests capabilities greater than its parent, the following actions shall occur:

- an error shall be generated,
- the URL shall be treated as not interpretable as specified in [9.3.2 X3DUrlObject](#), and
- the next URL shall be loaded and checked in accordance with [9.2 Concepts](#).

For more information on URLs, see [9.2.1 URLs](#).

## 9.4 Node reference

### 9.4.1 Anchor

```
Anchor : X3DGroupingNode,X3DUrlObject {
  MFNode [in] addChildren
  MFNode [in] removeChildren
  SFBool [in out] bboxDisplay FALSE
  MFNode [in,out] children [] [X3DChildNode]
  SFString [in,out] description ""
  SFBool [in,out] load TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFString [in,out] parameter []
  SFTime [in,out] refresh 0.0 [0,∞)
  MFString [in,out] url [] [URI]
  SFBool [in out] visible TRUE
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}
```

The Anchor grouping node retrieves the content of a URL when the user activates (such as, clicks) some geometry contained within the Anchor node's children. If the URL points to a valid X3D file, that world replaces the world of which the Anchor node is a part (except when the *parameter* field, described below, alters this behaviour). If non-X3D data is retrieved, the browser shall determine how to handle that data; typically, it will be passed to an appropriate non-X3D browser.

Exactly how a user activates geometry contained by the Anchor node depends on the pointing device and is determined by the X3D browser. Typically, clicking with the pointing device will result in the new scene replacing the current scene. An Anchor node with an empty *url* does nothing when its children are chosen. A description of how multiple Anchors and pointing-device sensors are resolved on activation is contained in [20.2 Concepts](#).

More details on the *children*, *addChildren*, and *removeChildren* fields can be found in [10.2 Concepts](#).

The *description* field in the Anchor node specifies a textual description of the Anchor node. This **information** may be used by browser-specific user interfaces that wish to present users with more detailed information about the Anchor.

**The *load* and *refresh* fields have no effect.**



The *parameter* field may be used to supply any additional information to be interpreted by the browser. Each string shall consist of "keyword=value" pairs. For example, some browsers allow the specification of a "target" for a link to display a link in another part of an HTML document. The *parameter* field is then:

```
Anchor {
  parameter [ "target=name_of_frame" ];
  ...
}
```

An Anchor node may be used to bind the initial Viewpoint node in a world by specifying a URL ending with "#ViewpointName" where "ViewpointName" is the DEF name of a viewpoint defined in the X3D file.

#### EXAMPLE

```
Anchor {
  url "http://www.school.edu/X3D/someScene.wrl#OverView";
  children Shape { geometry Box {} };
}
```

specifies an anchor that loads the X3D file "someScene.wrl" and binds the initial user view to the Viewpoint node named "OverView" when the Anchor node's geometry (Box) is activated. If the named Viewpoint node is not found in the X3D file, the X3D file is loaded using the default Viewpoint node binding stack rules (see [23.3.5 Viewpoint](#)).

If the *url* field is specified in the form "#ViewpointName" (*i.e.*, no file name), the Viewpoint node with the given name ("ViewpointName") in the Anchor's run-time name scope(s) shall be bound (*set\_bind* TRUE). The results are undefined if there are multiple nodes derived from *X3DViewpointNode* with the same name in the Anchor's run-time name scope(s). The results are undefined if the Anchor node is not part of any run-time name scope or is part of more than one run-time name scope. See [4.4.7 Run-time name scope](#) for a description of run-time name scopes. See [23.3.5 Viewpoint](#), for the *X3DViewpointNode* transition rules that specify how browsers shall interpret the transition from the old node derived from *X3DViewpointNode* to the new one. For example:

```
Anchor {
  url "#Doorway";
  children Shape { geometry Sphere {} };
}
```

binds the viewer to the viewpoint defined by the "Doorway" viewpoint in the current world when the sphere is activated. In this case, if the node derived from *X3DViewpointNode* is not found, no action occurs on activation.

More details on the *url* field are contained in [9.2.1 URLs](#).

NOTE Viewpoint functionality is in addition to the X3DUrlObject interface characteristics.

The *bboxCenter* and *bboxSize* fields specify a bounding box that encloses the Anchor's children. This is a hint that may be used for optimization purposes. The results are undefined if the specified bounding box is smaller than the actual bounding box of the children at any time. The default *bboxSize* value, (-1, -1, -1), implies that the bounding box is not specified and if needed shall be calculated by the browser. More details on the *bboxCenter* and *bboxSize* fields can be found in [10.2.2 Bounding boxes](#).



## 9.4.2 Inline

```

Inline : X3DChildNode, X3DBoundedObject, X3DUrlObject {
  SFBool [in,out] bboxDisplay FALSE
  SFString [in,out] description ""
  SFBool [in,out] load TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTime [in,out] refresh 0.0 [0,∞)
  MFString [in,out] url [] [URI]
  SFBool [in,out] visible TRUE
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}

```

The Inline node embeds an X3D scene stored at a location on the World Wide Web into the current scene. Exactly when the Inline scene is read and displayed is defined by the value of the `load` field. The `load` field controls when the Inline scene is read and displayed, in profiles that support that field. In profiles that do not support the `load` field, exactly when the scene is read and displayed is not defined (such as, reading the scene may be delayed until the Inline node's bounding box is `visible` available to the viewer).

The run-time system can support any number of 3D model resource types as long as those follow the abstract model definition (see 2.[REC2077]), provide a registered content type (e.g. `model/x3d-xml`, `model/gltf-bin`, `model/stl`, etc.), and can be determined with some form of content negotiation (see 2.[RFC2616]). The run-time system must support at least one X3D type (e.g. `model/x3d-xml`) but can also support and negotiate any number of X3D encodings and (optionally) non-X3D representation formats. Support for loading glTF assets also requires support for Shape component level 3.

Once the Inline scene is loaded, its children are added to the current scene and are treated as children of the Inline for rendering and interaction; however the children are not exposed to the current scene for routing and DEF name access unless their names have been explicitly imported into the scene using the IMPORT statement (see 4.4.6.2 [IMPORT semantics](#)).

**NOTE** When Inline is used to load a child scene, processing of the Inline content is as specified in the respective PROFILE, COMPONENT, UNIT, IMPORT, and EXPORT statements.

The `visible` field specifies whether or not the content within a node is visually displayed. The value of this field has no effect on animation behaviors, collision behaviors, event passing, or other non-visual characteristics.

If the `load` field is set to `TRUE` (the default field value), the X3D file specified by the `url` field is loaded immediately. If the `load` field is set to `FALSE`, no action is taken. It is possible to explicitly load the URL at a later time by sending a `TRUE` event to the `load` field (such as, the result of a ProximitySensor or other sensor firing an event). If a `FALSE` event is sent to the `load` field of a previously loaded Inline, the contents of the Inline will be unloaded from the scene graph.

An event sent to `url` can be used to change the scene that is inlined by the Inline node. If this value is set after the Inline is already loaded, its contents will be unloaded and the scene to which the new URL points will be loaded.

The user is able to specify a bounding box for the Inline node using the `bboxCenter` and `bboxSize` fields. This is a hint to the browser and `could` may be used for optimization purposes such as culling.

Security precaution: it is an error for a model to Inline itself, directly or indirectly, in order to avoid nonterminating recursion loops. X3D players SHALL NOT honor self-referential loading of model loops in order to avoid security vulnerabilities.

### 9.4.3 LoadSensor

```
LoadSensor : X3DNetworkSensorNode {
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTime [in,out] timeOut 0
  MFNode [in,out] watchList [] [X3DUrlObject]
  SFBool [out] isActive
  SFBool [out] isLoaded
  SFTime [out] loadTime
  SFFloat [out] progress
}
```

The `LoadSensor` monitors the progress and success of downloading URL elements over a network. Only nodes that contain a valid URL field (*i.e.*, descendants of `X3DUrlObject`), may be specified in the `watchList` field. Multiple nodes may be watched with a single `LoadSensor`.

The `timeOut` field specifies the maximum time for which the `LoadSensor` will monitor loading, starting from when the sensor becomes active. A value of 0 for the `timeOut` field indicates an indefinite time out period; *i.e.*, the `LoadSensor` will wait until loading has completed either with success or failure.

The `watchList` field contains one or more URL objects to monitor. Only nodes that contain a valid URL field (*i.e.*, descendants of `X3DUrlObject`), may be specified as elements of `watchList`. If multiple values are specified for this field, output events are generated only when all of the children have loaded or at least one has failed. If individual load status information is desired for different nodes, multiple `LoadSensor` nodes may be used, each with a single `watchList` element.

If an Anchor node is part of a `watchList` field value, `isLoaded` reports success for this node as follows. There are three cases that Anchor node can handle:

1. binding to a Viewpoint node in the current scene,
2. loading a replacement world or file asset, and
3. launching a separate window for a file asset.

When binding to a viewpoint (item a above), the asset is loaded when the Viewpoint is bound. When loading a replacement world or asset (item b above), no action is taken because the current world is lost. When launching a separate window or asset (item c above), the load is considered complete when the operating system or web browser acknowledges the load request.

The `isActive` field generates events when loading of the `LoadSensor`'s `watchList` elements begins and ends. An `isActive TRUE` event is generated when the first element begins loading. An `isActive FALSE` event is generated when loading has completed, either with a successful load of all objects or a failed load of one of the objects, or when the timeout period is reached as specified in the `timeout` field.

The `isLoaded` field generates events when loading of the `LoadSensor`'s `watchList` has completed. An `isLoaded TRUE` event is generated when all of the elements have been loaded. An `isLoaded FALSE` event is generated when one or more of the elements has

failed to load, or when the timeout period is reached as specified in the *timeout* field. If all elements in the watchlist are already loaded by the time the LoadSensor is processed, the LoadSensor shall generate an *isLoaded* event with value `TRUE` and a *progress* event with value 1 at the next event cascade.

The *loadTime* event is generated when loading of the LoadSensor's *watchList* has successfully completed. If loading fails or the timeout period is reached, a *loadTime* event is not generated.

The *progress* field generates events as loading progresses. The value of *progress* is a floating-point number between 0 and 1 inclusive. A value of 1 indicates complete loading of all *watchList* elements. The exact meaning of all other values (*i.e.*, whether these indicate a percentage of total bytes, a percentage of total number of files, or some other measurement) and the frequency with which *progress* events are generated are browser-dependent. Regardless, the browser shall in all cases guarantee that a *progress* value of 1 is generated upon successful load of all URL objects.

The following example defines a LoadSensor that monitors the progress of loading two different ImageTexture nodes:

```
Shape {
  appearance Appearance {
    material Material {
      texture DEF TEX1 ImageTexture { url "Ampic.png" }
    }
  }
  geometry Sphere {}
}
Shape {
  appearance Appearance {
    material Material {
      texture DEF TEX2 ImageTexture { url "Bmpic.png" }
    }
  }
  geometry Sphere {}
}
DEF LS LoadSensor {
  watchList [ USE TEX1, USE TEX2 ]
}
ROUTE LS.loadTime TO MYSCRIPT.loadTime
```

The events this would generate are:

- Success of all children:
  - *isLoaded* = true
  - *loadTime* = now
  - *progress* = 1
  - *isActive* = false
- Timeout of any children, failure of any children, or no network present:
  - *isLoaded* = false
  - *isActive* = false

For *watchList* elements that allow dynamic reloading of their contents, any reload of that element (EXAMPLE changing the *url* field of an ImageTexture or setting the *load* field of an Inline), resets the LoadSensor so that it monitors those elements based on the new values and resets its *timeout* period if one was specified.

For streamed media types, the first frame of data available means successful load of the URL object (*i.e.*, the browser can render one frame of a movie or start playing an audio file).

## 9.5 Support levels

The Networking component provides three levels of support as specified in [Table 9.3](#).

**Table 9.3 — Networking component support levels**

Level	Prerequisites	Nodes/Features	Support
<b>1</b>	Core 1		
		<i>X3DUrlObject</i> (Abstract)	n/a
		<i>X3DNetworkSensorNode</i> (Abstract)	n/a
		Protocols	file: protocol only.
		Name resolution	Fully-specified URLs.
<b>2</b>	Core 1 Grouping 1		
		Level 1 supported nodes	Support as specified for Level 1.
		Anchor	All fields fully supported.
		Inline	All fields except <i>load</i> which is optionally supported.
		Protocols	file: <b>and</b> http: <b>and</b> https protocols are required.
		Name resolution	Relative URLs; URNs.
<b>3</b>	Core 1 Grouping 1		
		Level 2 supported nodes	Support as specified for Level 2.
		Inline	All fields fully supported.
		LoadSensor	All fields fully supported.
		Statements: IMPORT EXPORT	Full support.
		Browser options	Implementation-dependent.

4	Core 1 Grouping 1		
		Level 3 supported nodes	Support as specified for Level 3.
		Protocols	https: protocol is required.
		Communication security	HTTP and HTTPS username/password is required.
		Model support	Support for .gltf (model/gltf+json) and .bin (application/octet-stream)





# Extensible 3D (X3D)

## Part 1: Architecture and base components

### 30 Event Utilities component

---



#### 30.1 Introduction

##### 30.1.1 Name

The name of this component is "EventUtilities". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 30.1.2 Overview

This clause describes the Event Utilities component of this part of ISO/IEC 19775. This includes Trigger and Sequencer node types that gives authors the capability to gate, convert, or sequence numerous event-types for common interactive applications without the use of a [Script](#) node. [Table 30.1](#) provides links to the major topics in this subclause.

**Table 30.1 — Topics**

- [30.1 Introduction](#)
  - [30.1.1 Name](#)
  - [30.1.2 Overview](#)
- [30.2 Concepts](#)
  - [30.2.1 Overview of event utility nodes](#)
  - [30.2.2 Mutating events of Single Field \(SF\) event types](#)
  - [30.2.3 Triggering events between Single Field \(SF\) event types](#)
  - [30.2.4 Sequencing Single Field \(SF\) events](#)
- [30.3 Abstract types](#)
  - [30.3.1 X3DSequencerNode](#)
  - [30.3.2 X3DTriggerNode](#)
- [30.4 Node reference](#)
  - [30.4.1 BooleanFilter](#)
  - [30.4.2 BooleanSequencer](#)
  - [30.4.3 BooleanToggle](#)
  - [30.4.4 BooleanTrigger](#)

- [30.4.5 IntegerSequencer](#)
- [30.4.6 IntegerTrigger](#)
- [30.4.7 TimeTrigger](#)
- [30.5 Support levels](#)
  
- [Table 30.1 — Topics](#)
- [Table 30.2 — Event utilities component support levels](#)

## 30.2 Concepts

### 30.2.1 Overview of event utility nodes

The Event Utilities component consists of 3 basic concepts:

- a. mutating events of Single Field (SF) events of a given type,
- b. triggering Single Field (SF) events of a given type from events of other types, and
- c. sequencing Single Field (SF) events along a timeline (as a discrete value generator).

These nodes may be composed using ROUTEs to create powerful authoring scenarios without writing script code. This is especially useful in profiles where interactivity would be otherwise significantly limited due to lack of a [Script](#) node.

The location of event utility nodes in the transformation hierarchy has no effect on their operation. For example, if a parent of a [BooleanSequencer](#) is a [Switch](#) node with whichChoice set to  $-1$  (*i.e.*, ignore its children), the BooleanSequencer continues to operate as specified (*i.e.*, receives and sends events).

### 30.2.2 Mutating events of Single Field (SF) event types

Mutator nodes allow content authors to alter values of a given type. In this part of ISO/IEC 19775, the [BooleanFilter](#) node accepts a single Boolean input event and generates either a `TRUE` or `FALSE` output event based on the value of its input; it also generates an event equal to the negation of its input. These events allow for the creation of conditional behaviors that would otherwise require a script.

### 30.2.3 Triggering events between Single Field (SF) event-types

Trigger nodes that generate an output event of a given type based on an input event of a different type are all derived from the [X3DTriggerNode](#) abstract node type. This part of ISO/IEC 19775 specifies the following types of [X3DTriggerNode](#) nodes:

- a. [BooleanTrigger](#)
- b. [IntegerTrigger](#)
- c. [TimeTrigger](#)

The BooleanTrigger node generates a Boolean output event upon receiving a time input

event.

The IntegerTrigger node generates an integer output event upon receiving a Boolean input event. The value of the integer can be specified.

The TimeTrigger node generates a time output event upon receiving a Boolean input event.

EXAMPLE Routing the *isActive* field of a [TouchSensor](#) to the TimeTrigger allows the content creator to start a [TimeSensor](#) when the *isActive* field generates an event.

## 30.2.4 Sequencing Single Field (SF) events

Sequencer nodes allow content authors to generate a specific sequence of discrete events over the course of a [TimeSensor](#)'s output. They are derived from the abstract node type [X3DSequencerNode](#) and thus share the signature fields of *set\_fraction* (SFFloat [in]) and *key* (MFFloat [in,out]).

The *set\_fraction* inputOnly field receives an SFFloat event and causes the sequencing function to evaluate, resulting in a *value\_changed* output event with the same timestamp as the *set\_fraction* event. The sequencer node sends only one *value\_changed* output event per *key*[*i*] interval. The usage of the *keyValue* and output fields are dependent on the type of the Sequencer.

[BooleanSequencer](#) and [IntegerSequencer](#) output a single-value field to *value\_changed*. Each value in the *keyValue* field corresponds in order to the parameter value in the *key* field. Results are undefined if the number of values in the *key* field of a sequencer is not the same as the number of values in the *keyValue* field.

The specified X3D sequencer nodes are designed for discrete events along a timeline. Each of these nodes defines a piecewise-linear function,  $f(t)$ , on the interval  $(-\infty, +\infty)$ . The piecewise-linear function is defined by  $n$  values of  $t$ , called *key*, and the  $n$  corresponding values of  $f(t)$ , called *keyValue*. The keys shall be monotonically non-decreasing, otherwise the results are undefined. The keys are not restricted to any interval.

Each of these nodes evaluates  $f(t)$  given any value of  $t$  (via the *fraction* field) as follows: Let the  $n$  keys  $t_0, t_1, t_2, \dots, t_{n-1}$  partition the domain  $(-\infty, +\infty)$  into the  $n+1$  subintervals given by  $(-\infty, t_0), [t_0, t_1), [t_1, t_2), \dots, [t_{n-1}, +\infty)$ . Also, let the  $n$  values  $v_0, v_1, v_2, \dots, v_{n-1}$  be the values of  $f(t)$  at the associated key values. The discrete value sequencing function,  $f(t)$ , is defined to be:

$$\begin{aligned} f(t) &= v_n, \text{ if } t_n \leq t < t_{n-1} \\ &= v_0, \text{ if } t \leq t_0, \\ &= v_{n-1}, \text{ if } t \geq t_{n-1} \end{aligned}$$

## 30.3 Abstract types

### 30.3.1 X3DSequencerNode



```

X3DSequencerNode : X3DChildNode {
  SFBool [in] next
  SFBool [in] previous
  SFFloat [in] set_fraction (-∞,∞)
  MFFloat [in,out] key [] (-∞,∞)
  MF<type> [in,out] keyValue []
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  [S|M]F<type> [out] value_changed
}

```

This abstract node type is the base node type from which all Sequencers are derived. [30.2.4 Sequencing Single Field \(SF\) events](#) contains a detailed discussion of Sequencer nodes.

Receipt of a *next* event with value `TRUE` triggers the next output value in *keyValue* array by issuing a *value\_changed* event with that value. Receipt of a *previous* event with value `TRUE` triggers previous output value in *keyValue* array. Sending a `FALSE` event to the *next* or *previous* fields has no effect. These trigger events "wrap around" after reaching the boundary of *keyValue* array; *i.e.*, *next* goes to initial element after last, and *previous* goes to last element after first.

### 30.3.2 X3DTriggerNode

```

X3DTriggerNode : X3DChildNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}

```

This abstract node type is the base node type from which all trigger nodes are derived. [30.2.3 Triggering events between Single Field \(SF\) event-types](#) contains a detailed discussion of Triggers.

## 30.4 Node Reference

### 30.4.1 BooleanFilter

```

BooleanFilter : X3DChildNode {
  SFBool [in] set_boolean
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFBool [out] inputFalse
  SFBool [out] inputNegate
  SFBool [out] inputTrue
}

```

BooleanFilter filters Boolean events, allowing for selective routing of `TRUE` or `FALSE` values and negation.

When the *set\_boolean* event is received, the BooleanFilter node generates two events: either *inputTrue* or *inputFalse*, based on the Boolean value received; and *inputNegate*, which contains the negation of the value received.

### 30.4.2 BooleanSequencer

```

BooleanSequencer : X3DSequencerNode {
  SFBool [in] next
  SFBool [in] previous
  SFFloat [in] set_fraction
  MFFloat [in,out] key [] (-∞,∞)
  MFBool [in,out] keyValue []
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFBool [out] value_changed
}

```

BooleanSequencer generates sequential *value\_changed* events selected from the

*keyValue* field when driven from a [TimeSensor](#) clock. Among other actions, it can enable/disable lights and sensors, or bind/unbind viewpoints and other [X3DBindableNode](#) nodes using *set\_bind* events.

The *keyValue* field is made up of a list of `FALSE` and `TRUE` values.

A BooleanSequencer shall be instantiated for every node enabled or bound.

### 30.4.3 BooleanToggle

```
BooleanToggle : X3DChildNode {
  SFBool [in]  set_boolean
  SFNode [in,out] metadata  NULL [X3DMetadataObject]
  SFBool [in,out] toggle  FALSE
}
```

BooleanToggle stores a Boolean value for toggling on/off. When a *set\_boolean* `TRUE` event is received, the BooleanToggle negates the value of the *toggle* field and generates the corresponding *toggle* field output event. *set\_boolean* `FALSE` events are ignored.

The BooleanToggle can be reset to a specific state by directly setting the value of the inputOutput *toggle* field.

### 30.4.4 BooleanTrigger

```
BooleanTrigger : X3DTriggerNode {
  SFTime [in]  set_triggerTime
  SFNode [in,out] metadata  NULL [X3DMetadataObject]
  SFBool [out] triggerTrue
}
```

BooleanTrigger is a trigger node that generates Boolean events upon receiving time events.

The *triggerTrue* event is generated when the BooleanTrigger receives a *set\_triggerTime* event. The value of *triggerTrue* shall always be `TRUE`.

### 30.4.5 IntegerSequencer

```
IntegerSequencer : X3DSequencerNode {
  SFBool [in]  next
  SFBool [in]  previous
  SFFloat [in]  set_fraction
  MFFloat [in,out] key  [] (-∞,∞)
  MFInt32 [in,out] keyValue  [] (-∞,∞)
  SFNode [in,out] metadata  NULL [X3DMetadataObject]
  SFInt32 [out]  value_changed
}
```

The IntegerSequencer node generates sequential discrete *value\_changed* events selected from the *keyValue* field in response to each *set\_fraction*, *next*, or *previous* event.

### 30.4.6 IntegerTrigger

```
IntegerTrigger : X3DTriggerNode {
  SFBool [in]  set_boolean
  SFInt32 [in,out] integerKey  -1 (-∞,∞)
  SFNode [in,out] metadata  NULL [X3DMetadataObject]
  SFInt32 [out]  triggerValue
}
```

IntegerTrigger handles single field Boolean events to set an integer value for the output event. This is useful for connecting environmental events to the [Switch](#) node's *whichChoice* field.

Upon receiving a *set\_boolean* event, the IntegerTrigger node will generate a *triggerValue* event with the current value of *integerKey*. The value of *set\_boolean* shall be ignored.

### 30.4.7 TimeTrigger

```
TimeTrigger : X3DTriggerNode {
  SFBool [in]  set_boolean
  SFNode [in,out] metadata  NULL [X3DMetadataObject]
  SFTime [out] triggerTime
}
```

TimeTrigger is a trigger node that generates time events upon receiving Boolean events.

The *triggerTime* event is generated when the TimeTrigger receives a *set\_boolean* event. The value of *triggerTime* shall be the time at which *set\_boolean* is received. The value of *set\_boolean* shall be ignored.

## 30.5 Support levels

The Event Utilities component provides one level of support as specified in [Table 30.2](#). Level 1 provides the full support for all nodes defined above.

**Table 30.2 — Event utilities component support levels**

Level	Prerequisites	Nodes/Features	Support
1	Core 1 Grouping 1		
		<i>X3DSequencerNode</i> (abstract)	All fields fully supported.
		<i>X3DTriggerNode</i> (abstract)	All fields fully supported.
		BooleanFilter	All fields fully supported.
		BooleanSequencer	All fields fully supported.
		BooleanToggle	All fields fully supported.
		BooleanTrigger	All fields fully supported.
			All fields fully

		IntegerSequencer	supported.
		IntegerTrigger	All fields fully supported.
		TimeTrigger	All fields fully supported.



Draft



## Extensible 3D (X3D) Part 1: Architecture and base components

### Annex I (normative)

## OpenGL shading language (GLSL) binding

---



### I.1 General

This annex defines the mapping of concepts of the programmable shaders component to the OpenGL Shading Language (GLSL) (see [\[GLSL\]](#)). It applies to a ComposedShader node that sets the *language* field to "GLSL".

### I.2 Topics

[Table I.1](#) provides links to the major topics in this annex.

**Table I.1 — Topics**

- [I.1 General](#)
- [I.2 Topics](#)
- [I.3 Interaction with Other Nodes and Components](#)
  - [I.3.1 Vertex Shader](#)
  - [I.3.2 Fragment Shader](#)
  - [I.3.3 LoadSensor](#)
  - [I.3.4 VertexAttributes](#)
- [I.4 Data Type Mapping](#)
  - [I.4.1 Node fields](#)
  - [I.4.2 X3D Field types to OpenGL Data Types](#)
- [I.5 Event Model](#)
  - [I.5.1 Changing URL fields](#)
  - [I.5.2 Changing the \*object\* field](#)
  - [I.5.3 Changing the \*attrib\* field](#)
  - [I.5.4 Relinking Programs](#)
- [Table I.1 — Topics](#)

- [Table I.2 — Mapping of X3D texture node types to GLSL sampler types](#)
- [Table I.3 — Mapping of X3D Field type to GLSL data type](#)

## I.3 Interaction with other nodes and components

### I.3.1 Vertex shader

The vertex shader replaces the fixed functionality of the vertex processor. The GLSL specification (see [\[GLSL\]](#)) states that the following functionality is disabled if a vertex shader is supplied:

- a. The model view matrix is not applied to vertex coordinates.
- b. The projection matrix is not applied to vertex coordinates.
- c. The texture matrices are not applied to texture coordinates.
- d. The normals are not transformed to eye coordinates.
- e. The normals are not rescaled or normalized.
- f. Texture coordinates are not generated automatically.
- g. Per-vertex lighting is not performed.
- h. Color material lighting is not performed.
- i. Point size distance attenuation is not performed.

### I.3.2 Fragment shader

The fragment shader replaces the fixed functionality of the fragment processor. The GLSL specification (see [\[GLSL\]](#)) states that the following functionality is disabled if a fragment shader is supplied:

- a. Textures are not applied.
- b. Fog is not applied.

### I.3.3 LoadSensor

The LoadSensor node (See [9.4.3 LoadSensor](#)) has two output fields *isActive* and *isLoaded*. The *isLoaded* field behaviour is unchanged.

The *isActive* field is defined to issue a `TRUE` event when all the following conditions have been satisfied:

- a. the content identified by the *url* field has been successfully loaded;
- b. a valid OpenGL program object handle has been created for the shader object (`GLhandleARB` in OpenGL 1.5 and `uint` in OpenGL 2.0);
- c. the shader source has been set without error; and
- d. the shader has been successfully compiled, without error.

The LoadSensor node does not have any interaction with the process of linking multiple shader objects into a complete shader program.

### I.3.4 Vertex attributes

Each vertex attribute node directly maps the *name* field to the uniform variable of the same name. If the name is not available as a uniform variable in the provided shader source, the values of the node shall be ignored.

The browser implementation shall automatically assign appropriate internal index values for each attribute.

## I.4 Data type mapping

### I.4.1 Node fields

Fields that are of type SFNode/MFNode are ignored unless the value is of type *X3DTextureNode*. Field instances of type *X3DTextureNode* are mapped according to the appropriate sampler data type. The texture types are mapped as defined in [Table I.2](#).

**Table I.2 — Mapping of X3D texture node types to GLSL sampler types**

X3D texture type	GLSL variable type
<i>X3DTexture2DNode</i>	sampler2D.
<i>X3DTexture3DNode</i>	sampler3D.
<i>X3DEnvironmentTextureNode</i>	samplerCube.

X3D does not define mappings to the GLSL types sampler1D, sampler1DShadow and sampler2DShadow.

### I.4.2 X3D Field types to GLSL data types

[Table I.3](#) indicates how the X3D field types shall be mapped to data types used in GLSL.

**Table I.3 — Mapping of X3D field type to GLSL data type**

X3D field type	GLSL variable type
SFBool	bool
MFBool	bool[]
MFInt32	int[]
SFInt32	int
SFFloat	float
MFFloat	float[]

SFDouble	float
MFDouble	float[]
SFTime	float
MFTime	float[]
SFNode	See <a href="#">4.1 Node fields</a>
MFNode	See <a href="#">4.1 Node fields</a>
SFVec2f	vec2
MFVec2f	vec2[]
SFVec3f	vec3
MFVec3f	vec3[]
SFVec4f	vec4
MFVec4f	vec4[]
SFVec3d	float3
MFVec3d	float3[]
SFVec4d	float4
MFVec4d	float4[]
SFRotation	vec4
MFRotation	vec4[]
MFCColor	vec4[]
SFCColor	vec4
SFImage	int[]
MFImage	int[]
SFString	Not supported
MFString	Not supported
SFMatrix3f	mat3
MFMatrix3f	mat3[]
SFMatrix4f	mat4



MFMatrix4f

mat4[]

OpenGL defines maximum supported lengths of each array data type, which may conflict with the minimum support requirements for X3D. OpenGL will automatically convert double-precision data types to single precision types.

## 1.5 Event model

### 1.5.1 Changing URL fields

When the *url* receives an event changing the value, the browser shall immediately attempt to download the new source. Upon successful download, the browser shall attempt to compile the new source and issue the appropriate LoadSensor events. It shall not automatically relink the shader program, nor disable the currently running shader. This follows the semantics of the OpenGL API requirements for separate register-compile-link steps.

Values defined at load time of the file do not require an explicit request to relink. It shall be assumed to automatically link once all the objects have successfully downloaded. If some of the shader source files are not downloaded or compiled (e.g., due to errors), no linking will occur for the shader program.

### 1.5.2 Changing the *object* field

If at any time after the initial load, the user changes the values of the *object* field, the user shall need to request an explicit relink of the containing shader program. The containing ComposedShader shall not automatically relink, nor should it automatically disable the current shader.

### 1.5.3 Changing the *attrib* field

Per-vertex attributes may be defined as one of the fields of *X3DComposedGeometryNode*. These may be changed at runtime by adding or removing node instances. Adding new node instances to the field shall require that the user request an explicit relink in order to make them visible to the shader.

### 1.5.4 Relinking Programs

The user may, at any time, request that OpenGL re-link the composing shader objects by sending a `TRUE` value to the *activate* inputOnly field of the ComposedShader node. Users may need to force a relink of the ComposedShader under various circumstances, such as changing the *url* field of one or more ShaderPart nodes, or adding or removing ShaderPart nodes. Relinking the shader shall replace the existing shader with the new executable.





## Extensible 3D (X3D) Part 1: Architecture and base components

### 10 Grouping component



#### 10.1 Introduction

##### 10.1.1 Name

The name of this component is "Grouping". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 10.1.2 Overview

This clause describes the Grouping component of this part of ISO/IEC 19775. This includes how nodes are organized into groups to establish a transformation hierarchy for the X3D scene graph. [Table 10.1](#) provides links to the major topics in this clause.

**Table 10.1 — Topics**

- [10.1 Introduction](#)
  - [10.1.1 Name](#)
  - [10.1.2 Overview](#)
- [10.2 Concepts](#)
  - [10.2.1 Grouping and children node types](#)
  - [10.2.2 Bounding boxes](#)
- [10.3 Abstract types](#)
  - [10.3.1 X3DBoundedObject](#)
  - [10.3.2 X3DGroupingNode](#)
- [10.4 Node reference](#)
  - [10.4.1 Group](#)
  - [10.4.2 StaticGroup](#)
  - [10.4.3 Switch](#)
  - [10.4.4 Transform](#)
- [10.5 Support levels](#)
- [Table 10.1 — Topics](#)
- [Table 10.2 — Grouping component support levels](#)

## 10.2 Concepts

### 10.2.1 Grouping and children node types

Grouping nodes have a field that contains a list of children nodes. Each grouping node defines a coordinate space for its children. This coordinate space is relative to the coordinate space of the node of which the group node is a child. Such a node is called a *parent* node. This means that transformations accumulate down the scene graph hierarchy.

This part of ISO/IEC 19775 defines several grouping nodes, including the following:

- [Anchor](#)
- [Billboard](#)
- [Collision](#)
- [Group](#)
- [LOD](#)
- [Switch](#)
- [Transform](#)

Components may add the following:

- new grouping node types,
- new node types that may be used as children, and
- node types that may not be used as children.

All grouping nodes have *addChildren* and *removeChildren* inputOnly fields. The *addChildren* event appends nodes to the *children* field of a grouping node. Any nodes passed to the *addChildren* inputOnly field that are already in the children list of the grouping node are ignored. For example, if the *children* field contains the nodes Q, L and S (in order) and the group receives an *addChildren* event containing (in order) nodes A, L, and Z, the result is a *children* field containing (in order) nodes Q, L, S, A, and Z.

The *removeChildren* event removes nodes from the *children* field of the grouping node. Any nodes in the *removeChildren* event that are not in the *children* list of the grouping node are ignored. For example, if the *children* field contains the nodes Q, L, S, A and Z and it receives a *removeChildren* event containing nodes A, L, and Z, the result is Q, S.

Note that a variety of node types reference other node types through fields. Some of these are parent-child relationships, while others are not (there are node-specific semantics).

All grouping nodes shall have a *children* field of type MFNode. Adding a node to this field will add that node to the grouping node's set of children. A *children* field is not allowed to directly contain multiple instances of the same node. A *children* field is not allowed to contain nodes that are ancestors of the grouping node.

A variety of node types reference other node types through fields. Some of these are parent-child relationships (e.g., the children field of the Transform node) while others are not (e.g., the appearance field of the Shape node). The field type specifies which type of node may be placed in them. For instance, the node type of the children field of the Transform node is MFNode where all nodes shall be derived from *X3DChildNode*. Therefore, only node types derived from *X3DChildNode* may be placed there. Shape is legal in the children field because it is derived from *X3DChildNode*, while Appearance is not. See [Figure 4.2](#) for a complete derivation hierarchy.

New nodes types may be defined using the extension mechanisms. These new node types can be placed in a node field as long as the node field's type is in the new type's derivation hierarchy.

## 10.2.2 Bounding boxes

Several node types include a bounding box specification comprised of two fields, *bboxSize* and *bboxCenter*. A bounding box is a rectangular parallelepiped of dimension *bboxSize* centred on the location *bboxCenter* in the local coordinate system. This is typically used by grouping nodes to provide a hint to the browser on the group's approximate size for culling optimizations. The default size for bounding boxes  $(-1, -1, -1)$  indicates that the user did not specify the bounding box and the effect shall be as if the bounding box were infinitely large. A *bboxSize* value of  $(0, 0, 0)$  is valid and represents a point in space (i.e., an infinitely small box). Specified *bboxSize* field values shall be  $\geq 0.0$  or equal to  $(-1, -1, -1)$ . The *bboxCenter* fields specify a position offset from the local coordinate system.

The *bboxCenter* and *bboxSize* fields may be used to specify a maximum possible bounding box for the objects inside a grouping node (EXAMPLE Transform). These are used as hints to optimize certain operations such as determining whether or not the group needs to be drawn. The bounding box shall be large enough at all times to enclose the union of the group's children's bounding boxes; it shall not include any transformations performed by the group itself (i.e., the bounding box is defined in the local coordinate system of the children). Results are undefined if the specified bounding box is smaller than the true bounding box of the group.

## 10.3 Abstract types

### 10.3.1 X3DBoundedObject

```
X3DBoundedObject {
  SFBool [in out] bboxDislay FALSE
  SFBool [in out] visible TRUE
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}
```

This abstract **node type interface** is the basis for all node types that have bounds specified as part of the definition.

The *bboxCenter* and *bboxSize* fields specify a bounding box that encloses the grouping node's children. This is a hint that may be used for optimization purposes. The results are undefined if the specified bounding box is smaller than the actual bounding box of the children at any time. A default *bboxSize* value,  $(-1, -1, -1)$ , implies that the

bounding box is not specified and, if needed, is calculated by the browser. A description of the *bboxCenter* and *bboxSize* fields is contained in [10.2.2 Bounding boxes](#).

When *bboxDisplay* is true, the bounding box is displayed for the associated geometry so that both are aligned with world coordinates. The bounding box is displayed regardless of whether contained content is visible.

The *visible* field specifies whether or not the content within a node is visually displayed. The value of this field has no effect on animation behaviors, collision behaviors, event passing, or other non-visual characteristics.

### 10.3.2 X3DGroupingNode

```
X3DGroupingNode : X3DChildNode, X3DBoundedObject {
  MFNode [in]  addChildren      [X3DChildNode]
  MFNode [in]  removeChildren   [X3DChildNode]
  MFNode [in,out] children      [] [X3DChildNode]
  SFBool [in,out] bboxDisplay   FALSE
  SFNode [in,out] metadata      NULL [X3DMetadataObject]
  SFBool [in,out] visible       TRUE
  SFVec3f []   bboxCenter      0 0 0 (-∞,∞)
  SFVec3f []   bboxSize        -1 -1 -1 [0,∞) or -1 -1 -1
}
```

This abstract node type indicates that concrete node types derived from it contain children nodes and is the basis for all aggregation.

More details on the *children*, *addChildren*, and *removeChildren* fields can be found in [10.2.1 Grouping and children node types](#).

## 10.4 Node reference

### 10.4.1 Group

```
Group : X3DGroupingNode {
  MFNode [in]  addChildren      [X3DChildNode]
  MFNode [in]  removeChildren   [X3DChildNode]
  MFNode [in,out] children      [] [X3DChildNode]
  SFBool [in,out] bboxDisplay   FALSE
  SFNode [in,out] metadata      NULL [X3DMetadataObject]
  SFBool [in,out] visible       TRUE
  SFVec3f []   bboxCenter      0 0 0 (-∞,∞)
  SFVec3f []   bboxSize        -1 -1 -1 [0,∞) or -1 -1 -1
}
```

A Group node contains children nodes without introducing a new transformation. It is equivalent to a [Transform](#) node containing an identity transform.

More details on the *children*, *addChildren*, and *removeChildren* fields can be found in [10.2.1 Grouping and children node types](#).

The *bboxCenter* and *bboxSize* fields specify a bounding box that encloses the Group node's children. This is a hint that may be used for optimization purposes. The results are undefined if the specified bounding box is smaller than the actual bounding box of the children at any time. A default *bboxSize* value, (-1, -1, -1), implies that the bounding box is not specified and, if needed, is calculated by the browser. A description of the *bboxCenter* and *bboxSize* fields is contained in [10.2.2 Bounding boxes](#).

### 10.4.2 StaticGroup

```
StaticGroup : X3DChildNode, X3DBoundedObject {
```

```

SFBool [in out] bboxDisplay FALSE
SFNode [in.out] metadata NULL [X3DMetadataObject]
SFBool [in out] visible TRUE
MFNode [] children [] [X3DChildNode]
SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}

```

The `StaticGroup` node contains children nodes which cannot be modified. `StaticGroup` children are guaranteed to not change, send events, receive events or contain any USE references outside the `StaticGroup`. This allows the browser to optimize this content for faster rendering and less memory usage.

A browser shall prevent all illegal attempts to modify the `StaticGroup` and its children. Children of the `StaticGroup` are guaranteed not to generate events.

Implementations are free to rearrange or remove nodes inside a `StaticGroup` as long as the final rendering is the same. These optimizations might include flattening a series of transformations into one transform, performing appearance bundling or heavy analysis of the scene graph for maximal rendering speed. A `StaticGroup` does not need to maintain its children's X3D representations (such as field data), as they cannot be accessed after creation time.

The *visible* field specifies whether or not the content within a node is visually displayed. The value of this field has no effect on animation behaviors, collision behaviors, event passing, or other non-visual characteristics.

### 10.4.3 Switch

```

Switch : X3DGroupingNode {
  MFNode [in] addChildren [X3DChildNode]
  MFNode [in] removeChildren [X3DChildNode]
  MFNode [in.out] children [] [X3DChildNode]
  SFBool [in out] bboxDisplay FALSE
  SFNode [in.out] metadata NULL [X3DMetadataObject]
  SFBool [in out] visible TRUE
  SFInt32 [in,out] whichChoice -1 [-1,∞)
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}

```

The `Switch` grouping node traverses zero or one of the nodes specified in the *children* field.

[10.2.1 Grouping and children node types](#), describes details on the types of nodes that are legal values for *children*.

The *whichChoice* field specifies the index of the child to traverse, with the first child having index 0. If *whichChoice* is less than zero or greater than the number of nodes in the *children* field, nothing is chosen.

All nodes under a `Switch` continue to receive and send events regardless of the value of *whichChoice*. For example, if an active `TimeSensor` is contained within an inactive choice of an `Switch`, the `TimeSensor` sends events regardless of the `Switch`'s state.

The *bboxCenter* and *bboxSize* fields specify a bounding box that encloses the `Switch` node's children. This is a hint that may be used for optimization purposes. The results are undefined if the specified bounding box is smaller than the actual bounding box of the child with the largest bounding box at any time. A default *bboxSize* value, (-1, -1, -1), implies that the bounding box is not specified and, if needed, is calculated by the



browser. A description of the *bboxCenter* and *bboxSize* fields is contained in [10.2.2 Bounding boxes](#).

## 10.4.4 Transform

```

Transform : X3DGroupingNode {
  MFNode [in] addChildren [X3DChildNode]
  MFNode [in] removeChildren [X3DChildNode]
  SFVec3f [in,out] center 0 0 0 (-∞,∞)
  MFNode [in,out] children [] [X3DChildNode]
  SFBool [in out] bboxDisplay FALSE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFRotation [in,out] rotation 0 0 1 0 [-1,1] or (-∞,∞)
  SFVec3f [in,out] scale 1 1 1 (-∞, ∞)
  SFRotation [in,out] scaleOrientation 0 0 1 0 [-1,1] or (-∞,∞)
  SFVec3f [in,out] translation 0.0.0 (-∞,∞)
  SFBool [in out] visible TRUE
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}

```

The Transform node is a grouping node that defines a coordinate system for its children that is relative to the coordinate systems of its ancestors. See [4.3.5 Transformation hierarchy](#) and [4.3.6 Standard units and coordinate system](#) for a description of coordinate systems and transformations.

[10.2.1 Grouping and children node types](#), provides a description of the *children*, *addChildren*, and *removeChildren* fields.

The *bboxCenter* and *bboxSize* fields specify a bounding box that encloses the children of the Transform node. This is a hint that may be used for optimization purposes. The results are undefined if the specified bounding box is smaller than the actual bounding box of the children at any time. A default *bboxSize* value, (-1, -1, -1), implies that the bounding box is not specified and, if needed, shall be calculated by the browser. The bounding box shall be large enough at all times to enclose the union of the group's children's bounding boxes; it shall not include any transformations performed by the group itself (*i.e.*, the bounding box is defined in the local coordinate system of the children). The results are undefined if the specified bounding box is smaller than the true bounding box of the group. A description of the *bboxCenter* and *bboxSize* fields is provided in [10.2.2 Bounding boxes](#).

The *translation*, *rotation*, *scale*, *scaleOrientation* and *center* fields define a geometric 3D transformation consisting of (in order):

- a (possibly) non-uniform scale about an arbitrary point;
- a rotation about an arbitrary point and axis;
- a translation.

The *center* field specifies a translation offset from the origin of the local coordinate system (0,0,0). The *rotation* field specifies a rotation of the coordinate system. The *scale* field specifies a non-uniform scale of the coordinate system. Scale values may have any value: positive, negative (indicating a reflection), or zero. A value of zero indicates that any child geometry shall not be displayed. The *scaleOrientation* specifies a rotation of the coordinate system before the scale (to specify scales in arbitrary orientations). The *scaleOrientation* applies only to the scale operation. The *translation* field specifies a translation to the coordinate system.

Given a 3-dimensional point **P** and Transform node, **P** is transformed into point **P'** in its

parent's coordinate system by a series of intermediate transformations. In matrix transformation notation, where C (*center*), SR (*scaleOrientation*), T (*translation*), R (*rotation*), and S (*scale*) are the equivalent transformation matrices,

$$P' = T * C * R * SR * S * -SR * -C * P$$

The following Transform node:

```
Transform {
  center          C
  rotation        R
  scale           S
  scaleOrientation SR
  translation     T
  children        [
    # Point P (or children holding other geometry)
  ]
}
```

is equivalent to the nested sequence of:

```
Transform {
  translation T
  children Transform {
    translation C
    children Transform {
      rotation R
      children Transform {
        rotation SR
        children Transform {
          scale S
          children Transform {
            rotation -SR
            children Transform {
              translation -C
              children [
                # Point P (or children holding other geometry)
              ]
            }
          }
        }
      }
    }
  }
}
```

## 10.5 Support levels

The Grouping component provides four levels of support as specified in [Table 10.2](#).

**Table 10.2 — Grouping component support levels**

Level	Prerequisites	Nodes/Features	Support
1	Core 1		
		<i>X3DBoundedObject</i> (abstract)	n/a
		<i>X3DGroupingNode</i> (abstract)	n/a
		Group	<i>addChilden</i> optionally supported. <i>removeChildren</i> optionally supported. Otherwise as for all groups.
		Transform	<i>addChilden</i> optionally supported. <i>removeChildren</i> optionally supported. Otherwise as for all groups.



<b>2</b>	Core 1		
		All Level 1 Grouping nodes	All fields fully supported.
		Switch	All fields fully supported.
<b>3</b>	Core 1		
		All Level 2 Grouping nodes	All fields fully supported.
		StaticGroup	All fields fully supported.





## Extensible 3D (X3D) Part 1: Architecture and base components

### 31 Programmable shaders component

---



#### 31.1 Introduction

##### 31.1.1 Name

The name of this component is "Shaders". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 31.1.2 Overview

This clause describes the Programmable Shaders component of this part of ISO/IEC 19775. This includes how programmable shaders are specified and how they affect the visual appearance of geometry. [Table 31.1](#) provides links to the major topics in this clause.

**Table 31.1 — Topics**

- [31.1 Introduction](#)
  - [31.1.1 Name](#)
  - [31.1.2 Overview](#)
- [31.2 Concepts](#)
  - [31.2.1 Overview](#)
  - [31.2.2 Shading languages](#)
    - [31.2.2.1 Shader language options](#)
    - [31.2.2.2 Node representation](#)
    - [31.2.2.3 Selecting an appropriate shader](#)
    - [31.2.2.4 Per-vertex attributes](#)
    - [31.2.2.5 Per-object attributes](#)
    - [31.2.2.6 Handling errors](#)
  - [31.2.3 Interaction with other nodes and components](#)
    - [31.2.3.1 Overview](#)
    - [31.2.3.2 Lighting](#)
    - [31.2.3.3 Geometry](#)
    - [31.2.3.4 LoadSensor](#)

- [31.2.4 Conformance](#)
    - [31.2.4.1 Component support](#)
    - [31.2.4.2 Node support](#)
    - [31.2.4.3 Language support](#)
    - [31.2.4.4 Scene graph interaction](#)
- [31.3 Abstract types](#)
  - [31.3.1 X3DProgrammableShaderObject](#)
  - [31.3.2 X3DShaderNode](#)
  - [31.3.3 X3DVertexAttributeNode](#)
- [31.4 Node reference](#)
  - [31.4.1 ComposedShader](#)
  - [31.4.2 FloatVertexAttribute](#)
  - [31.4.3 Matrix3VertexAttribute](#)
  - [31.4.4 Matrix4VertexAttribute](#)
  - [31.4.5 PackagedShader](#)
  - [31.4.6 ProgramShader](#)
  - [31.4.7 ShaderPart](#)
  - [31.4.8 ShaderProgram](#)
- [31.5 Support levels](#)
- [Table 31.1 — Topics](#)
- [Table 31.2 — Shader language node specifications](#)
- [Table 31.3 — Shader component support levels](#)

## 31.2 Concepts

### 31.2.1 Overview

Programmable shading provides a method for authors to directly specify how an object is rendered by providing a method of programmatically modifying sections of the rendering pipeline. This allows replacement of the traditional fixed function pipeline of the graphics API to support a variety of visual effects that typically cannot be implemented using other node components in this standard.

Shaders are typically defined by two separate program pieces. One piece is used to modify the vertex values. This piece may also generate values that can be interpolated between vertex values. Such program pieces are termed *vertex shaders*. The other piece is used to modify individual pixels as they are drawn to screen. These program pieces are termed *fragment shaders* or *pixel shaders*. Although not currently defined, future extensions may include other types of shaders that fit into other places in the graphics pipeline.

### 31.2.2 Shader languages

#### 31.2.2.1 Shader language options

Shader programs can be written in several shading languages. Each language is specific

to the underlying rendering API. Typically a language for one API (e.g., Microsoft DirectX) is not usable in another rendering API (e.g., OpenGL) and therefore there is no mandatory requirement for an X3D browser to implement any specific language. A browser implementing this component shall be required to support at least one shading language. The following annexes defines the interface to three popular shader languages:

- [Annex I OpenGL shading language \(GLSL\) binding](#)
- [Annex J Microsoft High Level shading language \(HLSL\) binding](#)
- [Annex K nVidia Cg shading language binding](#)

Shader programs are either defined in a file that can be externally loaded or defined internally within the X3D world. Typically, a separate file is used to specify each type of shader (fragment or vertex) although this is not required.

Some formats are being developed that allow all shaders to be collected together into a single file and used directly by the rendering API. For these file types, a separate [PackagedShader](#) node is used. This node is independent of the underlying rendering API, though the specific file format used within a PackagedShader node may be specific to a particular rendering API.

### 31.2.2.2 Node representation

Each shading language option has a node that implements its functionality. Since each language is quite different, it is not possible to define a single set of nodes that can represent the entire capabilities offered. Each language has its own set of nodes that pertain only to that shading language.

For each set of nodes for a given shading language, there are language-specific behaviours. Mappings for each of the languages are defined in their own annex to this specification as described in [31.2.2.1 Shader language options](#). [Table 2](#) lists the nodes and which annex shall be used to define language-specific behaviours:

**Table 31.2 — Shader language node specifications**

Shading Language	Nodes	Annex
OpenGL GLSLang	GLSLShader GLSLShaderObject	<a href="#">Annex I</a>
Direct3D HLSL	HLSLShader	<a href="#">Annex J</a>
nVidia Cg	CgShader	<a href="#">Annex K</a>

### 31.2.2.3 Selecting an appropriate shader

Browsers are not expected to be able to handle all the different forms of shading languages. In fact, most are incompatible with any rendering API apart from the one for which they are defined.

To allow the author to specify a collection of shader options for the browser to select and for the browser to choose the shader version it can run, a fallback mechanism is defined for the *shader* field of the [Appearance](#) node.

The *shader* field is an MFNode field that defines the collection of pertinent shader nodes of various languages in the order of preference. The first node declared is the highest preference. If the browser does not support the language defined for the current preference level, the browser shall set the node's *isSelected* field output to `FALSE`, and move to the next preference. A browser implementation shall support all nodes if the Programmable Shader component is supported, but is not required to execute the contained script in every shader node. Ignored nodes shall remain a functional part of the scene graph, continuing to interact with the event model as required by the field access types.

When a shader is found that can be executed by the browser, it shall set the *isSelected* field output to `TRUE`.

A browser may select an appropriate shader on grounds other than just the shading language used.

**EXAMPLE** The local hardware not supporting some of the features requested or the shader running in software rather than hardware are considered valid reasons for not selecting a shader to run.

### 31.2.2.4 Per-vertex attributes

Advanced vertex shaders often need to provide extra information on a per vertex basis (e.g., temperature information in an analysis system or weighting values for a skin and bones system). Per-vertex attributes may be supplied for any geometry that extends [X3DComposedGeometryNode](#) as described in [11.3.2 X3DComposedGeometryNode](#). Both matrix and vector values may be supplied on a per-vertex basis through nodes that are extended from [X3DVertexAttributeNode](#).

Each shading language uses a different method of mapping per-vertex attributes to the user-provided shading language code. The definition of how to interpret the *name* field value to the individual shading language file is defined in the language-specific annex (see [Table 31.2](#)).

Per vertex attributes are mapped 1:1 to each vertex value. When used in a node derived from [X3DComposedGeometryNode](#), the number of values defined in the node containing the attribute values shall be identical to the number of coordinate values specified for the geometry node. If the number of values does not match, the visual result is implementation specific.

### 31.2.2.5 Per-object attributes

Shaders often need to provide specific per-object values (e.g., the colour of the light). The most common name for one of these values is *uniform variable*. Uniform variables are defined using custom field definitions that allow objects to be set as required. The placement of these fields depends on the shading language itself, as the amount of customizability is dependent on the shading language.

Field names shall be mapped to the shading API as the uniform variable name of the

identical name in the shader file.

NOTE Some shading languages cannot handle the full UTF-8 character set required by this International Standard.

For fields of type SFNode or MFNode, the mapping to the shading language is dependent on the individual shading language. The applicable language binding annex specifies the required behaviour (see [Table 31.2](#)).

### 31.2.2.6 Handling errors

If a shader program does not have valid syntax or the environment does not contain enough information for the shader to render, implementations shall track errors during all stages of the shader process and display them to the browser's console.

A shader that fails during run-time or during the compilation or validation stages shall not run. A browser shall use the rendering API's default behaviour for this situation. If a user requires some fallback behaviour, such as the browser not supporting the shader capabilities requested, other nodes such as [LoadSensor](#), [Script](#), and/or [Switch](#) can be used to specify the required visual output.

## 31.2.3 Interaction with other nodes and components

### 31.2.3.1 Overview

Programmable shaders typically replace large amounts of functionality that would be traditionally implemented by the browser. The effect of each shader language varies depending on the amount of processing that the user will be required to perform. Some languages may completely disable anything that would be automatically generated (*e.g.*, texture coordinates or normals) while others may not. A reasonable assumption is that everything is disabled for any geometry that has a shader associated with it. Each language shading definition annex specifies exactly the semantics that can be expected of the underlying rendering API, and by implication, the browser.

### 31.2.3.2 Lighting

If the user provides a fragment shader, the shader shall be responsible for all lighting associated with the affected geometry. The lighting definitions in [17 Lighting component](#) shall be ignored. Where possible, all of the lighting information such as the currently set lights, material colours and textures shall be made available to the shader. Some rendering APIs may not be able to make available all of this information. In this case, it is acceptable to provide alternative mapping hints as part of the node definition. The individual shading language annexes contain more information (see [Table 31.2](#)).

### 31.2.3.3 Geometry

Since a vertex shader may move the vertex from its original location in the local coordinate system, it can produce many large-scale side effects. A major problem is that the browser implementation may have no idea where the final geometry has been placed. Any action that relies on knowing the exact position of vertices may fail to act properly. In particular, terrain following, collision detection and sensors can be

adversely effected.

Because a vertex may be shifted in world space, it is recommended that if a user requires this ability, a means of giving a rough approximation of the geometry to the browser should be provided, either through setting an explicit bounding box on the containing [Shape](#) node or by providing the source geometry as close to the final output shape as possible.

EXAMPLE A fuzzy rabbit shape would start with the source vertices in the shape of the base rabbit geometry.

### **31.2.3.4 LoadSensor**

A shader is considered loaded when the source for the shader program has been downloaded successfully. A shader is considered valid when the downloaded file has been compiled and registered with the rendering API, which then considers it a valid object.

## **31.2.4 Conformance**

### **31.2.4.1 Component support**

An implementation shall indicate support for this component if and only if the user's particular hardware is capable of supporting this component, either through direct hardware support or software emulation. If the user's machine is not capable of supporting this component, the browser shall indicate a failure by stopping at the appropriate PROFILE or COMPONENT statement of the file, in accordance with [7.2.5.3 PROFILE statement](#) or [7.2.5.4 COMPONENT statement](#).

### **31.2.4.2 Node support**

A conformant browser for this component shall support all the nodes at a given level. However, a conformant browser is not required to support the corresponding shading language for that node. If a browser is not supporting the language, the nodes that provide access to that language shall be read and ignored. These ignored nodes shall still exist as part of the X3D scene graph, and shall still honour the X3D event model.

EXAMPLE Any inputOutput fields shall still be required to implement output events if a value is written to the input.

### **31.2.4.3 Language support**

A browser conformant to this component shall support at least one shading language as listed in [Table 31.2](#).

### **31.2.4.4 Scene graph interaction**

A shader containing a vertex shader shall be required to be conformant only to either the explicit bounding boxes or the original source geometry definition. It is not required to obtain the output vertex information for use within the scene graph.



## 31.3 Abstract types

### 31.3.1 X3DProgrammableShaderObject

```
X3DProgrammableShaderObject {
}
```

This abstract node type is the base type interface is the marker for all shader-related node types that specify arbitrary fields for interfacing with per-object attribute values.

A concrete [X3DProgrammableShaderObject](#) node instance is used to program behaviour for a shader in a scene. The shader is able to receive and process events that are sent to it. Each event that can be received shall be declared in the shader node using the same field syntax as is used in a prototype definition:

```
inputOnly type name
```

The type can be any of the standard X3D fields (as defined in [5 Field type reference](#)). The name shall be an identifier that is unique for this shader node and is used to map the value to the shader program's uniform variable of the same name. If a shader program does not have a matching uniform variable, the field value is ignored.

OutputOnly fields are not required to generate output events from a shader. Current hardware shader technology does not support this capability, though future versions may.

It is recommended that user-defined field or event names defined in shader nodes follow the naming conventions described in [Part 2 of ISO/IEC 19775](#).

### 31.3.2 X3DShaderNode

```
X3DShaderNode : X3DAppearanceChildNode {
  SFBool [in] activate
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFBool [out] isSelected
  SFBool [out] isValid
  SFString [] language "" ["Cg"|"GLSL"|"HLSL"|"..."]
}
```

This abstract node type is the base type for all node types that specify a programmable shader.

The *activate* field forces the shader to activate the contained objects. The conditions under which an *activate* event may be required are described in [OpenGL 1.5 Event model](#), [Microsoft High Level Shading Language \(HLSL\) 1.5 Event model](#), and [nVidia Cg K.6 Event model](#).

The *isSelected* output field is used to indicate that this shader instance is the one selected for use by the browser. A `TRUE` value indicates that this instance is in use. A `FALSE` value indicates that this instance is not in use. The rules for when a browser decides to select a particular node instance are described in [31.2.2.3 Selecting an appropriate shader](#).

The *isValid* field is used to indicate whether the current shader objects can be run as a shader program.

EXAMPLE There are no syntax errors and the hardware can support all the required features.



The definition of this field may also add additional semantics on a per-language basis.

The *language* field is used to indicate to the browser which shading language is used for the source file(s). This field may be used as a hint for the browser if the shading language is not immediately determinable from the source (e.g., a generic MIME type of `text/plain` is returned). A browser may use this field for determining which node instance will be selected and to ignore languages that it is not capable of supporting. Three basic language types are defined for this specification and others may be optionally supported by a browser.

### 31.3.3 X3DVertexAttributeNode

```
X3DVertexAttributeNode : X3DGeometricPropertyNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFString [] name ""
}
```

This abstract node type is the base type for all node types that specify per-vertex attribute information to the shader.

The *name* field describes a name that is mapped to the shading language-specific name for describing per-vertex data. The appropriate shader language annex (see [Table 31.2](#)) annex contains language-specific binding information.

## 31.4 Node reference

### 31.4.1 ComposedShader

```
ComposedShader : X3DShaderNode, X3DProgrammableShaderObject {
  SFBool [in] activate
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFNode [in,out] parts [] [ShaderPart]
  SFBool [out] isSelected
  SFBool [out] isValid
  SFString [] language "" ["Cg"|"GLSL"|"HLSL"|...]
  # And any number of:
  fieldType [] fieldName
  fieldType [in] fieldName
  fieldType [out] fieldName
  fieldType [in,out] fieldName
}
```

The `ComposedShader` node defines a shader where the individual source files are not individually programmable. All access to the shading capabilities is defined through a single interface that applies to all parts.

EXAMPLE OpenGL Shading Language (GLSL)

The *isValid* field adds an additional semantic indicating whether the current shader parts can be linked together to form a complete valid shader program.

~~The *activate* field forces the shader to activate the contained objects. The conditions under which a activate may be required are described in [1.5 Event model](#).~~

### 31.4.2 FloatVertexAttribute

```
FloatVertexAttribute : X3DVertexAttributeNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
```

```

MFFloat [in,out] value      [] (- , )
SFString [] name           ""
SFInt32  [] numComponents 4 [1..4]
}

```

The `FloatVertexAttribute` node defines a set of per-vertex single-precision floating point attributes.

The `numComponents` field specifies how many consecutive floating point values should be grouped together per vertex. The length of the `value` field shall be a multiple of `numComponents`.

The `value` field specifies an arbitrary collection of floating point values that will be passed to the shader as per-vertex information. The specific type mapping to the individual shading language data types is in the appropriate language-specific annex (see [Table 31.2](#)).

### 31.4.3 Matrix3VertexAttribute

```

Matrix3VertexAttribute : X3DVertexAttributeNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFMatrix3f [in,out] value [] (-∞,∞)
  SFString [] name ""
}

```

The `Matrix3VertexAttribute` node defines a set of per-vertex  $3 \times 3$  matrix attributes.

The `value` field specifies an arbitrary collection of matrix values that will be passed to the shader as per-vertex information. The specific type mapping to the individual shading language data types shall be found in the appropriate language-specific annex (see [Table 31.2](#)).

### 31.4.4 Matrix4VertexAttribute

```

Matrix4VertexAttribute : X3DVertexAttributeNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFMatrix4f [in,out] value [] (-∞,∞)
  SFString [] name ""
}

```

The `Matrix4VertexAttribute` node defines a set of per-vertex  $4 \times 4$  matrix attributes.

The `value` field specifies an arbitrary collection of matrix values that will be passed to the shader as per-vertex information. The specific type mapping to the individual shading language data types shall be found in the appropriate language-specific annex (see [Table 31.2](#)).

### 31.4.5 PackagedShader

```

PackagedShader : X3DShaderNode, X3DUrlObject, X3DProgrammableShaderObject {
  SFBool [in] activate
  SFString [in,out] description ""
  SFBool [in,out] load TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTime [in,out] refresh 0.0 [0,∞)
  MFString [in,out] url [] [URI]
  SFBool [out] isSelected
  SFBool [out] isValid
  SFString [] language "" ["Cg"|"GLSL"|"HLSL"...]

  # And any number of:
  fieldType [in] fieldName
  fieldType [in,out] fieldName initialValue
  fieldType [out] fieldName
  fieldType [] fieldName initialValue
}

```

A `PackagedShader` node describes a single file that may contain a number of shaders and combined effects.

EXAMPLE The Microsoft `.fx` file format represents this type of shader (see [\[FX\]](#)).

The shader source is read from the URL specified by the `url` field. When the `url` field contains no values (`[]`), this object instance is ignored. The `url` field is defined in [9.2.1 URLs](#). No file formats are required to be supported for this node.

The `language` field may be used to optionally determine the language type if no MIME-type information is available.

If the `refresh` field results in a new script getting loaded or the prior script getting reloaded, then all fields are re-initialized to their initially defined values, and the `initialize()` method is invoked, if provided, as defined in [29.2.3 initialize\(\) and shutdown\(\)](#).

WARNING Automatically reloading content can have security considerations and needs to be considered carefully.

## 31.4.6 ProgramShader

```
ProgramShader : X3DShaderNode {
  SFBool [in] activate
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFNode [in,out] programs [] [ShaderProgram]
  SFBool [out] isSelected
  SFBool [out] isValid
  SFString [] language "" ["Cg"|"GLSL"|"HLSL"|"..."]
}
```

The `ProgramShader` node defines a shader that can consist of one or more individually programmable, self contained pieces. Each piece, represented by a `ShaderProgram` node, shall be a self-contained source that does not rely on any other source file and can manage one part of the programmable pipeline (e.g., vertex or fragment processing).

The `programs` field consists of zero or more [ShaderProgram](#) node instances. In general, only two `ShaderProgram` instances are needed: one each for vertex and fragment processing. Each shader language annex shall define the required behaviour for processing this field.

The `isValid` field may add an additional semantic to indicate whether all parts are available.

EXAMPLE Microsoft's HLSL requires that both vertex and fragment programs be provided. It specifies that it is an error to provide one and not the other.

## 31.4.7 ShaderPart

```
ShaderPart : X3DNode, X3DUrlObject {
  SFString [in,out] description ""
  SFBool [in,out] load TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTime [in,out] refresh 0.0 [0,∞)
  MFString [in,out] url [] [URI]
  SFString [] type "VERTEX" ["VERTEX"|"FRAGMENT"]
}
```

The ShaderPart node defines the source for a single object to be used by a [ComposedShader](#) node. The source is not required to be a complete shader for all of the vertex/fragment processing.

The *type* field indicates whether this object shall be compiled as a vertex shader, fragment shader, or other future-defined shader type.

The shader source is read from the URL specified by the *url* field. When the *url* field contains no values ([ ]), this object instance is ignored. The *url* field is defined in [9.2.1 URLs](#). Shader source files shall be plain text encoded as specified for MIME type `text/plain` and interpreted according to the *type* field.

If the *refresh* field results in a new script getting loaded or the prior script getting reloaded, the *initialize()* method is invoked, if provided, as defined in [29.2.3 initialize\(\) and shutdown\(\)](#).

**WARNING** Automatically reloading content can have security considerations and needs to be considered carefully.

## 31.4.8 ShaderProgram

```
ShaderProgram : X3DNode, X3DUrlObject, X3DProgrammableShaderObject {
  SFString [in,out] description ""
  SFBool [in,out] load TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTime [in,out] refresh 0.0 [0,∞)
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFString [in,out] url [] [URI]
  SFString [] type "VERTEX" ["VERTEX"|"FRAGMENT"]

  # And any number of:
  fieldType [in] fieldName
  fieldType [in,out] fieldName initialValue
  fieldType [out] fieldName
  fieldType [] fieldName initialValue
}
```

The ShaderProgram node provides the source and interface to a self contained program that occupies one part of the rendering process: either a vertex or fragment shader.

The shader source is read from the URL specified by the *url* field. When the *url* field contains no values ([ ]), this object instance is ignored. The *url* field is defined in [9.2.1 URLs](#). Shader source files shall be plain text encoded as specified for MIME type `text/plain` and interpreted according to the containing node's language definition.

If the *refresh* field results in a new script getting loaded or the prior script getting reloaded, then all fields are re-initialized to their initially defined values, and the *initialize()* method is invoked, if provided, as defined in [29.2.3 initialize\(\) and shutdown\(\)](#).

**WARNING** Automatically reloading content can have security considerations and needs to be considered carefully.

## 31.5 Support levels

The Programmable Shaders component defines a single level of support as specified in [Table 31.3](#).

**Table 31.3 — Programmable shaders component support levels**

Level	Prerequisites	Nodes/Features	Support
1	Core 1 Grouping 1 Shape 1 Rendering 1		
		<i>X3DProgrammableShaderObject</i>	n/a
		<i>X3DShaderNode</i>	n/a
		<i>X3DVertexAttributeNode</i>	n/a
		FloatVertexAttribute	All fields fully supported.
		ComposedShader	All fields fully supported.
		ShaderPart	All fields fully supported.
		ProgramShader	All fields fully supported.
		ShaderProgram	All fields fully supported.
		Matrix3VertexAttribute	All fields fully supported.
		Matrix4VertexAttribute	All fields fully supported.
		PackagedShader	All fields fully supported.





## Extensible 3D (X3D) Part 1: Architecture and base components

### Annex J (normative)

## Microsoft High Level Shading Language (HLSL) binding

---

### J.1 General

This annex defines the mapping of concepts of the programmable shaders component to the Microsoft High Level Shading Language (HLSL) (see [\[HLSL\]](#)). It applies to the ProgramShader, ShaderProgram and PackagedShader nodes with the *language* field set to "HLSL".

### J.2 Topics

[Table 1](#) provides links to the major topics in this annex.

**Table J.1 — Topics**

- [J.1 General](#)
- [J.2 Topics](#)
- [J.3 Interaction with other nodes and components](#)
  - [J.3.1 Vertex shader](#)
  - [J.3.2 Fragment shader](#)
  - [J.3.3 LoadSensor](#)
  - [J.3.4 Vertex attributes](#)
- [J.4 Data type and parameter mappings](#)
  - [J.4.1 Node fields](#)
  - [J.4.2 X3D field types to HLSL data types](#)
  - [J.4.3 X3D world state to HLSL parameter names](#)
- [J.5 Event Model](#)
  - [J.5.1 Changing URL fields](#)

- [J.5.2 Changing the \*attrib\* field](#)
- [J.5.3 Activating programs](#)
- [Table J.1 — Topics](#)
- [Table J.2 — Supported Direct3D vertex declaration usage types](#)
- [Table J.3 — Mapping of X3D texture node types to HLSL sampler types](#)
- [Table J.4 — Mapping of X3D material and light node types to HLSL structure declarations](#)
- [Table J.5 — Mapping of X3D field types to HLSL data types](#)
- [Table J.6 — Mapping of X3D world State to HLSL parameter names](#)

## J.3 Interaction with other nodes and components

### J.3.1 Vertex shader

The vertex shader replaces the vertex processing done by the Microsoft Direct3D graphics pipeline. While using a vertex shader, state information regarding transformation and lighting operations is ignored by the fixed-function pipeline. The HLSL specification states that the following functionality is disabled if a vertex shader is supplied:

- a. The model view matrix is not applied to vertex coordinates.
- b. The projection matrix is not applied to vertex coordinates.
- c. The texture matrices are not applied to texture coordinates.
- d. The normals are not transformed to eye coordinates.
- e. The normals are not rescaled or normalized.
- f. Texture coordinates are not generated automatically.
- g. Per vertex lighting is not performed.
- h. Color material lighting is not performed.
- i. Point size distance attenuation is not performed.

The fixed-function pipeline Direct3D graphics state is not available for use within an HLSL shader program. Shaders that wish to make use of this data, such as material, lighting, texture and transformation matrix state, shall declare parameters of the appropriate type and pass values into them via declared fields of the containing ShaderProgram node in the X3D scene graph. The parameter types and mappings to those types from built-in X3D values are defined in [J.4 Data type and parameter mappings](#).

### J.3.2 Fragment Shader

The fragment shader, also known as a *pixel* shader in HLSL, replaces the fixed functionality of the Direct3D fragment processor. The HLSL specification states that “textures are not applied” if a fragment shader is supplied.

The fixed function pipeline Direct3D graphics state is not available for use within an HLSL pixel shader program. Shaders that wish to make use of this data, such as material, lighting, texture and transformation matrix state, shall declare parameters of



the appropriate type and pass values into them via declared fields of the containing ShaderProgram node in the X3D scene graph. The parameter types and mappings to those types from built-in X3D values are defined in [J.4 Data type and parameter mappings](#).

### J.3.3 LoadSensor

The LoadSensor node (See [9.4.3 LoadSensor](#)) has two output fields *isActive* and *isLoaded*. The *isLoaded* field behaviour is unchanged.

The *isActive* field is defined to issue a `TRUE` event when all the following conditions have been satisfied:

- a. The content identified by the *url* field has been successfully loaded.
- b. The shader program has been successfully compiled without error.

### J.3.4 Vertex attributes

Each vertex attribute node directly maps the *name* field to a Direct3D usage type for use within a Direct3D vertex declaration (with the prefix "`D3DDECLUSAGE_`" prepended to the name), as well as an HLSL binding semantic of the same name defined on the varying inputs to a shader program. This language binding allows the use of the predefined Direct3D vertex declaration usage types and HLSL binding semantics listed in [Table J.2](#).

**Table J.2 — Supported Direct3D vertex declaration usage types**

Direct3D usage type
<i>POSITION</i>
<i>NORMAL</i>
<i>TEXCOORD</i>
<i>TANGENT</i>
<i>BINORMAL</i>
<i>COLOR</i>
<i>FOG</i>

The browser implementation shall automatically assign appropriate internal index values for each attribute in the case where multiple nodes have the same value in the *name* field.

## J.4 Data Type and Parameter Mappings

### J.4.1 Node fields

Fields that are of type SFNode/MFNode are ignored unless the value is of type *X3DTextureNode*, *X3DMaterialNode*, or *X3DLightNode*. Field instances of type *X3DTextureNode* are mapped according to the appropriate Direct3D sampler data type. The mapping from texture nodes to built-in sampler types is defined in [Table J.3](#).

**Table J.3 — Mapping of X3D texture node types to HLSL sampler types**

X3D Texture type	HLSL variable type
<i>X3DTexture2DNode</i>	sampler2D
<i>X3DTexture3DNode</i>	sampler3D
<i>X3DEnvironmentTextureNode</i>	samplerCube

X3D does not define mappings to the HLSL types sampler1D, sampler1DShadow and sampler2DShadow.

Field instances of type *X3DMaterialNode* and *X3DLightNode* are mapped to structures that shall be declared in the shader program as defined in [Table J.4](#).

**Table J.4 — Mapping of X3D material and light node types to HLSL structure declarations**

X3D node type	HLSL structure declaration	Additional information
<i>X3DMaterialNode</i>	<pre>struct X3DMaterial {     float4     diffuseColor;     float4     ambientColor;     float4     specularColor;     float4     emissiveColor;     float power; };</pre>	All color values are 4-component with alpha value = 1.0.
<i>X3DLightNode</i>	<pre>struct X3DLight {     int type;     float4     diffuseColor;     float4     specularColor;     float4     ambientColor;     point3 position;     point3 direction;     float range;     float falloff;     float     attenuation0;     float     attenuation1;     float     attenuation2;     float theta;     float phi;     bool on; };</pre>	<p>Valid <i>type</i> member values are 1 for Point light, 2 for Spot light and 3 for Direction light.</p> <p>All color values are 4-component with alpha value = 1.0.</p> <p>All position, direction and scalar values are assumed to be in world space.</p> <p>The <i>on</i> member specifies whether the light is enabled.</p>

## J.4.2 X3D field types to HLSL data types

[Table J.5](#) specifies the mapping of X3D field types to data types used in the HLSL Language.

**Table J.5 — Mapping of X3D Field Types to HLSL Data Types**

X3D Field type	HLSL Data Type
SFBool	bool
MFBBool	bool[]
MFInt32	int[]
SFInt32	int
SFFloat	float
MFFloat	float[]
SFDouble	double
MFDouble	double[]
SFTime	double
MFTime	double[]
SFNode	See <a href="#">J.4.1 Node fields</a>
MFNode	See <a href="#">J.4.1 Node fields</a>
SFVec2f	float2
MFVec2f	float2[]
SFVec3f	float3
MFVec3f	float3[]
SFVec4f	float4
MFVec4f	float4[]
SFVec3d	float3
MFVec3d	float3[]
SFVec4d	float4
MFVec4d	float4[]
SFRotation	float4

MFRotation	float4[]
MFColor	float4[]
SFColor	float4
SFImage	int[]
MFImage	int[]
SFString	Not supported
MFString	Not supported
SFMatrix3f	float3x3
MFMatrix3f	float3x3[]
SFMatrix4f	float4x4
MFMatrix4f	float4x4[]

HLSL defines maximum supported lengths of each array data type, which may conflict with the minimum support requirements for X3D.

### J4.3 X3D world state to HLSL parameter names

Certain internal states of the X3D world, such as transformation matrices, or the viewer's position in world space, are neither readily available via the HLSL shader program nor directly accessible from the X3D scene graph. Thus, if used, these state values shall be explicitly passed in to the shader program as named parameters. This binding defines an automatic mapping of these states to predefined shader program parameter names. [Table J.6](#) defines the mapping of internal states of the X3D world to parameter names used in HLSL programs.

**Table J.6 — Mapping of X3D world state to HLSL parameter names**

Parameter name	Description
<b>model</b>	This name refers to the matrix transforming from local to global coordinates. The model matrix transforms vertices from their model position to their position in world space ( <i>i.e.</i> , after the effects of all Transform nodes have been applied).
<b>view</b>	This name refers to the viewing matrix transforming from world to view relative coordinates.
<b>projection</b>	This name refers to the projection matrix transforming from viewing relative coordinates to clip space, including the projective part.

<b>modelView</b>	This name refers to the matrix that represents the concatenation of model and view matrices. This matrix transforms vertices from their model position to their position in view space ( <i>i.e.</i> , after the effects of all Transform nodes and the current viewpoint have been applied).
<b>modelViewProjection</b>	This name refers to the matrix that represents the concatenation of model, view and projection matrices. This matrix transforms vertices from their model position to their final position in clip space.
<b>viewPosition</b>	This name refers to the current viewer position in world space coordinates.

The following suffixes can be applied to the matrix built-in values. A suffix of *I* signifies the inverse of the matrix. *T* signifies the transpose of the matrix. *IT* signifies the inverse transpose of the matrix.

## J.5 Event model

### J.5.1 Changing URL fields

When the *url* receives an event changing the value, the browser shall immediately attempt to download the new source. Upon successful download, the browser shall attempt to compile the new source and issue the appropriate LoadSensor events. It shall not automatically activate the shader program, nor disable the currently running shader.

Values defined at load time of the file do not require an explicit request to activate the shader program. The browser shall be assumed to automatically activate the program once all the objects have successfully downloaded. If some of the shader source files are not downloaded or compiled (*e.g.*, due to errors) no activation will occur for the shader program.

### J.5.2 Changing the *attrib* field

Per-vertex attributes may be defined as one of the fields of *X3DComposedGeometryNode*. These may be changed at runtime by adding or removing node instances. Adding new node instances to the field shall require that the user request an explicit activate in order to make them visible to the shader.

### J.5.3 Activating programs

The user may, at any time, request that the browser activate the composing shader objects by sending a `TRUE` value to the *activate* inputOnly field of the ProgramShader or PackagedShader node. Users may need to force a re-activation of the node under various circumstances, such as changing the *url* field of one or more ShaderProgram or PackagedShader nodes, or adding or removing ShaderProgram nodes from the *programs* field of the ProgramShader node. Reactivating the shader shall replace the

existing shader with the new compiled shader for subsequent rendering.



Draft



## Extensible 3D (X3D) Part 1: Architecture and base components

### 11 Rendering component

---



#### 11.1 Introduction

##### 11.1.1 Name

The name of this component is "Rendering". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 11.1.2 Overview

This clause describes the Rendering component of this part of ISO/IEC 19775. This includes fundamental rendering primitives such as [TriangleSet](#) and [PointSet](#) nodes, and geometric properties nodes that define how coordinate indices, colors, normals and texture coordinates are specified. [Table 11.1](#) provides links to the major topics in this clause.

**Table 11.1 — Topics**

- [11.1 Introduction](#)
  - [11.1.1 Name](#)
  - [11.1.2 Overview](#)
- [11.2 Concepts](#)
  - [11.2.1 Rendering primitives](#)
  - [11.2.2 Geometric properties](#)
    - [11.2.2.1 Overview](#)
    - [11.2.2.2 Color](#)
    - [11.2.2.3 Coordinates](#)
    - [11.2.2.4 Normals](#)
  - [11.2.3 Common geometry fields](#)
  - [11.2.4 Clip planes](#)
    - [11.2.4.1 Overview](#)
    - [11.2.4.2 Clip plane semantics](#)
    - [11.2.4.3 Transformation hierarchy](#)
    - [11.2.4.4 Scoping of clip planes](#)



- [11.2.4.5 Clip plane limitations](#)
- [11.3 Abstract types](#)
  - [11.3.1 X3DColorNode](#)
  - [11.3.2 X3DComposedGeometryNode](#)
  - [11.3.3 X3DCoordinateNode](#)
  - [11.3.4 X3DGeometricPropertyNode](#)
  - [11.3.5 X3DGeometryNode](#)
  - [11.3.6 X3DNormalNode](#)
- [11.4 Node reference](#)
  - [11.4.1 ClipPlane](#)
  - [11.4.2 Color](#)
  - [11.4.3 ColorRGBA](#)
  - [11.4.4 Coordinate](#)
  - [11.4.5 IndexedLineSet](#)
  - [11.4.6 IndexedTriangleFanSet](#)
  - [11.4.7 IndexedTriangleSet](#)
  - [11.4.8 IndexedTriangleStripSet](#)
  - [11.4.9 LineSet](#)
  - [11.4.10 Normal](#)
  - [11.4.11 PointSet](#)
  - [11.4.12 TriangleFanSet](#)
  - [11.4.13 TriangleSet](#)
  - [11.4.14 TriangleStripSet](#)
- [11.5 Support levels](#)
- [Figure 11.1 — Effects of clip planes on geometry](#)
- [Figure 11.2 — TriangleFanSet node](#)
- [Figure 11.3 — TriangleSet node](#)
- [Figure 11.4 — TriangleStripSet node](#)
- [Table 11.1 — Topics](#)
- [Table 11.2 — Rendering component support levels](#)

## 11.2 Concepts

### 11.2.1 Rendering primitives

The following nodes represent the fundamental visual objects common to polygonal rendering systems:

- a. [IndexedLineSet](#),
- b. [IndexedTriangleFanSet](#),
- c. [IndexedTriangleSet](#),
- d. [IndexedTriangleStripSet](#),
- e. [PointSet](#),
- f. [TriangleFanSet](#),

- g. [TriangleSet](#), and
- h. [TriangleStripSet](#).

Most complex geometries, such as those found in the [13 Geometry3D component](#) and [14 Geometry2D component](#), can be implemented as a combination of these nodes. The Rendering component provides these nodes as basic services for building arbitrary geometry types.

All of the rendering primitive nodes are descendants of the [X3DGeometryNode](#) type.

## 11.2.2 Geometric properties

### 11.2.2.1 Overview

Several geometry nodes contain [Coordinate](#), [Color](#) or [ColorRGBA](#), [Normal](#), and [TextureCoordinate](#) as geometric property node types. The geometric property node types are defined as individual node types so that instancing and sharing is possible between different geometry nodes. The [TextureCoordinate](#) node type is defined in [18 Texturing component](#).

#### 11.2.2.2 Color

Color in X3D is specified using the RGB color model in which the three components of color specifications are red, green, and blue ranging in value from 0 to 1. This color model results in a color specification of (0,0,0) representing black and (1,1,1) representing white. Color may also be specified using the RGBA color model in which a fourth alpha component specifies a value ranging from 0 (fully transparent) to 1 (fully opaque). See [\[FOLEY\]](#) for more information on the RGB color model.

#### 11.2.2.3 Coordinates

Coordinates in X3D are specified as an (x, y, z) triplet in a right-handed, rectangular coordinate system.

#### 11.2.2.4 Normals

Normals define perpendicular directions from a piece of geometry and are used to perform lighting calculations. They may either be specified as part of the content or computed directly from the geometry by the browser. When specified as part of the content, each normal vector shall have unit length.

### 11.2.3 Common geometry fields

Certain geometry nodes have several fields that provide information about the rendering of the geometry. These fields specify the vertex ordering, if the shape is solid, if the shape contains convex faces, and at what angle a crease appears between faces, and are named *ccw*, *solid*, *convex* and *creaseAngle*, respectively.

The *ccw* field defines the ordering of the vertex coordinates of the geometry with respect to user-given or automatically generated normal vectors used in the lighting

model equations. If *ccw* is `TRUE`, the normals shall follow the right hand rule; the orientation of each normal with respect to the vertices (taken in order) shall be such that the vertices appear to be oriented in a counterclockwise order when the vertices are viewed (in the local coordinate system of the [Shape](#)) from the opposite direction as the normal. If *ccw* is `FALSE`, the normals shall be oriented in the opposite direction. If normals are not generated but are supplied using a [Normal](#) node, and the orientation of the normals does not match the setting of the *ccw* field, results are undefined.

The *solid* field determines whether one or both sides of each polygon shall be displayed. If *solid* is `FALSE`, each polygon shall be visible regardless of the viewing direction (*i.e.*, no backface culling shall be done, and two sided lighting shall be performed to illuminate both sides of lit surfaces). If *solid* is `TRUE`, the visibility of each polygon shall be determined as follows: Let  $\mathbf{V}$  be the position of the viewer in the local coordinate system of the geometry. Let  $\mathbf{N}$  be the geometric normal vector of the polygon, and let  $\mathbf{P}$  be any point (besides the local origin) in the plane defined by the polygon's vertices. Then if  $(\mathbf{V} \cdot \mathbf{N}) - (\mathbf{N} \cdot \mathbf{P})$  is greater than zero, the polygon shall be visible; if it is less than or equal to zero, the polygon shall be invisible (back face culled).

The *convex* field indicates whether all polygons in the shape are convex (`TRUE`). A polygon is convex if it is planar, does not intersect itself, and all of the interior angles at its vertices are less than 180 degrees. Non planar and self intersecting polygons may produce undefined results even if the *convex* field is `FALSE`.

The *creaseAngle* field affects how default normals are generated. If the angle between the geometric normals of two adjacent faces is less than the crease angle, normals shall be calculated so that the faces are shaded smoothly across the edge; otherwise, normals shall be calculated so that a lighting discontinuity across the edge is produced. Crease angles shall be greater than or equal to 0.0 angle base units.

**EXAMPLE** A crease angle of 0.5 angle base units means that an edge between two adjacent polygonal faces will be smooth shaded if the geometric normals of the two faces form an angle that is less than 0.5 angle base units. Otherwise, the faces will appear faceted.

## 11.2.4 Clip planes

### 11.2.4.1 Overview

The 3D graphics rendering pipeline uses an implicit step of trimming objects that are partially in the view frustum called clipping. In addition to these implied bounds, it is also possible to provide an additional clipping of the geometry through the provision of additional clip plane definitions.

### 11.2.4.2 Clip plane semantics

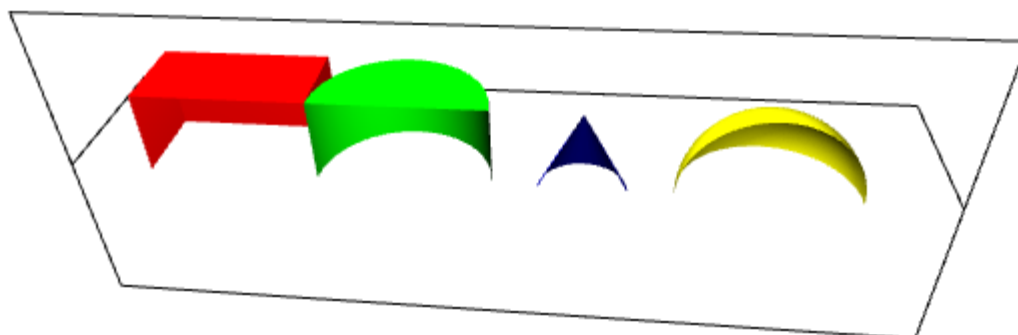
A clip plane is defined as a plane that generates two half-spaces. The effected geometry in the half-space that is defined as being outside the plane is removed from the rendered image as a result of a clipping operation.

### 11.2.4.3 Transformation hierarchy

Clip planes may be defined at any level of the transformation hierarchy. The clip plane

definitions are accumulated from the root of the scene graph down to the individual leaf nodes that are rendered. Clipping occurs against the intersection of the half-spaces resulting from the current list of transformed clip plane definitions. Since the clip planes are collected during the traversal of the scene graph, specifying both local and globally scoped planes is possible.

[Figure 11.1](#) illustrates four objects effected by a horizontal clip plane and a vertical clip plane.



**Figure 11.1 — Effects of clip planes on geometry**

#### 11.2.4.4 Scoping of clip planes

A [ClipPlane](#) node affects only objects that are in the same transformation hierarchy as the node. Each plane is transformed according to the parent transformation hierarchy but is not further transformed by the children it affects.

Clip planes shall affect nodes derived from [X3DBackgroundNode](#).

#### 11.2.4.5 Clip plane limitations

Many renderers only support a limited number of clip plane definitions (typically, six). If, while traversing from the root of the scene to a particular leaf, more than the number of supported clip planes are specified, the clip plane definitions closest to the leaf are discarded first (*i. e.*, the clip planes that are closer to the root of the scene graph are considered most important).

### 11.3 Abstract types

#### 11.3.1 *X3DColorNode*

```
X3DColorNode : X3DGeometricPropertyNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This is the base node type for color specifications in X3D.

#### 11.3.2 *X3DComposedGeometryNode*

```
X3DComposedGeometryNode : X3DGeometryNode {
  MFNode [in,out] attrib [] [X3DVertexAttributeNode]
```

```

SFNode [in,out] color      NULL [X3DColorNode]
SFNode [in,out] coord     NULL [X3DCoordinateNode]
SFNode [in,out] fogCoord  NULL [FogCoordinate]
SFNode [in,out] metadata  NULL [X3DMetadataObject]
SFNode [in,out] normal    NULL [X3DNormalNode]
SFNode [in,out] texCoord  NULL [X3DTextureCoordinateNode]
SFBool [] ccw            TRUE
SFBool [] colorPerVertex TRUE
SFBool [] normalPerVertex TRUE
SFBool [] solid          TRUE
}

```

This is the base node type for all composed 3D geometry in X3D.

A composed geometry node type defines an abstract type that composes geometry from a set of nodes that define individual components. Composed geometry may have color, coordinates, normal and texture coordinates supplied. The rendered output of the combination of these is dependent on the concrete node definition. However, in general, the following rules shall be applied for all nodes:

- If the *color* field is not `NULL`, it shall contain an [X3DColorNode](#) node whose colours are applied to the vertices or faces of the *X3DComposedGeometryNode* as follows:
- If *colorPerVertex* is `FALSE`, colours are applied to each face. If *colorPerVertex* is true, colours are applied to each vertex.
- If the *color* field is `NULL`, the geometry shall be rendered normally using the material and texture defined in the [Appearance](#) node (see [12.2.2 Appearance node](#) for details).
- If *normalPerVertex* is `FALSE`, normals are applied to each face. If *normalPerVertex* is true, normals are applied to each vertex.
- If the *normal* field is not `NULL`, it shall contain a [Normal](#) node whose normals are applied to the vertices or faces of the *X3DComposedGeometryNode* in a manner exactly equivalent to that described above for applying colours to vertices/faces (where *normalPerVertex* corresponds to *colorPerVertex* and *normalIndex* corresponds to *colorIndex*).
- If the *normal* field is `NULL`, the browser shall automatically generate normals in accordance with the node's definition. If the node does not define a behaviour, the default is to generate an averaged normal for all faces that share that vertex.
- If the *texCoord* field is not `NULL`, it shall contain a [TextureCoordinate](#) node.

If the *attrib* field is not empty it shall contain a list of per-vertex attribute information for programmable shaders as specified in [32.2.2.4 Per-vertex attributes](#).

If the *fogCoord* field is not empty, it shall contain a list of per-vertex depth values for calculating fog depth as specified in [24.2.2.5 Fog colour calculation](#).

### 11.3.3 X3DCoordinateNode

```

X3DCoordinateNode : X3DGeometricPropertyNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}

```

This is the base node type for all coordinate node types in X3D. All coordinates are specified in nodes derived from this abstract node type.

### 11.3.4 X3DGeometricPropertyNode

```

X3DGeometricPropertyNode : X3DNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}

```

}

This is the base node type for all geometric property node types defined in X3D

### 11.3.5 X3DGeometryNode

```
X3DGeometryNode : X3DNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This is the base node type for all geometry in X3D.

### 11.3.6 X3DNormalNode

```
X3DNormalNode : X3DGeometricPropertyNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This is the base node type for all normal node types in X3D. All normals are specified in nodes derived from this abstract node type.

## 11.4 Node reference

### 11.4.1 ClipPlane

```
ClipPlane : X3DChildNode {
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFVec4f [in,out] plane 0 1 0 0 [0,1]
}
```

The ClipPlane node specifies a single plane equation that will be used to clip the geometry. The *plane* field specifies a four-component plane equation that describes the inside and outside half space. The first three components are a normalized vector describing the direction of the plane's normal direction.

### 11.4.2 Color

```
Color : X3DColorNode {
  MFColor [in,out] color [NULL] [0,1]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This node defines a set of RGB colours to be used in the fields of another node.

Color nodes are only used to specify multiple colours for a single geometric shape, such as colours for the faces or vertices of an [IndexedFaceSet](#). A material node is used to specify the overall material parameters of lit geometry. If both a material node and a *Color* node are specified for a geometric shape, the colours shall replace the *main color* component of the material.

The *main color* is defined as:

- *Material.diffuseColor*, if *PhongMaterial* is used.
- *PhysicalMaterial.baseColor*, if *PhysicalMaterial* is used.
- *UnlitMaterial.emissiveColor*, if *UnlitMaterial* is used.



This definition of *main color* here is consistent with the definition of *main texture* used in case of *Gouraud shading*. See [17.2.2.8 Gouraud shading](#).

RGB or RGBA textures take precedence over colours; specifying both an RGB or RGBA texture and a Color node for geometric shape will result in the Color node being ignored. RGB or RGBA textures are mixed with colors. Details on lighting equations can be found in [17.2.2 Lighting model](#).

### 11.4.3 ColorRGBA

```
ColorRGBA : X3DColorNode {
  MFColorRGBA [in,out] color [NULL] [0,1]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This node defines a set of RGBA colours to be used in the fields of another node.

RGBA color nodes are only used to specify multiple colours with alpha for a single geometric shape, such as colours for the faces or vertices of an [IndexedFaceSet](#). A material node is used to specify the overall material parameters of lit geometry. If both a material node and a *ColorRGBA* node are specified for a geometric shape, the colours shall replace the *main color* and *transparency* components of the material.

The *main color* is defined as:

- *Material.diffuseColor*, if *Phong Material* is used.
- *PhysicalMaterial.baseColor*, if *PhysicalMaterial* is used.
- *UnlitMaterial.emissiveColor*, if *UnlitMaterial* is used.

This definition of *main color* here is consistent with the definition of *main texture* used in case of *Gouraud shading* is used. See [17.2.2.8 Gouraud shading](#).

RGB or RGBA textures take precedence over colours; specifying both an RGB or RGBA texture and a ColorRGBA node for geometric shape will result in the ColorRGBA node being ignored. RGB or RGBA textures are mixed with colors. Details on lighting equations can be found in [17.2.2 Lighting model](#).

### 11.4.4 Coordinate

```
Coordinate : X3DCoordinateNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFVec3f [in,out] point [] (-∞,∞)
}
```

This node defines a set of 3D coordinates to be used in the *coord* field of vertex-based geometry nodes including:

- a. [IndexedFaceSet](#),
- b. [IndexedLineSet](#),
- c. [IndexedTriangleFanSet](#),
- d. [IndexedTriangleSet](#),
- e. [IndexedTriangleStripSet](#),
- f. [PointSet](#),
- g. [TriangleFanSet](#),

- h. [TriangleSet](#), and
- i. [TriangleStripSet](#).

### 11.4.5 IndexedLineSet

```

IndexedLineSet : X3DGeometryNode {
  MFInt32 [in]  set_colorIndex
  MFInt32 [in]  set_coordIndex
  MFNode [in,out] attrib [] [X3DVertexAttributeNode]
  SFNode [in,out] color  NULL [X3DColorNode]
  SFNode [in,out] coord  NULL [X3DCoordinateNode]
  SFNode [in,out] fogCoord  NULL [FogCoordinate]
  SFNode [in,out] metadata  NULL [X3DMetadataObject]
  SFNode [in,out] normal  NULL [X3DNormalNode]
  MFInt32 []   colorIndex [] [0,∞) or -1
  SFBool []   colorPerVertex TRUE
  MFInt32 []   coordIndex [] [0,∞) or -1
}

```

The `IndexedLineSet` node represents a 3D geometry formed by constructing polylines from 3D vertices specified in the `coord` field. `IndexedLineSet` uses the indices in its `coordIndex` field to specify the polylines by connecting vertices from the `coord` field. An index of "-1" indicates that the current polyline has ended and the next one begins. The last polyline may be (but does not have to be) followed by a "-1". `IndexedLineSet` is specified in the local coordinate system and is affected by the transformations of its ancestors.

The `coord` field specifies the 3D vertices of the line set and contains a [Coordinate](#) node.

Lines are not lit, are not texture-mapped, and do not participate in collision detection. The width and style of lines are determined by the line properties specified in an associated [Appearance](#) node. If no line properties are specified, the default values for fields of the [LineProperties](#) node shall be used (see [12.4.3 LineProperties](#)).

If the `color` field is not `NULL`, it shall contain a node derived from [X3DColorNode](#). The colours are applied to the line(s) as follows:

- a. If `colorPerVertex` is `FALSE`:
  1. If the `colorIndex` field is not empty, one colour is used for each polyline of the `IndexedLineSet`. There shall be at least as many indices in the `colorIndex` field as there are polylines in the `IndexedLineSet`. If the greatest index in the `colorIndex` field is  $N$ , there shall be  $N+1$  colours in the `Color` node. The `colorIndex` field shall not contain any negative entries.
  2. If the `colorIndex` field is empty, the colours from the `Color` node are applied to each polyline of the `IndexedLineSet` in order. There shall be at least as many colours in the `X3DColorNode` node as there are polylines.
- b. If `colorPerVertex` is `TRUE`:
  1. If the `colorIndex` field is not empty, colours are applied to each vertex of the `IndexedLineSet` in exactly the same manner that the `coordIndex` field is used to supply coordinates for each vertex from the `Coordinate` node. The `colorIndex` field shall contain at least as many indices as the `coordIndex` field and shall contain end-of-polyline markers (-1) in exactly the same places as the `coordIndex` field. If the greatest index in the `colorIndex` field is  $N$ , there shall be  $N+1$  colours in the `Color` node.
  2. If the `colorIndex` field is empty, the `coordIndex` field is used to choose colours from the `Color` node. If the greatest index in the `coordIndex` field is  $N$ , there



shall be  $N+1$  colours in the Color node.

If the *color* field is `NULL` and there is a material defined for the Appearance affecting this `IndexedLineSet`, the *emissiveColor* of the material shall be used to draw the lines. Details on lighting equations as they affect `IndexedLineSet` nodes are described in [17 Lighting component](#).

No effect on rendering behavior is defined for data specified by the *normal* field. Rendering techniques that utilize normal information to refine presentation of geometry are allowed but not required.

## 11.4.6 IndexedTriangleFanSet

```
IndexedTriangleFanSet : X3DComposedGeometryNode {
  MFInt32 [in]  set_index    [] [0,∞) or -1
  MFNode  [in,out] attrib   [] [X3DVertexAttributeNode]
  SFNode  [in,out] color    NULL [X3DColorNode]
  SFNode  [in,out] coord    NULL [X3DCoordinateNode]
  SFNode  [in,out] fogCoord  NULL [FogCoordinate]
  SFNode  [in,out] metadata  NULL [X3DMetadataObject]
  SFNode  [in,out] normal    NULL [X3DNormalNode]
  SFNode  [in,out] texCoord  NULL [X3DTextureCoordinateNode]
  SFBool  []    ccw         TRUE
  SFBool  []    colorPerVertex TRUE
  SFBool  []    normalPerVertex TRUE
  SFBool  []    solid       TRUE
  MFInt32 []    index       [] [0,∞) or -1
}
```

An `IndexedTriangleFanSet` represents a 3D shape composed of triangles that form a fan shape around the first vertex declared in each fan as depicted in [Figure 11.1](#).

`IndexedTriangleFanSet` uses the indices in its *index* field to specify the triangle fans by connecting vertices from the *coord* field. An index of "-1" indicates that the current fan has ended and the next one begins. The last fan may be (but does not have to be) followed by a "-1". Each fan shall have at least three non-coincident vertices.

The `IndexedTriangleFanSet` node is specified in the local coordinate system and is affected by the transformations of its ancestors. Descriptions of the *color*, *coord*, *normal*, and *texCoord* fields are provided in the [Color](#), [ColorRGBA](#), [Coordinate](#), [Normal](#), and [TextureCoordinate](#) nodes, respectively. If values are provided for the *color*, *normal* and *texCoord* fields, the values are applied in the same manner as the values from the *coord* field and there shall be at least as many values as are present in the *coord* field. The value of the *colorPerVertex* field is ignored and always treated as `TRUE`. If the *normal* field is not provided, normals shall be generated as follows:

- If *normalPerVertex* is `TRUE`, the normal for each vertex shall be the average of the normals for all triangles sharing that vertex.
- If *normalPerVertex* is `FALSE`, the normal shall be generated for the current triangle based on the *ccw* field.

The *solid* field determines whether the `IndexedTriangleFanSet` is visible when viewed from the inside. [11.2.3 Common geometry fields](#) provides a complete description of the *solid* field.

## 11.4.7 IndexedTriangleSet

```
IndexedTriangleSet : X3DComposedGeometryNode {
  MFInt32 [in]  set_index    [] [0,∞)
  MFNode  [in,out] attrib   [] [X3DVertexAttributeNode]
  SFNode  [in,out] color    NULL [X3DColorNode]
  SFNode  [in,out] coord    NULL [X3DCoordinateNode]
}
```

```

SFNode [in,out] fogCoord      NULL [FogCoordinate]
SFNode [in,out] metadata     NULL [X3DMetadataObject]
SFNode [in,out] normal       NULL [X3DNormalNode]
SFNode [in,out] texCoord     NULL [X3DTextureCoordinateNode]
SFBool  [] ccw              TRUE
SFBool  [] colorPerVertex   TRUE
SFBool  [] normalPerVertex  TRUE
SFBool  [] solid            TRUE
MFInt32 [] index           [] [0,∞)
}

```

The IndexedTriangleSet node represents a 3D shape composed of a collection of individual triangles as depicted in [Figure 11.2](#). IndexedTriangleSet uses the indices in its *index* field to specify the vertices of each triangle from the *coord* field. Each triangle is formed from a set of three vertices of the [Coordinate](#) node identified by three consecutive indices from the index field. If the *index* field does not contain a multiple of three coordinate values, the remaining vertices shall be ignored.

The IndexedTriangleSet node is specified in the local coordinate system and is affected by the transformations of its ancestors. Descriptions of the *color*, *coord*, *normal*, and *texCoord* fields are provided in the [Color](#), [ColorRGBA](#), [Coordinate](#), [Normal](#), and [TextureCoordinate](#) nodes, respectively. If values are provided for the *color*, *normal* and *texCoord* fields, the values are applied in the same manner as the values from the *coord* field and there shall be at least as many values as are present in the *coord* field. The value of the *colorPerVertex* field is ignored and always treated as `TRUE`. If the *normal* field is not supplied, normals shall be generated as follows:

- If *normalPerVertex* is `TRUE`, the normal at each vertex shall be the average of the normals for all triangles that share that vertex.
- If *normalPerVertex* is `FALSE`, the normal at each vertex shall be perpendicular to the face for that triangle.

The *solid* field determines whether the IndexedTriangleSet is visible when viewed from the backside. [11.2.3 Common geometry fields](#) provides a complete description of the *solid* field.

## 11.4.8 IndexedTriangleStripSet

```

IndexedTriangleStripSet : X3DComposedGeometryNode {
MFInt32 [in] set_index      [] [0,∞) or -1
MFNode  [in,out] attrib    [] [X3DVertexAttributeNode]
SFNode  [in,out] color     NULL [X3DColorNode]
SFNode  [in,out] coord     NULL [X3DCoordinateNode]
SFNode  [in,out] fogCoord  NULL [FogCoordinate]
SFNode  [in,out] metadata  NULL [X3DMetadataObject]
SFNode  [in,out] normal    NULL [X3DNormalNode]
SFNode  [in,out] texCoord  NULL [X3DTextureCoordinateNode]
SFBool  [] ccw            TRUE
SFBool  [] colorPerVertex  TRUE
SFBool  [] normalPerVertex TRUE
SFBool  [] solid          TRUE
MFInt32 [] index          [] [0,∞) or -1
}

```

An IndexedTriangleStripSet represents a 3D shape composed of strips of triangles as depicted in [Figure 11.3](#). IndexedTriangleStripSet uses the indices in its *index* field to specify the triangle strips by connecting vertices from the *coord* field. An index of "-1" indicates that the current strip has ended and the next one begins. The last strip may be (but does not have to be) followed by a "-1". Each strip shall have at least three non-coincident vertices.

The IndexedTriangleStripSet node is specified in the local coordinate system and is affected by the transformations of its ancestors. Descriptions of the *color*, *coord*,

*normal*, and *texCoord* fields are provided in the [Color](#), [ColorRGBA](#), [Coordinate](#), [Normal](#), and [TextureCoordinate](#) nodes, respectively. If values are provided for the *color*, *normal* and *texCoord* fields, the values are applied in the same manner as the values from the *coord* field and there shall be at least as many values as are present in the *coord* field. The value of the *colorPerVertex* field is ignored and always treated as `TRUE`. If the *normal* field is not supplied, normals shall be generated as follows:

- If *normalPerVertex* is `TRUE`, the normal shall be the average of all triangles sharing that vertex.
- If *normalPerVertex* is `FALSE`, the normal shall be generated for the triangle based on the *ccw* field.

The *solid* field determines whether the IndexedTriangleStripSet is visible when viewed from the inside. [11.2.3 Common geometry fields](#) provides a complete description of the *solid* field.

### 11.4.9 LineSet

```
LineSet : X3DGeometryNode {
  MFNode [in,out] attrib    [] [X3DVertexAttributeNode]
  SFNode [in,out] color     NULL [X3DColorNode]
  SFNode [in,out] coord     NULL [X3DCoordinateNode]
  SFNode [in,out] fogCoord  NULL [FogCoordinate]
  SFNode [in,out] metadata  NULL [X3DMetadataObject]
  SFNode [in,out] normal    NULL [X3DNormalNode]
  MFInt32 [in,out] vertexCount [] [2,∞)
}
```

The LineSet node represents a 3D geometry formed by constructing polylines from 3D vertices specified in the *coord* field.

The *color* field specifies the colour of the line set at each vertex and contains a node derived from [X3DColorNode](#). A description of the *color* field is provided in the color node. If the *color* field is `NULL` and there is a material defined for the [Appearance](#) affecting this LineSet, the *emissiveColor* of the material shall be used to draw the lines. Details on lighting equations as they affect LineSet nodes are described in [17 Lighting component](#)

The *coord* field specifies the 3D vertices of the line set and contains a [Coordinate](#) node.

The *vertexCount* field describes how many vertices are to be used in each polyline from the coordinate field. Coordinates are assigned to each line by taking *vertexCount*[*n*] vertices from the coordinate field. Each value of the *vertexCount* array shall be greater than or equal to two. It shall be an error to have a value less than two.

Lines are not lit, are not texture-mapped, and do not participate in collision detection. The width and style of lines are determined by the line properties specified in an associated Appearance node. If no line properties are specified, the default values for the fields of the [LineProperties](#) node shall be used (see [12.4.3 LineProperties](#)).

No effect on rendering behavior is defined for data specified by the *normal* field. Rendering techniques that utilize normal information to refine presentation of geometry are allowed but not required.

### 11.4.10 Normal

```

Normal : X3DNormalNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFVec3f [in,out] vector [] [-1,1]
}

```

This node defines a set of **3D surface normal** **3D direction vectors** to be used for the *normal* field of some geometry nodes (EXAMPLE [IndexedFaceSet](#), [IndexedLineSet](#), [LineSet](#), [PointSet](#), and [ElevationGrid](#)). The term 'normal' is common usage to indicate **direction vectors**, even though the direction vectors might not necessarily indicate **perpendicularity**. This node contains one multiple-valued field that contains the normal vectors. Normals shall be of unit length.

### 11.4.11 PointSet

```

PointSet : X3DGeometryNode {
  MFNode [in,out] attrib [] [X3DVertexAttributeNode]
  SFNode [in,out] color NULL [X3DColorNode]
  SFNode [in,out] coord NULL [X3DCoordinateNode]
  SFNode [in,out] fogCoord NULL [FogCoordinate]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFNode [in,out] normal NULL [X3DNormalNode]
}

```

The PointSet node specifies a set of 3D points, in the local coordinate system, with associated colours at each point. The *coord* field specifies a [Coordinate](#) node (or instance of a Coordinate node). The results are undefined if the *coord* field specifies any other type of node. PointSet uses the coordinates in order. If the *coord* field is `NULL`, the point set is considered empty.

PointSet nodes are not lit, not texture-mapped, nor do they participate in collision detection. The size of each point is implementation-dependent.

If the *color* field is not `NULL`, it shall specify a node derived from [X3DColorNode](#) that contains at least the number of points contained in the *coord* node. The results are undefined if the *color* field specifies any other type of node. Colours shall be applied to each point in order. The results are undefined if the number of values in the [X3DColorNode](#) node is less than the number of values specified in the Coordinate node.

If the *color* field is `NULL` and there is a material node defined for the [Appearance](#) node affecting this PointSet node, the *emissiveColor* of the material node shall be used to draw the points. More details on lighting equations can be found [17 Lighting component](#).

**No effect on rendering behavior is defined for data specified by the *normal* field. Rendering techniques that utilize normal information to refine presentation of geometry are allowed but not required.**

### 11.4.12 TriangleFanSet

```

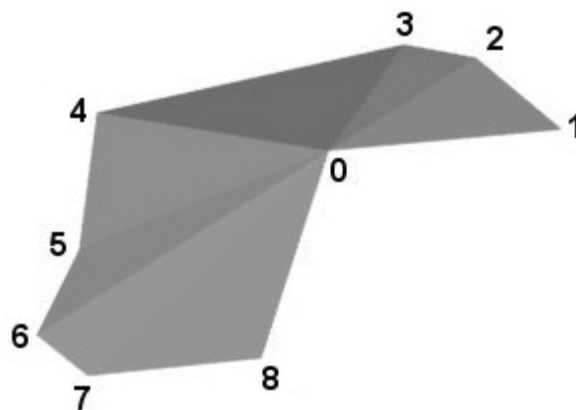
TriangleFanSet : X3DComposedGeometryNode {
  MFNode [in,out] attrib [] [X3DVertexAttributeNode]
  SFNode [in,out] color NULL [X3DColorNode]
  SFNode [in,out] coord NULL [X3DCoordinateNode]
  MFInt32 [in,out] fanCount [] [3,∞)
  SFNode [in,out] fogCoord NULL [FogCoordinate]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFNode [in,out] normal NULL [X3DNormalNode]
  SFNode [in,out] texCoord NULL [X3DTextureCoordinateNode]
  SFBool [] ccw TRUE
  SFBool [] colorPerVertex TRUE
  SFBool [] normalPerVertex TRUE
  SFBool [] solid TRUE
}

```

A `TriangleFanSet` represents a 3D shape composed of triangles that form a fan shape around the first vertex declared in each fan.

The `fanCount` field describes how many vertices are to be used in each fan from the coordinate field. Coordinates are assigned to each strip by taking `fanCount[n]` vertices from the coordinate field. Each value of the `fanCount` array shall be greater than or equal to three. It shall be an error to have a value less than three.

[Figure 11.2](#) displays a `TriangleFanSet` containing a single fan showing the ordering of the vertices for that fan.



**Figure 11.2 — TriangleFanSet node**

The `TriangleFanSet` node is specified in the local coordinate system and is affected by the transformations of its ancestors. Descriptions of the `color`, `coord`, `normal`, and `texCoord` fields are provided in the [Color/ColorRGBA](#), [Coordinate](#), [Normal](#), and [TextureCoordinate](#) nodes, respectively. If values are provided for the `color`, `normal`, and `texCoord` fields, there shall be at least as many values as are present in the `coord` field. The value of the `colorPerVertex` field is ignored and always treated as `TRUE`. If the normal field is not provided, for each fan, the normal shall be generated as follows: if `normalPerVertex` is `TRUE`, the normal shall be the average of all triangles within that fan sharing that vertex. For the vertex of the fan, the normal shall be the average of the contributions of all of the individual face normals. If `normalPerVertex` is `FALSE`, the normal shall be generated for the current triangle based on the `ccw` field.

The `solid` field determines whether the `TriangleFanSet` is visible when viewed from the inside. [11.2.3 Common geometry fields](#) provides a complete description of the `solid` field.

### 11.4.13 TriangleSet

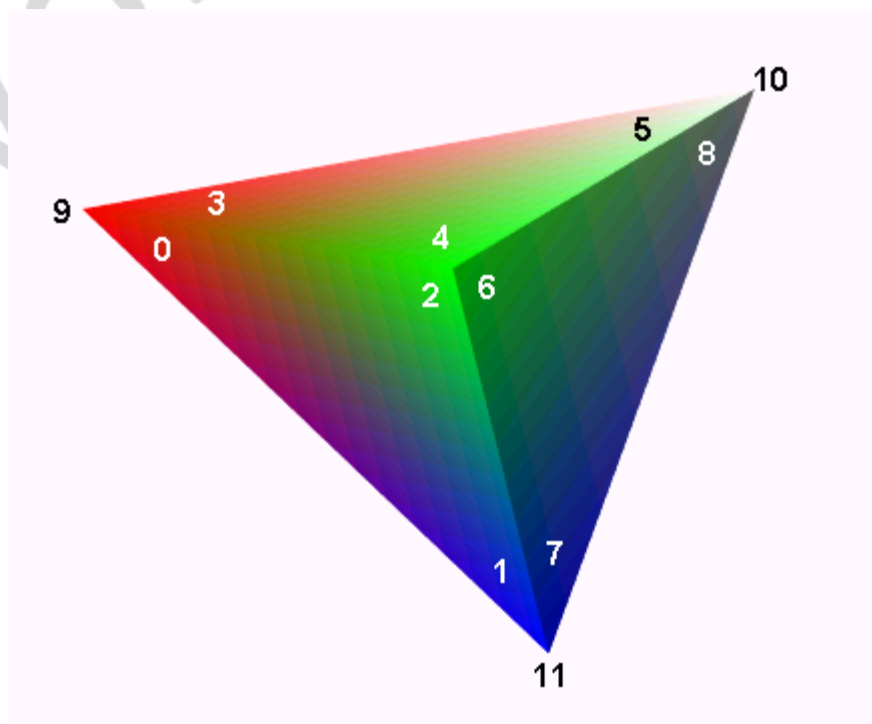
```
TriangleSet : X3DComposedGeometryNode {
  MFNode [in,out] attrib    [] [X3DVertexAttributeNode]
  SFNode [in,out] color     NULL [X3DColorNode]
  SFNode [in,out] coord     NULL [X3DCoordinateNode]
  SFNode [in,out] fogCoord  NULL [FogCoordinate]
  SFNode [in,out] metadata  NULL [X3DMetadataObject]
  SFNode [in,out] normal    NULL [X3DNormalNode]
  SFNode [in,out] texCoord  NULL [X3DTextureCoordinateNode]
  SFBool  [] ccw            TRUE
  SFBool  [] colorPerVertex TRUE
  SFBool  [] normalPerVertex TRUE
  SFBool  [] solid         TRUE
}
```



The TriangleSet node represents a 3D shape that represents a collection of individual triangles.

The *coord* field contains a [Coordinate](#) node that defines the 3D vertices that define the triangle. Each triangle is formed from a consecutive set of three vertices of the Coordinate node. If the Coordinate node does not contain a multiple of three coordinate values, the remaining vertices shall be ignored.

[Figure 11.3](#) depicts a TriangleSet node with several triangles. The ordering of the vertices is also shown.



**Figure 11.3 — TriangleSet node**

The TriangleSet node is specified in the local coordinate system and is affected by the transformations of its ancestors. Descriptions of the *color*, *coord*, *normal*, and *texCoord* fields are provided in the [Color/ColorRGBA](#), [Coordinate](#), [Normal](#), and [TextureCoordinate](#) nodes, respectively. If values are provided for the *color*, *normal*, and *texCoord* fields, there shall be at least as many values as are present in the *coord* field. The value of the *colorPerVertex* field is ignored and always treated as `TRUE`. If the *normal* field is not supplied, the normal shall be generated as perpendicular to the face for either version of *normalPerVertex*.

The *solid* field determines whether the TriangleSet is visible when viewed from the backside. [11.2.3 Common geometry fields](#) provides a complete description of the *solid* field.

### 11.4.14 TriangleStripSet

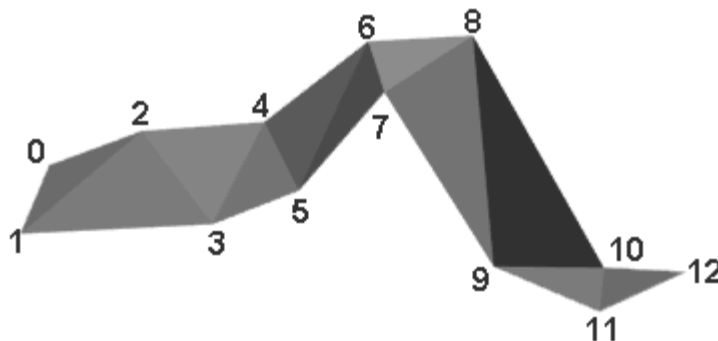
```
TriangleStripSet : X3DComposedGeometryNode {
  MFNode [in,out] attrib [] [X3DVertexAttributeNode]
  SFNode [in,out] color NULL [X3DColorNode]
  SFNode [in,out] coord NULL [X3DCoordinateNode]
  SFNode [in,out] fogCoord NULL [FogCoordinate]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFNode [in,out] normal NULL [X3DNormalNode]
```

```

MFInt32 [in,out] stripCount    [] [3,∞)
SFNode [in,out] texCoord      NULL [X3DTextureCoordinateNode]
SFBool [] ccw                 TRUE
SFBool [] colorPerVertex      TRUE
SFBool [] normalPerVertex     TRUE
SFBool [] solid                TRUE
    }
    
```

A `TriangleStripSet` represents a 3D shape composed of strips of triangles.

The `stripCount` field describes how many vertices are to be used in each strip from the coordinate field. Coordinates are assigned to each strip by taking `stripCount[i]` vertices from the coordinate field, where `i` is a sequential index of `stripCount`. Each value of the `stripCount` array shall be greater than or equal to three. It shall be an error to have a value less than three. [Figure 11.4](#) depicts a `TriangleStripSet` with a single triangle strip.



**Figure 11.4 — TriangleStripSet node**

The `TriangleStripSet` node is specified in the local coordinate system and is affected by the transformations of its ancestors. Descriptions of the `color`, `coord`, `normal`, and `texCoord` fields are provided in the [Color/ColorRGBA](#), [Coordinate](#), [Normal](#), and [TextureCoordinate](#) nodes, respectively. If values are provided for the `color`, `normal`, and `texCoord` fields, there shall be at least as many values as are present in the `coord` field. The value of the `colorPerVertex` field is ignored and always treated as `TRUE`. If the normal field is not provided, for each strip, the normal shall be generated as follows: if `normalPerVertex` is `TRUE`, the normal shall be the average of all triangles within that strip sharing that vertex. If `normalPerVertex` is `FALSE`, the normal shall be generated for the triangle based on the `ccw` field.

The `solid` field determines whether the `TriangleStripSet` is visible when viewed from the inside. [11.2.3 Common geometry fields](#) provides a complete description of the `solid` field.

## 11.5 Support levels

The Rendering component provides three levels of support as specified in [Table 11.2](#).

**Table 11.2 — Rendering component support levels**

Level	Prerequisites	Nodes/Features	Support
1	Core 1 Grouping 1		



		<i>X3DComposedGeometryNode</i> (abstract)	n/a
		<i>X3DGeometricPropertyNode</i> (abstract)	n/a
		<i>X3DGeometryNode</i> (abstract)	n/a
		<i>X3DColorNode</i> (abstract)	n/a
		<i>X3DCoordinateNode</i> (abstract)	n/a
		Color	All fields fully supported.
		ColorRGBA	Alpha value optionally supported.
		Coordinate	All fields fully supported.
		IndexedLineSet	<i>set_colorIndex</i> optionally supported. <i>set_coordIndex</i> optionally supported. <b>normal</b> optionally supported.
		LineSet	<b>normal</b> optionally supported.
		PointSet	<b>normal</b> optionally supported.
<b>2</b>	Core 1 Grouping 1		
		All Level 1 Rendering nodes	All fields as supported in Level 1.
		<i>X3DNormalNode</i> (abstract)	n/a
		Normal	All fields fully supported.
<b>3</b>	Core 1 Grouping 1		
		All Level 2 Rendering nodes	All fields fully supported except for ColorRGBA supported as in

			Level 2.
		IndexedTriangleFanSet	All fields fully supported.
		IndexedTriangleSet	All fields fully supported.
		IndexedTriangleStripSet	All fields fully supported.
		TriangleFanSet	All fields fully supported.
		TriangleSet	All fields fully supported.
		TriangleStripSet	All fields fully supported.
<b>4</b>	Core 1 Grouping 1		
		All Level 3 Rendering nodes	All fields as supported in Level 3.
		ColorRGBA	Alpha value fully supported.
<b>5</b>	Core 1 Grouping 1		
		All Level 4 Rendering nodes	All fields as supported in Level 4.
		ClipPlane	All fields fully supported.





## Extensible 3D (X3D) Part 1: Architecture and base components

### 32 CAD geometry component



#### 32.1 Introduction

##### 32.1.1 Name

The name of this component is "CADGeometry". The CADGeometry component provides X3D support for Computer-Aided Design (CAD) model geometry. This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 32.1.2 Overview

This clause describes the CADGeometry component of this part of ISO/IEC 19775. This includes how 3D geometry is specified and what shapes are available. [Table 32.1](#) provides links to the major topics in this clause.

**Table 32.1 — Topics**

- [32.1 Introduction](#)
  - [32.1.1 Name](#)
  - [32.1.2 Overview](#)
- [32.2 Concepts](#)
  - [32.2.1 Overview of CAD geometry](#)
  - [32.2.2 Product Structure Nodes](#)
  - [32.2.3 CAD layer relationships](#)
  - [32.2.4 Quad nodes](#)
  - [32.2.5 Common geometry fields](#)
- [32.3 Abstract Types](#)
  - [32.3.1 X3DProductStructureChildNode](#)
- [32.4 Node reference](#)
  - [32.4.1 CADAssembly](#)
  - [32.4.2 CADFace](#)
  - [32.4.3 CADLayer](#)
  - [32.4.4 CADPart](#)

- [32.4.5 IndexedQuadSet](#)
- [32.4.6 QuadSet](#)
- [32.5 Support levels](#)
- [Figure 32.1 — QuadSet](#)
- [Table 32.1 — Topics](#)
- [Table 32.2 — CADGeometry component support levels](#)

## 32.2 Concepts

### 32.2.1 Overview of geometry

The CADGeometry component consists of two types of nodes: product structure nodes and quad geometry nodes. Together, these node types are used to describe CAD specific data representations for X3D worlds.

### 32.2.2 Product structure nodes

Three nodes maintain CAD structural relationships. These nodes define the shape of a tangible object. Additional content may be grouped with these nodes using the CADLayer node. These nodes are, in hierarchy order:

- a. [CADAssembly](#) represents a product assembly composed of subassemblies and parts.
- b. [CADPart](#) is a physical object with a defined shape. It is composed of CADFace nodes which represent the spatial boundary of the object.
- c. [CADFace](#) contains a single Shape node defining one face of CADPart.

This hierarchy structures the file in a way that facilitates reuse of the CAD data in different domains.

### 32.2.3 CAD layer relationships

The CADLayer node maintains CAD layer relationships. CAD layers are used to visually and/or functionally organize geometric content.

The CADLayer node allows CADAssembly and CADPart nodes to be grouped together to model relationship beyond assembly structure. It also allows additional content beyond physical shape to be grouped with CADAssembly and CADPart nodes. This additional content may include:

- Text nodes containing annotations and dimension information.
- Shape nodes representing abstract geometric relations such as tolerance datums and features.

Add definition of datums as a spatial feature

### 32.2.4 Quad nodes

Quad nodes represent collections of planar quadrilateral polygons. The [IndexedQuadSet](#) node specifies the vertices using indices while the [QuadSet](#) node specifies the vertices directly.

## 32.2.5 Common geometry fields

Several 3D CADGeometry nodes share common fields to describe attributes. These fields that specify the vertex ordering and whether the shape is solid are named *ccw* and *solid* respectively. Common 3D geometry fields are described in [11.2.3 Common geometry fields](#).

## 32.3 Abstract types

### 32.3.1 X3DProductStructureChildNode

```
X3DProductStructureChildNode : X3DChildNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFString [in,out] name ""
}
```

The *X3DProductStructureChildNode* abstract node type marks nodes that are valid product structure children.

## 32.4 Node reference

### 32.4.1 CADAssembly

```
CADAssembly : X3DGroupingNode, X3DProductStructureChildNode {
  MFNode [in] addChildren
  MFNode [in] removeChildren
  MFNode [in,out] children [] [X3DProductStructureChildNode, X3DGroupingNode, X3DChildNode]
  SFBool [in out] bboxDisplay FALSE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFString [in,out] name ""
  SFBool [in out] visible TRUE
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}
```

The CADAssembly node holds a set of assemblies or parts grouped together.

The *children* field can contain [X3DProductStructureChildNode](#) types. Each child will be either a sub-assembly or a part.

The *children* field can contain [X3DChildNode](#) types.

The *name* field specifies the name of the CADAssembly.

### 32.4.2 CADFace

```
CADFace : X3DProductStructureChildNode, X3DBoundedObject {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFBool [in out] bboxDisplay FALSE
  SFString [in,out] name ""
  SFNode [in,out] shape NULL [X3DShapeNode, LODShape|LOD|Transform]
  SFBool [in out] visible TRUE
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}
```

The CADFace node holds the geometry representing a face of a part.

The *name* field specifies the name of the CADFace.

The *shape* field contains the [Shape](#) node providing the geometry and appearance for the face or an [LOD](#) node containing different detail levels of the shape. If an LOD node is provided, each child of the LOD node shall be a single Shape of varying complexity.

The *shape* field contains the [Shape](#) node providing the geometry and appearance for the face, or a [Transform](#) node relocating its children, or an [LOD](#) node containing different detail levels of the shape. If an LOD node is provided, each child of the LOD node shall be a single Shape of varying complexity or another Transform node. If a Transform node is provided, each child of the Transform node shall be a single Shape or another Transform or LOD node. In any case, only zero or one Shape under the CADFace node shall be active at any time.

### 32.4.3 CADLayer

```
CADLayer : X3DGroupingNode {
  MFNode [in] addChildren
  MFNode [in] removeChildren
  MFNode [in,out] children [] [X3DChildNode]
  SFBool [in out] bboxDisplay FALSE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFString [in,out] name ""
  MFBool [in,out] visible FALSE
  SFBool [in out] visible TRUE
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}
```

The CADLayer node defines a hierarchy of nodes used for showing layer structure for the CAD model.

The *name* field describes the content of the layer.

The *children* field contains all nodes defined for this layer.

The *visible* field specifies whether a particular child and its sub-children are visible. If the number of values is less than the number of children, the remaining children shall be visible.

### 32.4.4 CADPart

```
CADPart : X3DGroupingNode, X3DProductStructureChildNode {
  MFNode [in] addChildren
  MFNode [in] removeChildren
  SFVec3f [in,out] center 0 0 0 (-∞,∞)
  MFNode [in,out] children [] [CADFace]
  SFBool [in out] bboxDisplay FALSE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFString [in,out] name ""
  SFRotation [in,out] rotation 0 0 1 0 [-1,1] or (-∞,∞)
  SFVec3f [in,out] scale 1 1 1 (0,∞)
  SFRotation [in,out] scaleOrientation 0 0 1 0 [-1,1] or (-∞,∞)
  SFVec3f [in,out] translation 0 0 0 (-∞,∞)
  SFBool [in out] visible TRUE
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}
```

The CADPart node is a grouping node that defines a coordinate system for its children that is relative to the coordinate systems of its ancestors. See [4.3.5 Transformation hierarchy](#) and [4.3.6 Standard units and coordinate system](#) for a description of coordinate systems and transformations.

The CADPart node represents the location and faces that constitute a part in the CAD model.

[10.2.1 Grouping and children node types](#) provides a description of the *children*, *addChildren*, and *removeChildren* fields.

The *bboxCenter* and *bboxSize* fields specify a bounding box that encloses the children of the Part node. This is a hint that may be used for optimization purposes. The results are undefined if the specified bounding box is smaller than the actual bounding box of the children at any time. A default *bboxSize* value, (-1, -1, -1), implies that the bounding box is not specified and, if needed, shall be calculated by the browser. The bounding box shall be large enough at all times to enclose the union of the group's children's bounding boxes; it shall not include any transformations performed by the group itself (*i.e.*, the bounding box is defined in the local coordinate system of the children).

The *translation*, *rotation*, *scale*, *scaleOrientation* and *center* fields define a geometric 3D transformation consisting of (in order):

- a (possibly) non-uniform scale about an arbitrary point;
- a rotation about an arbitrary point and axis;
- a translation.

The *center* field specifies a translation offset from the origin of the local coordinate system (0,0,0). The *rotation* field specifies a rotation of the coordinate system. The *scale* field specifies a non-uniform scale of the coordinate system. The *scale* field may have values that are positive, negative (indicating a reflection), or zero. A value of zero indicates that any child geometry shall not be displayed. The *scaleOrientation* specifies a rotation of the coordinate system before the scale (to specify scales in arbitrary orientations). The *scaleOrientation* applies only to the scale operation. The *translation* field specifies a translation to the coordinate system.

Given a 3-dimensional point **P** and Part node, **P** is transformed into point **P'** in its parent's coordinate system by a series of intermediate transformations. In matrix transformation notation, where C (*center*), SR (*scaleOrientation*), T (*translation*), R (*rotation*), and S (*scale*) are the equivalent transformation matrices,

$$P' = T * C * R * SR * S * -SR * -C * P$$

The following Part node:

```
CADPart {
  center          C
  rotation        R
  scale           S
  scaleOrientation SR
  translation      T
  children        [...]
```

is equivalent to the nested sequence of:

```
CADPart {
  translation T
  children CADPart {
    translation C
    children CADPart {
      rotation R
      children CADPart {
        rotation SR
```



```

children CADPart {
  scale S
  children CADPart {
    rotation -SR
    children CADPart {
      translation -C
      children [...]
    }
  }
}
}
}
}
}

```

The name field documents the name of this part.

## 32.4.5 IndexedQuadSet

```

IndexedQuadSet : X3DComposedGeometryNode {
  MFInt32 [in] set_index [] [0,∞)
  MFNode [in,out] attrib [] [X3DVertexAttributeNode]
  SFNode [in,out] color NULL [X3DColorNode]
  SFNode [in,out] coord NULL [X3DCoordinateNode]
  SFNode [in,out] fogCoord NULL [FogCoordinate]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFNode [in,out] normal NULL [X3DNormalNode]
  SFNode [in,out] texCoord NULL [X3DTextureCoordinateNode]
  SFBool [] ccw TRUE
  SFBool [] colorPerVertex TRUE
  SFBool [] normalPerVertex TRUE
  SFBool [] solid TRUE
  MFInt32 [] index [] [0,∞)
}

```

The IndexedQuadSet node represents a 3D shape composed of a collection of individual quadrilaterals (quads) as depicted in [Figure 32.1](#). IndexedQuadSet uses the indices in its *index* field to specify the vertices of each quad from the *coord* field. Each quad is formed from a set of four vertices of the [X3DCoordinateNode](#) node identified by four consecutive indices from the index field. If the *index* field does not contain a multiple of four coordinate values, the remaining vertices shall be ignored.

The IndexedQuadSet node is specified in the local coordinate system and is affected by the transformations of its ancestors. Descriptions of the *color*, *coord*, *normal*, and *texCoord* fields are provided in the [X3DColorNode](#), [X3DCoordinateNode](#), [X3DNormalNode](#), and [X3DTextureCoordinateNode](#) nodes, respectively. If values are provided for the *color*, *normal* and *texCoord* fields, the values are applied in the same manner as the values from the *coord* field and there shall be at least as many values as are present in the *coord* field. The value of the *colorPerVertex* field is ignored and always treated as `TRUE`. If the *normal* field is not supplied, normals shall be generated as follows:

- If *normalPerVertex* is `TRUE`, the normal at each vertex shall be the average of the normals for all quads that share that vertex.
- If *normalPerVertex* is `FALSE`, the normal at each vertex shall be perpendicular to the face for that quad.

The *solid* field determines whether the IndexedQuadSet is visible when viewed from the inside. [11.2.3 Common geometry fields](#) provides a complete description of the *solid* field.

## 32.4.6 QuadSet

```

QuadSet : X3DComposedGeometryNode {
  MFNode [in,out] attrib [] [X3DVertexAttributeNode]
  SFNode [in,out] color NULL [X3DColorNode]
  SFNode [in,out] coord NULL [X3DCoordinateNode]
  SFNode [in,out] fogCoord NULL [FogCoordinate]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFNode [in,out] normal NULL [X3DNormalNode]
  SFNode [in,out] texCoord NULL [X3DTextureCoordinateNode]
  SFBool [] ccw TRUE
  SFBool [] colorPerVertex TRUE
}

```

```

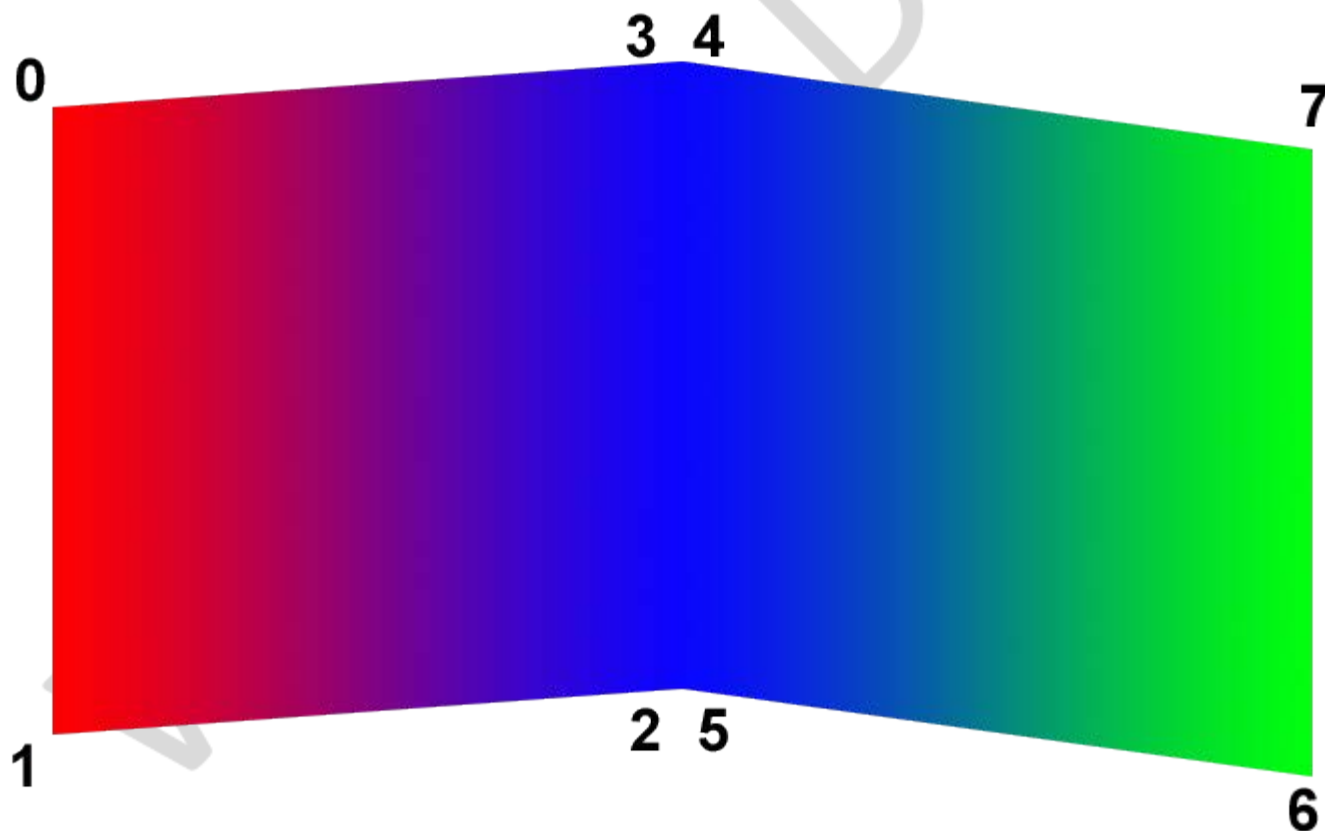
SFBool [] normalPerVertex TRUE
SFBool [] solid TRUE
}

```

The QuadSet node represents a 3D shape that represents a collection of individual planar quadrilaterals.

The *coord* field contains an [X3DCoordinateNode](#) node that defines the 3D vertices that define the quad. Each quad is formed from a consecutive set of four vertices of the coordinate node. If the coordinate node does not contain a multiple of four coordinate values, the remaining vertices shall be ignored.

[Figure 32.1](#) depicts a QuadSet node with two quads. The ordering of the vertices is also shown.



**Figure 32.1 — QuadSet node**

The QuadSet node is specified in the local coordinate system and is affected by the transformations of its ancestors. Descriptions of the *color*, *coord*, *normal*, and *texCoord* fields are provided in the [X3DColorNode](#), [X3DCoordinateNode](#), [X3DNormalNode](#), and [X3DTextureCoordinateNode](#) nodes, respectively. If values are provided for the *color*, *normal*, and *texCoord* fields, there shall be at least as many values as are present in the *coord* field. The value of the *colorPerVertex* field is ignored and always treated as `TRUE`. If the *normal* field is not supplied, the normal shall be generated as perpendicular to the face for either version of *normalPerVertex*.

The *solid* field determines whether the QuadSet is visible when viewed from the inside. [11.2.3 Common geometry fields](#) provides a complete description of the *solid* field.

## 32.5 Support levels

The CADGeometry component provides two levels of support as specified in [Table 32.2](#). Level 1 provides quad support. Level 2 adds support to describe product structure and layers.

**Table 32.2 — CADGeometry component support levels**

Level	Prerequisites	Nodes/Features	Support
1	Core 1 Grouping 1 Rendering 1 Shape 1		
		IndexedQuadSet	All fields fully supported.
		QuadSet	All fields fully supported.
2	Core 1 Grouping 1 Rendering 1 Shape 1		
		CADAssembly	All fields fully supported.
		CADFace	All fields fully supported.
		CADLayer	All fields fully supported.
		CADPart	All fields fully supported.





## Extensible 3D (X3D) Part 1: Architecture and base components

### Annex K

(normative)

## nVidia Cg shading language binding

---

### K.1 General

This annex defines the mapping of concepts of the Programmable shaders component to the nVidia Cg shading language (see [\[Cg\]](#)). It applies to the ProgramShader, ShaderProgram and PackagedShader nodes with the *language* field set to "Cg".

### K.2 Topics

[Table K.1](#) provides links to the major topics in this annex.

**Table K.1 — Topics**

- [K.1 General](#)
- [K.2 Topics](#)
- [K.3 Concepts](#)
  - [K.3.1 Rendering API support differences](#)
  - [K.3.2 Language strings](#)
- [K.4 Interaction with other nodes and components](#)
  - [K.4.1 Vertex shader](#)
    - [K.4.1.1 OpenGL profiles](#)
    - [K.4.1.2 Direct3D profiles](#)
  - [K.4.2 Fragment shader](#)
    - [K.4.2.1 OpenGL profiles](#)
    - [K.4.2.2 Direct3D profiles](#)
  - [K.4.3 LoadSensor](#)
  - [K.4.4 Vertex attributes](#)
    - [K.4.4.1 OpenGL profiles](#)

#### [K.4.4.2 Direct3D profiles](#)

- [K.5 Data type and parameter mappings](#)
  - [K.5.1 Node fields](#)
  - [K.5.2 X3D field types to Cg data types](#)
  - [K.5.3 X3D world state to Cg parameter names](#)
- [K.6 Event model](#)
  - [K.6.1 Changing URL fields](#)
  - [K.6.2 Changing the \*attrib\* field](#)
  - [K.6.3 Activating programs](#)
- [Table K.1 — Topics](#)
- [Table K.2 — Language string to Cg profile mapping](#)
- [Table K.3 — Supported Direct3D vertex declaration usage types](#)
- [Table K.4 — Mapping of X3D texture node types to Cg sampler types](#)
- [Table K.5 — Mapping of X3D material and light node types to Cg structure declarations](#)
- [Table K.6 — Mapping of X3D field types to Cg data types](#)
- [Table K.7 — Mapping of X3D world state to Cg parameter names](#)

## K.3 Concepts

### K.3.1 Rendering API support differences

The Cg language is a diverse set of shading capabilities that aim to support programmable shaders for a variety of APIs. This part of ISO/IEC 19775 supports the OpenGL and Microsoft Direct3D APIs. Programming APIs may express the same concepts in very distinctly different ways and the two cited APIs do so. Thus, a Cg shader program written to work on OpenGL will not work on Direct3D. Conversely, a Cg shader program written to work on Direct3D will not work on OpenGL.

Cg handles the incompatible code problem by defining *Cg profiles*. A Cg profile is a set of available shading language functionality. At the time the browser downloads the file, it can use the profile information to guide how to compile the code to the appropriate target. This annex defines its behaviour based on the Cg profile specified by the user.

### K.3.2 Language strings

Cg profile information is encoded as part of the language string of the ProgramShader node. All strings starting with "CG-" define behaviour defined in this annex. The part after the prefix defines the programming API and profile for which the Cg code shall be compiled. A browser thus may quickly distinguish which nodes to ignore and which to investigate further. The source files are referenced in the ShaderProgram nodes. This specification requires that the same profile is used for both the vertex and fragment programs.

[Table K.2](#) defines the mappings between the language string and the appropriate Cg profile. As Cg evolves, newer profiles may be defined and shall follow a similar naming convention.

**Table K.2 — Language string to Cg profile mapping**

Language string	Cg vertex shader profile	Cg fragment shader profile
CG_OPENGL_ARB	CG_PROFILE_ARBVP1	CG_PROFILE_ARBFP1
CG_OPENGL_NV30	CG_PROFILE_VP30	CG_PROFILE_VP30
CG_OPENGL_NV20	CG_PROFILE_VP20	CG_PROFILE_VP20
CG_D3D_SHADER_2.0	CG_PROFILE_VS_2_0	CG_PROFILE_PS_2_0
CG_D3D_SHADER_3.0	CG_PROFILE_VS_3_0	CG_PROFILE_PS_3_0
CG_D3D_SHADER_1.3	CG_PROFILE_VS_1_3	CG_PROFILE_PS_1_3

## K.4 Interaction with other nodes and components

### K.4.1 Vertex shader

#### K.4.1.1 OpenGL profiles

The vertex shader replaces the fixed functionality of the vertex processor. The OpenGL specification states that the following functionality is disabled if a vertex shader is supplied:

- a. The model view matrix is not applied to vertex coordinates.
- b. The projection matrix is not applied to vertex coordinates.
- c. The texture matrices are not applied to texture coordinates.
- d. The normals are not transformed to eye coordinates.
- e. The normals are not rescaled or normalized.
- f. Texture coordinates are not generated automatically.
- g. Per-vertex lighting is not performed.
- h. Color material lighting is not performed.
- i. Point size distance attenuation is not performed.

#### K.4.1.2 Direct3D profiles

In Cg language Direct3D profiles, the vertex shader replaces the vertex processing done by the Microsoft Direct3D graphics pipeline. While using a vertex shader, state information regarding transformation and lighting operations is ignored by the fixed-function pipeline. The Direct3D specification states that the following functionality is disabled if a vertex shader is supplied:

- a. The model view matrix is not applied to vertex coordinates.
- b. The projection matrix is not applied to vertex coordinates.

- c. The texture matrices are not applied to texture coordinates.
- d. The normals are not transformed to eye coordinates.
- e. The normals are not rescaled or normalized.
- f. Texture coordinates are not generated automatically.
- g. Per-vertex lighting is not performed.
- h. Color material lighting is not performed.
- i. Point size distance attenuation is not performed.

The fixed-function pipeline Direct3D graphics state is not available for use within a Cg shader program. Shaders that wish to make use of this data, such as material, lighting, texture and transformation matrix state, shall declare parameters of the appropriate type and pass values into them via declared fields of the containing ShaderProgram node in the X3D scene graph. The parameter types and mappings to those types from built-in X3D values are defined in [K.4 Data type and parameter mappings](#).

## K.4.2 Fragment shader

### K.4.2.1 OpenGL profiles

The fragment shader replaces the fixed functionality of the fragment processor. The OpenGL specification states that the following functionality is disabled if a fragment shader is supplied:

- a. Textures are not applied.
- b. Fog is not applied.

### K.4.2.2 Direct3D profiles

In Cg language Direct3D profiles, the fragment shader, also known as a *pixel* shader in Cg, replaces the fixed functionality of the Direct3D fragment processor. The Direct3D specification states that “textures are not applied” if a fragment shader is supplied.

The fixed function pipeline Direct3D graphics state is not available for use within a Cg pixel shader program. Shaders that wish to make use of this data, such as material, lighting, texture and transformation matrix state, shall declare parameters of the appropriate type and pass values into them via declared fields of the containing ShaderProgram node in the X3D scene graph. The parameter types and mappings to those types from built-in X3D values are defined in [K.4 Data Type and Parameter Mappings](#).

## K.4.3 LoadSensor

The LoadSensor node (See [9.4.3 LoadSensor](#)) has two output fields *isActive* and *isLoaded*. The *isLoaded* field behaviour is unchanged.

The *isActive* field is defined to issue a `TRUE` event when all the following conditions have been satisfied:

- a. The content identified by the *url* field has been successfully loaded.



- b. The shader program has been successfully compiled without error.

## K.4.4 Vertex attributes

### K.4.4.1 OpenGL profiles

Each vertex attribute node directly maps the *name* field to the uniform variable of the same name. If the name is not available as a uniform variable in the provided shader source, the values of the node shall be ignored.

The browser implementation shall automatically assign appropriate internal index values for each attribute

### K.4.4.2 Direct3D profiles

In Cg language Direct3D profiles, each vertex attribute node directly maps the *name* field to a Direct3D usage type for use within a Direct3D vertex declaration (with the prefix "D3DDECLUSAGE\_" prepended to the name), as well as a Cg binding semantic of the same name defined on the varying inputs to a shader program. This language binding allows the use of the predefined Direct3D vertex declaration usage types and Cg binding semantics listed in [Table K.3](#). If the name cannot be interpreted as a valid Direct3D usage type or Cg binding semantic, the values of the node shall be ignored.

**Table K.3 — Supported Direct3D vertex declaration usage types**

Direct3D usage type
<i>POSITION</i>
<i>NORMAL</i>
<i>TEXCOORD</i>
<i>TANGENT</i>
<i>BINORMAL</i>
<i>COLOR</i>
<i>FOG</i>

The browser implementation shall automatically assign appropriate internal index values for each attribute in the case where multiple nodes are defined having the same value in the *name* field.

## K.5 Data Type and Parameter Mappings

### K.5.1 Node fields

Fields that are of type SFNode/MFNode are ignored unless the value is of type *X3DTextureNode*, or in Direct3D profiles, *X3DMaterialNode*, or *X3DLightNode*. Field instances of type *X3DTextureNode* are mapped according to the appropriate Direct3D or OpenGL sampler data type. The mappings from texture nodes to built-in sampler types are defined in [Table K.4](#).

**Table K.4 — Mapping of X3D texture node types to OpenGL or Direct3D sampler types**

X3D texture type	OpenGL variable type	Direct3D variable type
<i>X3DTexture2DNode</i>	sampler2D	sampler2D
<i>X3DTexture3DNode</i>	sampler3D	sampler3D
<i>X3DEnvironmentTextureNode</i>	samplerCube	samplerCube

X3D does not define mappings to the OpenGL types sampler1D, sampler1DShadow and sampler2DShadow or the Direct3D types sampler1D, sampler1DShadow and sampler2DShadow.

In Cg language OpenGL profiles, the current geometry and pipeline state is exposed through the built-in variable *glstate*.

In Cg language Direct3D profiles, field instances of type *X3DMaterialNode* and *X3DLightNode* are mapped to structures that shall be declared in the shader program as defined in [Table K.5](#).

**Table K.5 — Mapping of X3D material and light node types to Cg structure declarations (Direct3D profiles only)**

X3D node type	Cg structure declaration	Additional information
<i>X3DMaterialNode</i>	<pre>struct X3DMaterial {     float4     diffuseColor;     float4     ambientColor;     float4     specularColor;     float4     emissiveColor;     float power; };</pre>	All color values are 4-component with alpha value = 1.0.
<i>X3DLightNode</i>	<pre>struct X3DLight {     int type;     float4     diffuseColor;     float4     specularColor;     float4     ambientColor;     point3     position;     point3     direction;     float range; };</pre>	<p>Valid <i>type</i> member values are 1 for Point light, 2 for Spot light and 3 for Direction light.</p> <p>All color values are 4-component with alpha value = 1.0.</p>

<pre> float falloff; float attenuation0; float attenuation1; float attenuation2; float theta; float phi; bool on; }; </pre>	<p>All position, direction and scalar values are assumed to be in world space.</p> <p>The <i>on</i> member specifies whether the light is enabled.</p>
---	--

## K.5.2 X3D field types to Cg data types

[Table K.6](#) indicates how the X3D field types shall be mapped to data types used in the Cg Language.

**Table K.6 — Mapping of X3D field types to Cg data types**

X3D Field type	Cg Data Type
SFBool	bool
MFBool	bool[]
MFInt32	float[]
SFInt32	float
SFFloat	float
MFFloat	float[]
SFDouble	double
MFDouble	double[]
SFTime	double
MFTime	double[]
SFNode	See <a href="#">K.4.1 Node fields</a>
MFNode	See <a href="#">K.4.1 Node fields</a>
SFVec2f	float2
MFVec2f	float2[]
SFVec3f	float3
MFVec3f	float3[]
SFVec4f	float4
MFVec4f	float4[]

SFVec3d	float3
MFVec3d	float3[]
SFVec4d	float4
MFVec4d	float4[]
SFRotation	float4
MFRotation	float4[]
MFColor	float4[]
SFColor	float4
SFImage	int[]
MFImage	int[]
SFString	Not supported
MFString	Not supported
SFMatrix3f	float3x3
MFMatrix3f	float3x3[]
SFMatrix4f	float4x4
MFMatrix4f	float4x4[]

Cg defines maximum supported lengths of each array data type, which may conflict with the minimum support requirements for X3D.

### K.5.3 X3D world state to Cg parameter names

In Cg language Direct3D profiles, certain internal states of the X3D world, such as transformation matrices, or the viewer's position in world space, are neither readily available via the Cg shader program or directly accessible from the X3D scene graph. Thus if used, these world state values shall be explicitly passed in to the shader program as named parameters. This binding defines an automatic mapping of these states to predefined shader program parameter names. [Table K.7](#) specifies the mapping of internal states of the X3D world to parameter names used in the Cg programs.

**Table K.7 — Mapping of X3D world state to Cg parameter names (Direct3D profiles only)**

Parameter name	Description
	This name refers to the matrix transforming from local to global coordinates. The model matrix transforms

<b>model</b>	vertices from their model position to their position in world space ( <i>i.e.</i> , after the effects of all Transform nodes have been applied).
<b>view</b>	This name refers to the viewing matrix transforming from world to view relative coordinates.
<b>projection</b>	This name refers to the projection matrix transforming from viewing relative coordinates to clip space, including the projective part.
<b>modelView</b>	This name refers to the matrix that represents the concatenation of model and view matrices. This matrix transforms vertices from their model position to their position in view space ( <i>i.e.</i> , after the effects of all Transform nodes and the current viewpoint have been applied).
<b>modelViewProjection</b>	This name refers to the matrix that represents the concatenation of model, view and projection matrices. This matrix transforms vertices from their model position to their final position in clip space.
<b>viewPosition</b>	This name refers to the current viewer position in world space coordinates.

The following suffixes can be applied to the matrix built-in values. A suffix of *I* signifies the inverse of the matrix. *T* signifies the transpose of the matrix. *IT* signifies the inverse transpose of the matrix.

## K.6 Event model

### K.6.1 Changing URL fields

When the *url* receives an event changing the value, the browser shall immediately attempt to download the new source. Upon successful download, the browser shall attempt to compile the new source and issue the appropriate LoadSensor events. It shall not automatically activate the shader program, nor disable the currently running shader.

Values defined at load time of the file do not require an explicit request to activate the shader program. It shall be assumed to automatically activate the program once all the objects have successfully downloaded. If some of the shader source files are not downloaded or compiled (*e.g.*, due to errors), no activation shall occur for the shader program.

### K.6.2 Changing the *attrib* field

Per-vertex attributes may be defined as one of the fields of *X3DComposedGeometryNode*. These may be changed at runtime by adding or removing node instances. Adding new node instances to the field shall require that the user request an explicit activate in order to make them visible to the shader.

### K.6.3 Activating programs

The user may, at any time, request that the browser activate the composing shader objects by sending a `TRUE` value to the *activate* inputOnly field of the ProgramShader or PackagedShader node. Users may need to force a re-activation of the node under various circumstances, such as changing the *url* field of one or more ShaderProgram or PackagedShader nodes, or adding or removing ShaderProgram nodes from the *programs* field of the ProgramShader node. Reactivating the shader shall replace the existing shader with the new compiled shader for subsequent rendering.





## Extensible 3D (X3D) Part 1: Architecture and base components

### 12 Shape component



#### 12.1 Introduction

##### 12.1.1 Name

The name of this component is "Shape". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 12.1.2 Overview

This clause describes the Shape component of this part of ISO/IEC 19775. The Shape component defines nodes for associating geometry with their visible properties and the scene environment. [Table 12.1](#) provides links to the major topics in this clause.

**Table 12.1 — Topics**

- [12.1 Introduction](#)
  - [12.1.1 Name](#)
  - [12.1.2 Overview](#)
- [12.2 Concepts](#)
  - [12.2.1 Shape characteristics](#)
  - [12.2.2 Appearance characteristics](#)
  - [12.2.3 Two-sided materials](#)
  - [12.2.4 Texture mapping specified in material nodes](#)
    - [12.2.4.1 Texture coordinates](#)
    - [12.2.4.2 Texture coordinates transformation](#)
  - [12.2.5 Coexistence of textures specified in material nodes with the "Appearance.texture" field](#)
- [12.3 Abstract types](#)
  - [12.3.1 X3DAppearanceChildNode](#)
  - [12.3.2 X3DAppearanceNode](#)
  - [12.3.3 X3DMaterialNode](#)
  - [12.3.4 X3DOneSidedMaterialNode](#)
  - [12.3.5 X3DShapeNode](#)



- [12.4 Node reference](#)
  - [12.4.1 AcousticProperties](#)
  - [12.4.2 Appearance](#)
  - [12.4.3 FillProperties](#)
  - [12.4.4 LineProperties](#)
  - [12.4.5 Material](#)
  - [12.4.6 PhysicalMaterial](#)
  - [12.4.7 PointProperties](#)
  - [12.4.8 Shape](#)
  - [12.4.9 TwoSidedMaterial](#)
  - [12.4.10 UnlitMaterial](#)
- [12.5 Support levels](#)
- [Figure 12.1 — Effects of two-sided materials on geometry](#)
- [Table 12.1 — Topics](#)
- [Table 12.2 — International register of items hatchstyles](#)
- [Table 12.3 — International register of items linetypes](#)
- [Table 12.4 — Shape component support levels](#)

## 12.2 Concepts

### 12.2.1 Shape characteristics

The [Shape](#) node associates a geometry node with nodes that define that geometry's appearance. Shape nodes shall be part of the transformation hierarchy to have any visible result, and the transformation hierarchy shall contain Shape nodes for any geometry to be visible (the only nodes that render visible results are Shape nodes and the background nodes described in [24 Environmental effects](#)). A Shape node contains exactly one geometry node in its *geometry* field, which is of type *X3DGeometryNode*. The Shape node descends from the abstract base type *X3DShapeNode*.

### 12.2.2 Appearance characteristics

Shape nodes may specify an [Appearance](#) node that describes the appearance properties (material, texture and texture transformation) to be applied to the Shape's geometry. All valid children of the Appearance node descend from the abstract base type [X3DAppearanceChildNode](#).

Nodes of the following types may be specified in the *material* field of the Appearance node:

- [Material](#)
- [PhysicalMaterial](#)
- [TwoSidedMaterial](#) (deprecated)
- [UnlitMaterial](#)

This set of nodes may be extended by creating new nodes derived from the

[X3DMaterialNode](#) abstract base type.

Nodes of the following types may be specified in the *backMaterial* field of the Appearance node:

- [Material](#)
- [PhysicalMaterial](#)
- [UnlitMaterial](#)

This set may be extended by creating new nodes derived from the [X3DOneSidedMaterialNode](#) abstract base type.

The Appearance node specifies texture mapping in its *texture* field. Valid values of the texture field are descendants of [X3DTextureNode](#), including:

- [ImageTexture](#)
- [PixelTexture](#)
- [MovieTexture](#)
- [MultiTexture](#)

This set may be extended by creating new nodes derived from the abstract [X3DTextureNode](#) base class as defined in [18.3.2 X3DTextureNode](#).

Nodes of the type [X3DTextureTransformNode](#) may be specified in the *textureTransform* field of the Appearance node (see [18.3.4 X3DTextureTransformNode](#)), including:

- [TextureTransform](#)

Interaction between the appearance properties and properties specific to geometry nodes are described in [13 Geometry3D component](#) and [14 Geometry2D component](#).

An Appearance node may specify additional information about the appearance of the corresponding geometry. The *acousticProperties* field can be provided by an [AcousticProperties](#) node. Special properties may be defined for lines and filled areas. These properties are defined in the *lineProperties*, *fillProperties* and *pointProperties* fields by the following nodes, respectively:

- [LineProperties](#)
- [FillProperties](#)
- [PointProperties](#)

### 12.2.3 Two-sided materials

A polygon defines a front face based on the direction of the normal. That normal is either explicitly provided by the end user or implicitly calculated by the browser based on the winding rules for the node (see the *ccw* field on many of the polygonal geometry nodes such as [IndexedFaceSet](#)).

The [TwoSidedMaterial](#) node provides a way to render the front and back sides of the polygon with different material properties. [Figure 12-1](#) depicts the effects of the [TwoSidedMaterial](#) node.

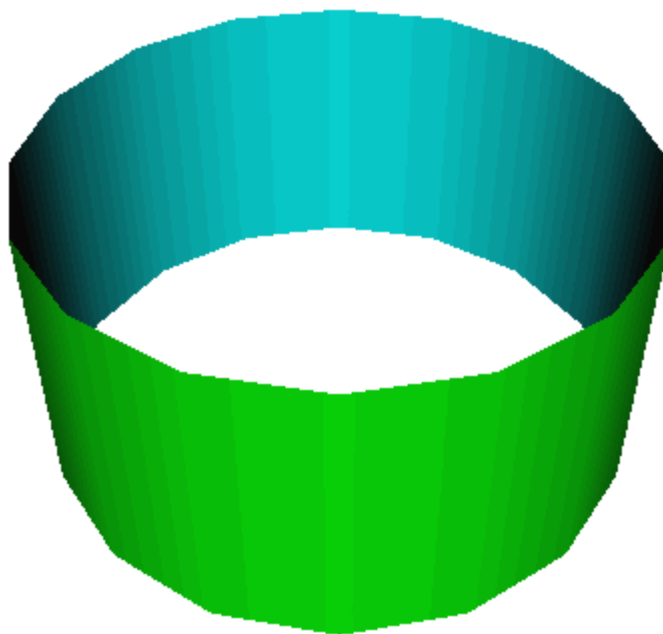
The *solid* field, described in the [11.2.3 Common geometry fields](#), controls whether the geometry is visible from the back side.

- When *solid* is `TRUE`, the back faces are not visible at all.
- When *solid* is `FALSE`, the back faces of the geometry are lit, using an inverted normal vector than the corresponding front faces.

Using the [Appearance](#) field *backMaterial* allows rendering the front and back sides of the polygon with different material properties. It is meaningful only when *solid* is `FALSE`, since otherwise the back faces are never rendered. When the *solid* is `TRUE`, the value of *backMaterial* has no effect on rendering.

[TwoSidedMaterial](#) (deprecated) provides a similar functionality (through its *separateBackColor* field) but limited only to the Phong lighting model.

[Figure 12.1](#) depicts the effects of the [Appearance](#) field *backMaterial* node.



**Figure 12.1 — Effects of different material properties on front and back sides of the geometry**

Several constraints pertain to the *backMaterial* field value, to make the definition reasonable and easy to implement by the browsers.

- The *backMaterial* field can have a value different than `NULL` only when the *material* field also has a value different than `NULL`.
- It is not allowed to provide a *backMaterial* when the *material* specifies a [TwoSidedMaterial](#) (deprecated).
- When both the *material* and *backMaterial* are provided (not `NULL`), it is required that they:
  1. Specify the same node class. In other words, both of them should be [Material](#), or both should be [PhysicalMaterial](#), or both should be [UnlitMaterial](#).

2. Use the same textures with the same mapping. In other words, the values of the fields `xxxTexture` and `xxxTextureMapping` documented in [12.2.4 Texture mapping specified in material nodes](#) shall be equal for both front and back materials. The author should use the DEF / USE mechanism to have the same references to texture nodes.

In effect, the front and back material parameters may differ only in their scalar or vector values. For example, front side may have a different *diffuseColor* than the back side.

To summarize, the following combinations are allowed:

- Both *material* and *backMaterial* are `NULL`. In this case we have an unlit (pure white) material, when viewed from both the front and back side.

This case is *exactly equivalent* to using an [UnlitMaterial](#) with default *emissiveColor* white for both *material* and *backMaterial*.

- *material* is a [TwoSidedMaterial](#) node, *backMaterial* is `NULL`.

This case is only provided for compatibility. The [TwoSidedMaterial](#) is deprecated since X3D 4.0. Whether the rendering parameters are the same, or different, for front and back sides is determined by the *separateBackColor* field of the relevant [TwoSidedMaterial](#) node.

- *material* contains a [Material](#), [PhysicalMaterial](#) or [UnlitMaterial](#) node, and *backMaterial* is `NULL`.

In this case, both front and back sides will be rendered with the same parameters. This case is *exactly equivalent* to reusing the same material node (through DEF / USE mechanism) for both *material* and *backMaterial* fields.

- *material* contains a [Material](#), [PhysicalMaterial](#) or [UnlitMaterial](#) node, and *backMaterial* also contains a node of the same type.

In this case, some front and back material parameters may differ.

## 12.2.4 Texture mapping specified in material nodes

The [X3DOneSidedMaterialNode](#) and descendants ([Material](#), [PhysicalMaterial](#), [UnlitMaterial](#)) introduce a number of fields to modify material parameters using textures. They are consistently defined by a pair of fields like this:

```
SFNode [in,out] xxxTexture NULL
SFString [in,out] xxxTextureMapping ""
```

The field `xxxTexture` indicates a texture node.

The `xxxTextureMapping` determines the *texture coordinates* and *texture coordinate transformation* for given texture `xxxTexture`.

The corresponding texture coordinate and texture coordinate transformation nodes have a field *mapping* that will match the value of the `xxxTextureMapping` field. See the [X3DSingleTextureCoordinateNode](#) and [X3DSingleTextureTransformNode](#) definitions.

Multiple textures may use the same texture coordinates and their transformations. For example, it is common that both `normalTextureMapping` and `diffuseTextureMapping` are equal, if the graphic artist prepared both `normalTexture` and `diffuseTexture` simultaneously, assuming the same mapping.

### 12.2.4.1 Texture coordinates

Let's define a *list of texture coordinates* for each geometry node like this:

- If the geometry field doesn't have a *texCoord* field then this list is empty.

Most geometric objects with a predefined geometry (e.g. *Sphere*) don't have *texCoord* field. Most geometric objects with geometry defined by the author (e.g. all the nodes derived from [X3DComposedGeometryNode](#)) have *texCoord* field.

- Otherwise, if the value of the *texCoord* field is `NULL`, then this list is again empty.
- Otherwise, if the value of the *texCoord* field is a single node derived from [X3DSingleTextureCoordinateNode](#), then place this one node on the list.

[X3DSingleTextureCoordinateNode](#) includes all texture coordinate nodes (like [TextureCoordinate](#) or [TextureCoordinateGenerator](#)) except [MultiTextureCoordinate](#).

- Otherwise, the value of this field must be [MultiTextureCoordinate](#) node. Then use the *MultiTextureCoordinate.texCoord* contents list as our *list of texture coordinates*.

Note: The above definition means that using a [MultiTextureCoordinate](#) with exactly one child is equivalent to using this child directly. This is a general rule in X3D 4.0, see also the [MultiTextureCoordinate](#) specification for details and an example.

All the [X3DSingleTextureCoordinateNode](#) nodes on the *list of texture coordinates* defined above must have a different *mapping* value. An exception is the empty *mapping* value, which may occur many times.

If the *xxxTextureMapping* field is not empty, it must refer to a corresponding [X3DSingleTextureCoordinateNode](#) node on a *list of texture coordinates*.

If the *xxxTextureMapping* field is empty, then the **first** item on a *list of texture coordinates* is used (regardless of its *mapping* value). Only if no such texture coordinate exists (the list is empty), then the *default texture coordinates* for the specific geometry node are used.

The algorithm to perform the *default texture coordinate calculation* is described at each geometry node. For example [IndexedFaceSet](#) determines the coordinates based on the local bounding box sizes, [Box](#) has the texture applied 6 times on 6 faces etc.

Hint for implementations: This section makes an important guarantee. Generating *default texture coordinates* only needs to be done when the *texCoord* field of the geometry is empty, or contains an empty *MultiTextureCoordinate* node. In all other cases, you know that *default texture coordinates* are not necessary, because all textures will use one of the coordinates in the *texCoord* list.

This is an important property, because browsers may want to avoid generating *default texture coordinates* as it is a time-consuming process (e.g. requires to iterate over vertexes at least twice in case of *IndexedFaceSet*) and often not necessary (models exported by 3D authoring software typically have all texture coordinates provided in the file).

So we wanted to enable this optimization, and make it easy to detect looking only at *texCoord* contents. In effect, it doesn't matter what [Appearance](#) or material will be used with this geometry node — you can easily avoid most cases when *default texture coordinates* would be unused just by inspecting the geometry *texCoord* value.

### 12.2.4.2 Texture coordinates transformation

Let's define a *list of texture transformations* for each geometry node like this:

- If the shape uses no [Appearance](#) node then this list is empty.
- Otherwise, if the value of *Appearance.textureTransform* is NULL, then this list is again empty.
- Otherwise, if the value of *Appearance.textureTransform* is a single node [X3DSingleTextureTransformNode](#), then place this one node in the list.

Most texture transformation nodes are derived from [X3DSingleTextureTransformNode](#), like [TextureTransform](#) and [TextureTransform3D](#). But not [MultiTextureTransform](#).

- Otherwise, the value of *Appearance.textureTransform* must be [MultiTextureTransform](#). Then use the *MultiTextureTransform.textureTransform* contents as our *list of texture transformations*.

Note that we treat a [MultiTextureTransform](#) with a single child always the same as using this child directly. This is a general rule in X3D 4.0, see also the [MultiTextureTransform](#) specification for details and an example.

If the *xxxTextureMapping* field is not empty, it must refer to a corresponding *X3DSingleTextureTransformNode* node within the *list of texture transformations*. The *X3DSingleTextureTransformNode* node must have equal *mapping* value.

If the *xxxTextureMapping* is an empty string, then the **first** item on a *list of texture transformations* is used (regardless of its *mapping* value). If the list is empty, no texture transformation is used.

Note: Throughout this section, we treat empty string as a special case for *xxxTextureMapping* and *mapping* fields. Such mapping names are allowed (they are even the default) but they do not constitute a "match". It would be error-prone if two empty mapping values would match, as it's easy to use them accidentally, since they are the default field values. Instead, empty *xxxTextureMapping* just indicates "use the first coordinates / transformations" — this is simplest and most natural.

### 12.2.5 Coexistence of textures specified in material nodes

## with the "Appearance.texture" field

In X3D 4.0, models can specify textures using the *xxxTexture* fields inside the various [X3DOneSidedMaterialNode](#) descendants. This allows to control every material parameter by a different texture.

Alternatively, models can also use the mechanism known from X3D 3.x, and provide a texture inside the *Appearance.texture* field. This is also the only way to use the [MultiTexture](#) node (which cannot be placed in *xxxTexture* fields, as it would make implementation complicated).

The exact behavior of *MultiTexture* node and *Appearance.texture* is this:

1. If the *Appearance.material* is [Material](#), and the *Material.diffuseTexture* is `NULL`, then *Appearance.texture* affects the *diffuseParameter* for the lighting equation.

In a way, *Appearance.texture* performs then the role of *Material.diffuseTexture*. It can even use *MultiTexture* to calculate the diffuse parameter by a composition (e.g. addition or multiplication) of other textures.

The *Material.diffuseTextureMapping* value doesn't matter in this case.

2. If the *Appearance.material* is [PhysicalMaterial](#), and the *PhysicalMaterial.baseTexture* is `NULL`, then *Appearance.texture* affects the *baseParameter* for the lighting equation.

The *PhysicalMaterial.baseTextureMapping* value doesn't matter in this case.

3. If the *Appearance.material* is [UnlitMaterial](#), and the *UnlitMaterial.emissiveTexture* is `NULL`, then *Appearance.texture* affects the *emissiveParameter* for the lighting equation.

The *UnlitMaterial.emissiveTextureMapping* value doesn't matter in this case.

4. Otherwise, if the *Appearance.material* is `NULL`, then we behave as if the [UnlitMaterial](#) with all fields at default was used. So the *Appearance.texture* affects the *emissiveParameter* for the lighting equation, and is used with the unlit lighting model.

Note that when the *Appearance.texture* is used to calculate one of the parameters described above, the texture coordinates/transformations are determined following the [MultiTexture](#) specification. This means that [MultiTextureCoordinate](#) and [MultiTextureTransform](#) nodes can be used, with the order corresponding to the order of textures inside *MultiTexture.texture* list. If the *Appearance.texture* is not *MultiTexture* then the first set of texture coordinates/transformations are used. See the [MultiTextureCoordinate](#) and [MultiTextureTransform](#) specification for details.

The [17 Lighting component](#) describes the exact equations to calculate the lighting parameters, consistent with the above description.

## 12.3 Abstract types



### 12.3.1 *X3DAppearanceChildNode*

```
X3DAppearanceChildNode : X3DNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This is the base node type for the child nodes of the *X3DAppearanceNode* type.

### 12.3.2 *X3DAppearanceNode*

```
X3DAppearanceNode : X3DNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This is the base node type for all Appearance nodes.

### 12.3.3 *X3DMaterialNode*

```
X3DMaterialNode : X3DAppearanceChildNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This is the base node type for all material nodes.

There are two direct descendants of this node type:

#### 1. Abstract [X3DOneSidedMaterialNode](#).

In turn, the *X3DOneSidedMaterialNode* is a descendant for all non-abstract and non-deprecated material nodes that you shall use in X3D models:

- [Material](#) (Phong lighting model)
- [PhysicalMaterial](#) (physically-based lighting model)
- [UnlitMaterial](#) (trivial lighting model that ignores light sources, for non-realistic rendering and special effects)

#### 2. [TwoSidedMaterial](#) (deprecated)

### 12.3.4 *X3DOneSidedMaterialNode*

```
X3DOneSidedMaterialNode : X3DMaterialNode {
  SFColor [in,out] emissiveColor 0 0 0 [0, 1]
  SFNode [in,out] emissiveTexture NULL [X3DSingleTextureNode]
  SFString [in,out] emissiveTextureMapping ""
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFNode [in,out] normalTexture NULL [X3DTexture2DNode]
  SFString [in,out] normalTextureMapping ""
  SFFloat [in,out] normalScale 1 [0, ∞]
}
```

Editorial note: We consider merging this abstract node with *X3DMaterialNode*. On one hand, it would simplify the hierarchy. On the other hand, (deprecated) *TwoSidedMaterial* would be left in a weird state, with fields it doesn't use (like *emissiveTexture*, *normalTexture*...). Implementations that still support *TwoSidedMaterial* would need to "invent" a class similar to "*X3DOneSidedMaterialNode*" on their own. See [here for details](#).

This is the base node type for material nodes that describe how the shape looks like from one side.

This node defines common properties for a lighting calculation, but independent of the lighting model (Phong, physically-based, unlit).

This node can be used within *Appearance.material* or *Appearance.backMaterial*.

The *normalTexture* field affects normal vectors information (surface curvature) in the following way:

- Each normal encoded in a texture is a 3D vector (normalized direction).

3D direction of each normal shall be calculated from texture RGB color, using this equation:

$$normal.xyz = normalize((textureSample(normalTexture).rgb * vec3(2,2,2) - vec3(1,1,1)) * vec3(normalScale, normalScale, 1))$$

That is, assuming *normalScale* equal 1 (default), the red color component is linearly mapped from [0..1] to [-1..1] range and represents the X axis of the normal vector. Analogously the green component is mapped to Y, and the blue component is mapped to Z.

- The normals are provided in the *tangent space*.

In tangent space:

- The (0,0,1) vector is pointing perfectly outward from a polygon.

More precisely, the "outward" direction (mapped to (0,0,1) in tangent space) is the direction of the "normal vector" derived from other X3D mechanisms: from the per-vertex or per-face normal vectors (if provided in the *Normal* node), or calculating the normals automatically (e.g. using *IndexedFaceSet.creaseAngle*).

- The vectors (1,0,0) and (0,1,0) in the tangent space indicate the direction where the texture coordinate U and V grows. Naturally, they are adjusted to be always orthogonal to (0,0,1) and each other.

In the future we may add to the X3D standard a way to provide explicit tangent vectors. In X3D 4.0, the implementation should always calculate tangent and bitangent vectors using a standard algorithm, like the *MikkTSpace algorithm*.

- Observe that a correct normalmap texture is typically blueish, since most of the normals on a more-or-less smooth surface revolve around (0,0,1), thus the texture colors revolve around (0.5,0.5,1).
- Alpha channel of the *normalTexture* is ignored by the calculations. X3D browsers *can* use the alpha channel of the *normalTexture* to specify heights (from which the normal vectors have been derived). In the current X3D standard version, these heights are not used for anything, although browsers may already use them for browser-specific rendering effects (for example to perform *parallax bump mapping* or *displacement*, activated by browser-specific extensions).

The *emissiveColor*, together with *emissiveTexture*, allow to model "glowing" objects.

This can be useful for displaying unlit (pre-lit) models (where the light energy of the room is computed explicitly), or for displaying scientific data. To display an "unlit" object (whose visible color should not be modified by any light in the scene), author can use [UnlitMaterial](#) node.

The *emissiveTexture* RGB channel is multiplied with the *emissiveColor* to yield the *emissiveParameter* in the [lighting equations](#).

The meaning of the alpha channel of the *emissiveTexture* depends on the *X3DOneSidedMaterialNode* descendant. It is ignored by [Material](#) and [PhysicalMaterial](#). It is used, as the transparency factor, by the [UnlitMaterial](#). Across the specification, the treatment of *Material.diffuseTexture*, *PhysicalMaterial.baseTexture* and *UnlitMaterial.emissiveTexture* is consistent: these "main" textures provide the transparency information for given material. For details, refer to the documentation of each *X3DOneSidedMaterialNode* descendant.

See the section [12.2.4 Texture mapping specified in material nodes](#) for a description how the texture coordinates and texture coordinate transformations are determined based on the *xxxTextureMapping* fields of this node.

### 12.3.5 X3DShapeNode

```
X3DShapeNode : X3DChildNode, X3DBoundedObject {
  SFNode [in,out] appearance NULL [X3DAppearanceNode]
  SFBool [in,out] bboxDisplay FALSE
  SFNode [in,out] geometry NULL [X3DGeometryNode]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFBool [in,out] visible TRUE
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}
```

This is the base node type for all Shape nodes.

## 12.4 Node reference

### 12.4.1 AcousticProperties

```
AcousticProperties : X3DAppearanceChildNode {
  SFFloat [in,out] absorption 0 [0,1]
  SFFloat [in,out] diffuse 0 [0,1]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFFloat [in,out] refraction 0 [0,1]
  SFFloat [in,out] specular 0 [0,1]
}
```

The *AcousticProperties* node specifies the interaction of sound waves with the characteristics of objects in the scene. Properties influencing sound propagation include surface-related physical phenomena such as the *specular* reflection, *diffuse* reflection, *absorption*, and *refraction* coefficients of materials. These coefficient values are expected to fully account for physical and structural characteristics of the associated geometry such as width, height, thickness, shape, softness and/or hardness, and density variations.

The *absorption* field specifies the sound absorption coefficient of a surface which is the ratio of the sound intensity absorbed or otherwise not reflected by a specific surface that of the initial sound intensity. This characteristic depends on the nature and thickness of the material. Sound energy is partially absorbed when it encounters fibrous

or porous materials, panels that have some flexibility, volumes of air that resonate, and openings in room boundaries (e.g. doorways). Moreover, the absorption of sound by a particular shape depends on the angle of incidence and frequency of the sound wave.

The *diffuse* field describes the diffuse coefficient of sound reflection. This is one of the physical phenomena of sound that occurs when a sound wave strikes a plane surface, and part of the sound energy is reflected back into space in multiple directions.

The *refraction* field describes the sound refraction coefficient of a medium, which determines the change in propagation direction of a sound wave when it obliquely crosses the boundary between two mediums where its speed is different. These relationships are described by Snell's Law.

The *specular* field describes the specular coefficient of sound reflection, which is one of the physical phenomena of sound that occurs when a sound wave strikes a plane surface. Part of the sound energy is directly reflected back into space, where the angle of reflection is equal to the angle of incidence.

## 12.4.2 Appearance

```
Appearance : X3DAppearanceNode {
  SFNode [in,out] acousticProperties NULL [AcousticProperties]
  SFNode [in,out] backMaterial NULL [X3DOneSidedMaterialNode]
  SFNode [in,out] fillProperties NULL [FillProperties]
  SFNode [in,out] lineProperties NULL [LineProperties]
  SFNode [in,out] pointProperties NULL [PointProperties]
  SFNode [in,out] material NULL [X3DMaterialNode]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFNode [in,out] shaders [] [X3DShaderNode]
  SFNode [in,out] texture NULL [X3DTextureNode]
  SFNode [in,out] textureTransform NULL [X3DTextureTransformNode]
}
```

The Appearance node specifies the visual properties of geometry. The value for each of the fields in this node may be `NULL`. However, if the field is non-`NULL`, it shall contain one node of the appropriate type.

The *acousticProperties* field, if specified, shall contain an [AcousticProperties](#) node describing coefficients related to the physical propagation of sound for various materials.

The *material* field, if specified, shall contain a [Material](#), [PhysicalMaterial](#), [TwoSidedMaterial \(deprecated\)](#) or [UnlitMaterial](#) node. If the *material* field is `NULL` or unspecified, lighting is off (all lights are ignored during rendering of the object that references this Appearance) and the unlit object colour is (1, 1, 1). Details of the X3D lighting model are in [17 Lighting component](#).

The *backMaterial* field, if specified, shall contain a [Material](#), [PhysicalMaterial](#) or [UnlitMaterial](#) node. It is only allowed to define a *backMaterial* if the *material* is also defined (not `NULL`). The node type provided to *backMaterial* (if any) must match the node type provided to *material*. This field allows to render back faces with a different material parameters than the front faces. The meaning and all constraints of this field are explained in the section [Two-sided materials](#).

The *texture* field, if specified, shall contain one of the various types of texture nodes (see [18 Texturing component](#)). If the texture node is `NULL` or the *texture* field is unspecified, the object that references this Appearance is not textured.

The *textureTransform* field, if specified, shall contain a [TextureTransform](#) node as defined in [18.4.8 TextureTransform](#). If the *textureTransform* is `NULL` or unspecified, the *textureTransform* field has no effect.

The *fillProperties* field, if specified, shall contain a [FillProperties](#) node. If *fillProperties* is `NULL` or unspecified, the *fillProperties* field has no effect.

The *lineProperties* field, if specified, shall contain a [LineProperties](#) node. If *lineProperties* is `NULL` or unspecified, the *lineProperties* field has no effect.

The *pointProperties* field, if specified, shall contain a [PointProperties](#) node. If *pointProperties* is `NULL` or unspecified, the *pointProperties* field has no effect.

The *shaders* field contains a listing, in order of preference, of nodes that describe programmable shaders that replace the fixed rendering requirements of this part of ISO/IEC 19775 with user-provided functionality. If the field is not empty, one shader node is selected and the fixed rendering requirements defined by this specification are ignored. The field shall contain one of the various types of shader nodes as specified in [31 Programmable shaders component](#).

### 12.4.3 FillProperties

```
FillProperties : X3DAppearanceChildNode {
  SFBool [in,out] filled TRUE
  SFColor [in,out] hatchColor 1 1 1 [0,1]
  SFBool [in,out] hatched TRUE
  SFInt32 [in,out] hatchStyle 1 [0,∞)
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

The *FillProperties* node specifies additional properties to be applied to all polygonal areas on top of whatever appearance is specified by the other fields of the respective [Appearance](#) node. Thus, hatches are applied on top of the already rendered appearance of the node. Thus, if *filled* is `TRUE`, the polygonal area is filled according to the other fields of the *Appearance* node. If *hatched* is `TRUE`, the polygonal area is hatched as specified by the *hatchStyle* field. Hatches shall be applied after fills are applied.

The *hatchStyle* field selects a hatch pattern as defined in the International Register of Graphical Items (see [2.\[REG\]](#)). The hatches are rendered using the colour specified by the *hatchColor* field. Browsers shall support hatchstyles 1-6 with hatchstyle 1 being the default. X3D browsers may support any other of the registered hatchstyles. If a hatchstyle that is not supported is requested, hatchstyle 1 shall be used. [Table 12.2](#) specifies the first nineteen hatch styles as defined in the [Hatchstyle Section of the International Register of Items](#). Examples of each hatch style are available at the International Register of Items.

**Table 12.2 — International register of items hatchstyles**

1	Horizontal equally spaced parallel lines
2	Vertical equally spaced parallel lines
3	Positive slope equally spaced parallel lines
4	Negative slope equally spaced parallel lines

5	Horizontal/vertical crosshatch
6	Positive slope/negative slope crosshatch
7	(cast iron or malleable iron and general use for all materials)
8	(steel)
9	(bronze, brass, copper, and compositions)
10	(white metal, zinc, lead, babbit, and alloys)
11	(magnesium, aluminum, and aluminum alloys)
12	(rubber, plastic, and electrical insulation)
13	(cork, felt, fabric, leather, and fibre)
14	(thermal insulation)
15	(titanium and refractory material)
16	(marble, slate, porcelain, glass, etc.)
17	(earth)
18	(sand)
19	(repeating dot)

The associated geometry shall be filled and/or hatched only when the respective values of the *filled* and/or *hatched* fields have value `TRUE`.

### 12.4.4 LineProperties

```
LineProperties : X3DAppearanceChildNode {
  SFBool [in,out] applied      TRUE
  SFInt32 [in,out] linetype    1 [1,∞)
  SFFloat [in,out] linewidthScaleFactor 0 (-∞,∞)
  SFNode [in,out] metadata     NULL [X3DMetadataObject]
}
```

The `LineProperties` node specifies additional properties to be applied to all line geometry. The *linetype* and *linewidth* *linewidthScaleFactor* fields shall only be applied when the *applied* field has value `TRUE`. When the value of the *applied* field is `FALSE`, a solid line of nominal width shall be produced. The colour of the line is specified by the associated [Material](#) node or [X3DColorNode](#) color values.

The *linetype* field selects a line pattern as defined in the International Register of Graphical Items (see [2.\[REG\]](#)). X3D browsers shall support *linetype* values 1 through 5, with 1 being the default value. X3D browsers may support any other of the registered *linetype* values. If a *linetype* that is not supported is requested, value 1 shall be used. [Table 12.2](#) specifies the first sixteen *linetype* values as defined in the [Linetype Section of the International Register of Items](#).



**Table 12.3 — International register of items linetypes**

1	Solid
2	Dashed
3	Dotted
4	Dashed-dotted
5	Dash-dot-dot
6	(single arrow)
7	(single dot)
8	(double arrow)
10	(chain line)
11	(center line)
12	(hidden line)
13	(phantom line)
14	(break line 1)
15	(break line 2)
16	User-specified dash pattern

The arrowhead is drawn as short lines forming barbs at any convenient angle between 15 and 90 degrees. The arrowhead is closed and filled in. For linetype "single arrow", the arrowhead is rendered so that the arrow tip occurs at the last point of the each individual list of points passed to a polyline and is in the direction of the last vector. For linetype "double arrow", the first arrowhead is rendered so that the arrow tip occurs at the first point of the list of points passed to a polyline and is in the reverse direction of the first vector. The second arrowhead is rendered as for "single arrow" at the opposite end of the polyline.

The *linewidthScaleFactor* field is a multiplicative value that scales a browser-dependent nominal line width by the given value. This resulting value shall then be mapped to the nearest available line width. A value less than or equal to zero refers to the minimum available line width.

### 12.4.5 Material

```
Material : X3DOneSidedMaterialNode {
  SFFloat [in,out] ambientIntensity 0.2 0.11
  SFNode [in,out] ambientTexture NULL [X3DSingleTextureNode]
  SFString [in,out] ambientTextureMapping ""
  SFCOLOR [in,out] diffuseColor 0.8 0.8 0.8 0.11
```



```

SFNode [in,out] diffuseTexture NULL [X3DSingleTextureNode]
SFString [in,out] diffuseTextureMapping ""

SFColor [in,out] emissiveColor 0 0 0 [0,1]
SFNode [in,out] emissiveTexture NULL [X3DSingleTextureNode]
SFString [in,out] emissiveTextureMapping ""

SFNode [in,out] metadata NULL [X3DMetadataObject]

SFNode [in,out] normalTexture NULL [X3DTexture2DNode]
SFString [in,out] normalTextureMapping ""
SFFloat [in,out] normalScale 1 [0, ∞]

SFFloat [in,out] occlusionStrength 1 [0,1]
SFNode [in,out] occlusionTexture NULL [X3DTexture2DNode]
SFString [in,out] occlusionTextureMapping ""

SFFloat [in,out] shininess 0.2 [0,1]
SFNode [in,out] shininessTexture NULL [X3DSingleTextureNode]
SFString [in,out] shininessTextureMapping ""

SFColor [in,out] specularColor 0 0 0 [0,1]
SFNode [in,out] specularTexture NULL [X3DSingleTextureNode]
SFString [in,out] specularTextureMapping ""

SFFloat [in,out] transparency 0 [0,1]
}

```

The `Material` node specifies surface material properties for associated geometry nodes and is used by the X3D lighting equations during rendering. [17 Lighting component](#) contains a detailed description of the X3D lighting model equations.

All of the fields in the `Material` node range from 0.0 to 1.0.

The fields in the `Material` node determine how light reflects off an object to create colour:

- The *ambientIntensity* field specifies how much ambient light from light sources this surface shall reflect. Ambient light is omnidirectional and depends only on the number of light sources, not their positions with respect to the surface. Ambient colour is calculated as  $ambientIntensity \times diffuseColor$ .
- The *diffuseColor* field reflects all X3D light sources depending on the angle of the surface with respect to the light source. The more directly the surface faces the light, the more diffuse light reflects.
- The *emissiveColor* field models "glowing" objects. This can be useful for displaying pre-lit models (where the light energy of the room is computed explicitly), or for displaying scientific data.
- The *specularColor* and *shininess* fields determine the specular highlights (e.g., the shiny spots on an apple). When the angle from the light to the surface is close to the angle from the surface to the viewer, the *specularColor* is added to the diffuse and ambient colour calculations. Lower shininess values produce soft glows, while higher values result in sharper, smaller highlights.
- The *transparency* field specifies how "clear" an object is, with 1.0 being completely transparent, and 0.0 completely opaque.

The [Material](#) node specifies surface material properties for associated geometry nodes. It indicates that a surface is using *Phong lighting model*. [17 Lighting component](#) contains a detailed description of the X3D lighting model equations.

The material parameters are specified as scalars or RGB colors in the X3D file. All of the `SFFloat` and `SFColor` fields in the `Material` node range from 0.0 to 1.0.

Moreover every material parameter can be adjusted using a texture. This allows to vary this parameter across the surface. The information sampled from the texture is always

multiplied by the simple scalar/color fields.

Examples of texture usage:

- Texture assigned to the *diffuseTexture* controls the most intuitive "visible color of the object". This is the most often used texture.
- Texture assigned to the *specularTexture* allows the surface to be partially shiny (white values in the texture) and partially matte (black values in the texture).

The fields in the Material node determine how light reflects off an object to create color:

- a. The *ambientIntensity* and *ambientTexture* fields specify how much ambient light from light sources this surface shall reflect. Ambient light is omnidirectional and depends only on the number of light sources, not their positions with respect to the surface.

Ambient parameter is calculated as

$ambientIntensity \times diffuseColor \times textureSample(ambientTexture).rgb$ .

- b. The *diffuseColor* and *diffuseTexture* fields reflect all X3D light sources depending on the angle of the surface with respect to the light source. The more directly the surface faces the light, the more diffuse light reflects.
- c. The *emissiveColor* and *emissiveTexture* fields model "glowing" or "unlit" objects. See [X3DOneSidedMaterialNode](#) for the description of these fields.
- d. The *specularColor*, *specularTexture*, *shininess* and *shininessTexture* fields determine the specular highlights (*e.g.*, the shiny spots on an apple).

When the angle from the light to the surface is close to the angle from the surface to the viewer, the  $specularColor \times textureSample(specularTexture).rgb$  is added to the diffuse and ambient color calculations.

Lower shininess values produce soft glows, while higher values result in sharper, smaller highlights. Shininess is calculated as  $shininess \times textureSample(shininessTexture).a$ .

- e. The *transparency* field (together with alpha channel of the *diffuseTexture*) specifies how "clear" an object is, with 1.0 being completely transparent, and 0.0 completely opaque.

The transparency determines the *opacity* as  $opacity = 1.0 - transparency$ . This is then multiplied by the alpha channel of *diffuseTexture* to determine the final alpha of the rendered pixel.

The RGB channels of *diffuseTexture*, *specularTexture* and *emissiveTexture* are multiplied by the corresponding *diffuseColor*, *specularColor*, *emissiveColor* before being used in the current lighting calculation. The alpha channel of a *diffuseTexture* is multiplied by the material *opacity* (which equals just  $1.0 - transparency$ ). The alpha channels contents of *specularTexture* and *emissiveTexture* are ignored.

The *shininessTexture* alpha channel contains values multiplied with the *shininess* factor of the [Material](#) node. The RGB channels contents of the *shininessTexture* are ignored.

It is expected, and advised, that authors reuse the same texture node for *specularTexture* and *shininessTexture*. The specular data is deliberately contained in different channels (RGB) than the shininess data (Alpha).

The optional *occlusionTexture* can be used to indicate areas of indirect lighting, typically called *ambient occlusion*. Only the *Red* channel of the texture is used for the computation, the other channels are ignored. Higher values indicate areas that should receive full indirect lighting and lower values indicate no indirect lighting. The *occlusionStrength* determines how much does the occlusion texture affect the final result.

See the section [12.2.4 Texture mapping specified in material nodes](#) for a description how the texture coordinates and texture coordinate transformations are determined based on the *xxxTextureMapping* fields of this node.

## 12.4.6 PhysicalMaterial

```
PhysicalMaterial : X3DOneSidedMaterialNode {
  SFColor [in,out] baseColor          1 1 1 [0,1]
  SFNode [in,out] baseTexture         NULL [X3DSingleTextureNode]
  SFString [in,out] baseTextureMapping ""

  SFFloat [in,out] metallic           1 [0,1]
  SFNode [in,out] metallicRoughnessTexture NULL [X3DSingleTextureNode]
  SFString [in,out] metallicRoughnessTextureMapping ""

  SFColor [in,out] emissiveColor      0 0 0 [0,1]
  SFNode [in,out] emissiveTexture     NULL [X3DSingleTextureNode]
  SFString [in,out] emissiveTextureMapping ""

  SFNode [in,out] metadata            NULL [X3DMetadataObject]

  SFNode [in,out] normalTexture        NULL [X3DTexture2DNode]
  SFString [in,out] normalTextureMapping ""
  SFFloat [in,out] normalScale         1 [0, ∞]

  SFFloat [in,out] occlusionStrength   1 [0,1]
  SFNode [in,out] occlusionTexture     NULL [X3DSingleTextureNode]
  SFString [in,out] occlusionTextureMapping ""

  SFFloat [in,out] roughness           1 [0,1]

  SFFloat [in,out] transparency        0 [0,1]
}
```

The [PhysicalMaterial](#) node specifies surface material properties for associated geometry nodes. It indicates that a physical lighting model should be used for the computation. [17 Lighting component](#) contains a detailed description of the X3D lighting model equations.

The physical lighting equation, as an input, relies on the following parameters:

- *baseParameter* (RGB color) is, in simple cases, a multiplication of *baseTexture* RGB channel (if such texture was specified) with the *baseColor*.

In other words, it is calculated at every pixel as  
 $baseColor \times textureSample(baseTexture).rgb$ .

Note: This interpretation is true in most cases, but in general it is a simplification of what actually happens. The texture may also come from *Appearance.texture*, and it can even be *MultiTexture* in which case it is not necessarily multiplied. See the [17.2.2.6 Physical lighting model](#) for the exact specification how the *baseParameter* is calculated in every possible case.

- *metallicParameter* is a multiplication of *metallic* with the *Blue* texture channel of

*metallicRoughnessTexture* (if such texture was specified).

In other words, it is calculated at every pixel as  
 $metallic \times textureSample(metallicRoughnessTexture).b$ .

- *roughnessParameter* is a multiplication of *roughness* with the *Green* texture channel of *metallicRoughnessTexture* (if such texture was specified).

In other words, it is calculated at every pixel as  
 $roughness \times textureSample(metallicRoughnessTexture).g$ .

When calculating *metallicParameter* and *roughnessParameter* terms, the *Red* and *Alpha* channels of the *metallicRoughnessTexture* are ignored. It is possible to use the same texture for *metallicRoughnessTexture* and *occlusionTexture*, as they deliberately look at different channels, so all the information can be contained in one RGB texture.

The final alpha, used for blending or alpha-testing, is calculated as *baseTexture* alpha channel multiplied with the opacity ( $1.0 - transparency$ ). This is consistent with the behavior of *diffuseColor*, *diffuseTexture* and *transparency* on the Phong [Material](#). If the *baseTexture* was not specified, it is also possible to use the *Application.texture*. See the [17.2.2.6 Physical lighting model](#) for the exact specification, and the [12.2.5 Coexistence of textures specified in material nodes with the "Appearance.texture" field](#) for a description how *Appearance.texture* is used.

Moreover the [PhysicalMaterial](#) defines the *emissiveColor* and optional *emissiveTexture*. The resulting *emissiveParameter* term is simply added to the pixel color, this behavior is consistent for all X3D materials.

The optional *occlusionTexture* can be used to indicate areas of indirect lighting, typically called *ambient occlusion*. Only the *Red* channel of the texture is used for the computation, the other channels are ignored. Higher values indicate areas that should receive full indirect lighting and lower values indicate no indirect lighting. The *occlusionStrength* determines how much does the occlusion texture affect the final result.

### Physical interpretation of the material parameters:

*Note: The physical material properties of X3D are deliberately consistent with the glTF 2.0 material definition. Effectively, converting between (in both directions) between X3D PhysicalMaterial and glTF 2.0 material definitions is trivial.*

*The description of the parameter meaning below follows very closely the glTF specification.*

The *baseParameter* color has two different interpretations depending on the value of *metallicParameter*. When the material is a metal, the *baseParameter* color is the specific measured reflectance value at normal incidence ( $F_0$ ). For a non-metal the *baseParameter* color represents the reflected diffuse color of the material. In this model it is not possible to specify a  $F_0$  value for non-metals, and a linear value of 4% (0.04) is used.

The following equations show how to calculate bidirectional reflectance distribution function (BRDF) inputs ( $c_{diff}$ ,  $F_0$ ,  $\alpha$ ) from the metallic-roughness material properties.

```
const dielectricSpecular = rgb(0.04, 0.04, 0.04)
const black = rgb(0, 0, 0)
cdiff = lerp(baseParameter * (1 - dielectricSpecular.r), black, metallicParameter)
F0 = lerp(dielectricSpecular, baseParameter, metallicParameter)
α = roughnessParameter ^ 2
```

See the section [12.2.4 Texture mapping specified in material nodes](#) for a description how the texture coordinates and texture coordinate transformations are determined based on the `xxxTextureMapping` fields of this node.

## 12.4.7 PointProperties

```
PointProperties : X3DAppearanceChildNode {
  SFFloat [in,out] pointSizeScaleFactor 1 [1,∞)
  SFFloat [in,out] pointSizeMinValue 1 [0,∞)
  SFFloat [in,out] pointSizeMaxValue 1 [0,∞)
  SFVec3f [in,out] attenuation 1 0 0 [0,∞)
  SFString [in,out] colorMode "TEXTURE_AND_POINT_COLOR" ["POINT_COLOR" | "TEXTURE_COLOR" |
"TEXTURE_AND_POINT_COLOR"]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

The `PointProperties` node specifies additional properties to be applied to all point geometry. The colour of the line is specified by the associated [Material](#) node or [X3DColorNode](#) color values.

*pointSizeScaleFactor* is a value determining the nominal point size before modification by the sizing modifications, as determined by the *pointSizeMinValue*, *pointSizeMaxValue*, and *attenuation* values discussed below. The nominal rendered point size is a browser-dependent minimum renderable point size.

*pointSizeMinValue* is minimum allowed scaling factor on nominal browser point scaling. *pointSizeMaxValue* is maximum allowed scaling factor on nominal browser point scaling. The provided value for *pointSizeMinValue* must be less than or equal to value for *pointSizeMaxValue*.

The *attenuation* field defines a depth perception effect in a point cloud rendering by making points close to the viewer appear larger. The modification of point size depending on distance from the view occurs in two steps, starting with the nominal point size as determined by the *pointSizeScaleFactor* field. The *attenuation* field defines three parameters *a*, *b*, and *c* from the components of a single `SFVec3f` value:

```
a = attenuation[0]
b = attenuation[1]
c = attenuation[2]
```

Together these parameters define an attenuation factor  $1/(a + b \times r + c \times r^2)$  where *r* is the distance from the observer position (current viewpoint) to each point. The nominal point size is multiplied by the attenuation factor and then clipped to a minimum value of *pointSizeMinValue* × the minimum renderable point size, then clipped to a maximum size of *pointSizeMaxValue* × minimum renderable point size.

When a `X3DTextureNode` is defined in the same `Appearance` instance as `PointProperties` node, the points of a `PointSet` shall be displayed as point sprites using the given texture(s). The *colorMode* field has a blending effect on the rendering of point sprites. A

value of:

- POINT\_COLOR shall display the RGB channels of the color instance defined in X3DMaterialNode or X3DColorNode, and the A channel of the texture if any. If no color is associated to the point, the default RGB color (0, 0, 0) shall be used.
- TEXTURE\_COLOR shall display the original texture with its RGBA channels and regardless to the X3DMaterialNode or X3DColorNode which might be associated to the point set.
- TEXTURE\_AND\_POINT\_COLOR shall display the RGBA channels of a texture added to the RGB channels of the color defined in X3DMaterialNode or X3DColorNode node, and the A channel of the texture if any. If no color is associated to the point, the result shall be exactly the same as TEXTURE\_COLOR.

If no X3DTextureNode is defined in the same Appearance instance as PointProperties, points of the PointSet shall be displayed anti-aliased and using their associated colors.

TODO reference/bibliography International register of items for markertype.

## 12.4.8 Shape

```
Shape : X3DShapeNode {
  SFNode [in,out] appearance NULL [X3DAppearanceNode]
  SFBool [in,out] bboxDisplay FALSE
  SFNode [in,out] geometry NULL [X3DGeometryNode]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFBool [in,out] visible TRUE
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}
```

The Shape node has two fields, *appearance* and *geometry*, that are used to create rendered objects in the world. The *appearance* field contains an Appearance node that specifies the visual attributes (e.g., material and texture) to be applied to the geometry. The *geometry* field contains a geometry node. The specified geometry node is rendered with the specified appearance nodes applied. See [12.2 Concepts](#) for more information.

[17 Lighting component](#) contains details of the X3D lighting model and the interaction between Appearance nodes and geometry nodes.

If the *geometry* field is `NULL`, the object is not drawn.

The *bboxCenter* and *bboxSize* fields specify a bounding box that encloses the Shape node's geometry. This is a hint that may be used for optimization purposes. The results are undefined if the specified bounding box is smaller than the actual bounding box of the geometry at any time. A default *bboxSize* value, (-1 -1 -1), implies that the bounding box is not specified and, if needed, is calculated by the browser. A description of the *bboxCenter* and *bboxSize* fields is contained in [10.2.2 Bounding boxes](#).

## 12.4.9 TwoSidedMaterial (deprecated)

```
TwoSidedMaterial : X3DMaterialNode {
  SFFloat [in,out] ambientIntensity 0.2 [0,1]
  SFFloat [in,out] backAmbientIntensity 0.2 [0,1]
  SFColor [in,out] backDiffuseColor 0.8 0.8 0.8 [0,1]
  SFColor [in,out] backEmissiveColor 0 0 0 [0,1]
  SFFloat [in,out] backShininess 0.2 [0,1]
  SFColor [in,out] backSpecularColor 0 0 0 [0,1]
  SFFloat [in,out] backTransparency 0 [0,1]
  SFColor [in,out] diffuseColor 0.8 0.8 0.8 [0,1]
```



```

SFColor [in,out] emissiveColor    0 0 0    [0,1]
SFNode [in,out] metadata         NULL    [X3DMetadataObject]
SFFloat [in,out] shininess       0.2      [0,1]
SFBool [in,out] separateBackColor FALSE
SFColor [in,out] specularColor   0 0 0    [0,1]
SFFloat [in,out] transparency    0      [0,1]
}

```

**This node is deprecated since X3D version 4.0. Future versions of the standard may remove this node.** The upgrade path is as follows:

- If you used *TwoSidedMaterial* with *separateBackColor* equal `FALSE` (default), then simply use the simpler [Material](#) node instead. In case of *separateBackColor* equal `FALSE`, the *TwoSidedMaterial* was actually useless. Only the *solid* field, described in the [11.2.3 Common geometry fields](#), controls whether the geometry is visible from the back side (this was true in X3D 3.x and remains true in X3D 4.x).
- If you used *TwoSidedMaterial* with *separateBackColor* equal `TRUE`, then instead use *Appearance.backMaterial* field to specify different rendering parameters for the back faces. See [Two-sided materials](#).

This node defines material properties that can effect both the front and back side of a polygon individually. These materials are used for both the front and back side of the geometry whenever the X3D lighting model is active.

If the *separateBackColor* field is set to `TRUE`, the rendering shall render the front and back faces of the geometry with different values. If the value is `FALSE`, the front colours are used for both the front and back side of the polygon, as per the existing X3D lighting rules.

When calculating the terms in the lighting equations, the front geometry shall use the fields *ambientIntensity*, *diffuseColor*, *emissiveColor*, *shininess*, *specularColor*, and *transparency*. The faces that are determined to be the back side are rendered using *backAmbientIntensity*, *backDiffuseColor*, *backEmissiveColor*, *backShininess*, and *backTransparency* as the appropriate components in the lighting equations.

## 12.4.10 UnlitMaterial

```

UnlitMaterial : X3DOneSidedMaterialNode {
  SFColor [in,out] emissiveColor    1 1 1 [0,1]
  SFNode [in,out] emissiveTexture   NULL [X3DSingleTextureNode]
  SFString [in,out] emissiveTextureMapping ""
  SFNode [in,out] metadata         NULL [X3DMetadataObject]
  SFNode [in,out] normalTexture     NULL [X3DTexture2DNode]
  SFString [in,out] normalTextureMapping ""
  SFFloat [in,out] normalScale      1 [0, ∞]
  SFFloat [in,out] transparency    0 [0,1]
}

```

Material that is unaffected by light sources. Suitable to create various non-realistic effects, when the colors are defined explicitly and are not affected by the placement of the shape relative to the lights or camera.

The output color and opacity, called *emissiveParameter* by the [lighting equations](#), are determined like this:

1. Use the *emissiveColor* field value as the *emissiveParameter.rgb*. Use the `1.0 - transparency` as the *emissiveParameter.a*.
2. If shape is using [Color](#) node then the information from [Color](#) node overrides the *emissiveParameter.rgb*. If shape is using [ColorRGBA](#) node then the information



from [ColorRGBA](#) overrides both the *emissiveParameter.rgb* and the *emissiveParameter.a*.

*Note: This is consistent with how [Color](#) or [ColorRGBA](#) override *diffuseColor* and *transparency* in case of [Material](#).*

- If the *emissiveTexture* is not NULL, then it multiplies (component-wise) the *emissiveParameter.rgb* (multiplied by the texture RGB channels) and *emissiveParameter.a* (multiplied by the texture alpha channel).

If the *emissiveTexture* is NULL, but *Appearance.texture* field is not NULL, then the same logic is applied to the *Appearance.texture* texture: it multiplies *emissiveParameter.rgb* and *emissiveParameter.a*. See [12.2.5 Coexistence of textures specified in material nodes with the "Appearance.texture" field](#) for a description how is the *Appearance.texture* field used.

*Note about default values:* This node inherits the *emissiveColor* field from the [X3DOneSidedMaterialNode](#) ancestor, but the default value of this field changes: only for [UnlitMaterial](#), the default *emissiveColor* is 1 1 1 (white), instead of 0 0 0 (black, default of *X3DOneSidedMaterialNode.emissiveColor*).

*Implementation hint:* Normal vectors information is not useful for the calculation of unlit material. Implementations can ignore the normal vectors provided in the geometry node (per-face or per-vertex) and in the *normalTexture* field. Implementations are encouraged to optimize this case, and not send unneeded normals data to GPU, and not calculate implicit normal vectors (normally derived from *creaseAngle* and *ccw* fields). *However, there is an exception to this optimization:* if the shape is using [TextureCoordinateGenerator](#) with some modes (*CAMERASPACE**NORMAL*, *CAMERASPACE**REFLECTIONVECTOR*) then the shader code may need access to normals anyway.

See the section [12.2.4 Texture mapping specified in material nodes](#) for a description how the texture coordinates and texture coordinate transformations are determined based on the *xxxTextureMapping* fields of this node.

## 12.5 Support levels

The Shape component provides three levels of support as specified in [Table 12.4](#).

**Table 12.4 — Shape component support levels**

Level	Prerequisites	Nodes/Features	Support
1	Core 1 Rendering 1 Texturing 1		
		<i>X3DAppearanceChildNode</i> (abstract)	n/a
		<i>X3DAppearanceNode</i> (abstract)	n/a
		<i>X3DMaterialNode</i> (abstract)	n/a

		<i>X3DOneSidedMaterialNode</i> (abstract)	n/a
		<i>X3DShapeNode</i> (abstract)	n/a
		Appearance	Optional support for <i>textureTransform</i> , <i>lineProperties</i> , <i>fillProperties</i> , <i>shaders</i> , <i>backMaterial</i> .
		Material	Optional support for <i>ambientIntensity</i> , <i>shininess</i> , <i>specularColor</i> and all <i>xxxTexture...</i> fields except <i>diffuseTexture</i> (that is, optional support for: <i>ambientTexture</i> , <i>emissiveTexture</i> , <i>normalTexture</i> , <i>occlusionTexture</i> , <i>shininessTexture</i> , <i>specularTexture</i> ).
		UnlitMaterial	All fields fully supported.
		Shape	All fields fully supported.
2	Core 1 Rendering 1 Texturing 1		
		All Level 1 Appearance nodes except Appearance	All fields fully supported.
		Appearance	Optional support for the same properties as on level 1.
		LineProperties	All fields fully supported.
		PhysicalMaterial	All fields fully supported.
	Core 1		

<b>3</b>	Rendering 1 Texturing 1		
		All Level 2 Appearance nodes except Appearance	All fields fully supported.
		Appearance	Optional support for <i>backMaterial</i> .
		FillProperties	All fields fully supported.
<b>4</b>	Core 1 Grouping 1 Rendering 1		
		All Level 3 Appearance nodes	All fields fully supported.
		AcousticProperties	All fields fully supported.
		PointProperties	All fields fully supported.

Note: Support for *TwoSidedMaterial* is not required at *any* level, as it is a deprecated node. As such, X3D browser qualifies for *Full* conformance even without implementing this node. Instead a new field *backMaterial* is required to be supported at level 5 of this component.





## Extensible 3D (X3D) Part 1: Architecture and base components

### 33 Texturing3D Component

#### 33.1 Introduction

##### 33.1.1 Name

The name of this component is "Texturing3D". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 33.1.2 Overview

This clause describes the Texturing3D component of this part of ISO/IEC 19775. [Table 33.1](#) provides links to the major topics in this clause.

**Table 33.1 — Topics**

- [33.1 Introduction](#)
  - [33.1.1 Name](#)
  - [33.1.2 Overview](#)
- [33.2 Concepts](#)
  - [33.2.1 Overview](#)
  - [33.2.2 3D texturing concepts](#)
  - [33.2.3 Texture coordinates](#)
  - [33.2.4 Texture coordinate generation for primitive objects](#)
  - [33.2.5 Texture map image formats](#)
- [33.3 Abstract types](#)
  - [33.3.1 X3DTexture3DNode](#)
- [33.4 Node reference](#)
  - [33.4.1 ComposedTexture3D](#)
  - [33.4.2 ImageTexture3D](#)
  - [33.4.3 PixelTexture3D](#)
  - [33.4.4 TextureCoordinate3D](#)
  - [33.4.5 TextureCoordinate4D](#)
  - [33.4.6 TextureTransformMatrix3D](#)

[33.4.7 TextureTransform3D](#)

- [33.5 Support levels](#)
- [Figure 33.1 — Illustration of how two 2D images can form a 3D volume of texture](#)
- [Table 33.1 — Topics](#)
- [Table 33.2 — Texturing component support levels](#)

## 33.2 Concepts

### 33.2.1 Overview

This component provides additional texturing extensions to the basic capabilities defined in X3D. Many applications need to describe surface properties as data points in a volume of space, rather than a flat surface. These textures operate with three dimensions. A texture of this type is termed a *volumetric texture*.

Volumetric textures are essential for advanced rendering effects related to fog and lighting, as well as industry-specific needs such as medical and CAD visualization.

### 33.2.2 3D texturing concepts

3D texturing specifies texel colours based on a volume of space. An object that is being rendered on that 3D texture effectively cuts a volume out of the texels provided by the texture.

This part of ISO/IEC 19775 assumes standard commodity hardware that presents 3D textures as a series of 2D slices of the volume that can then be interpolated and composited together to form a 3D volume of space. There is no assumption about the existence of true voxel rendering hardware capability.

A 3D volume of texture is specified as a number of 2D planes (images) of data that are ordered in a depth-wise manner. [Figure 33.1](#) shows two base images that can be layered together resulting in the volume of a 3D texture. In this example, the texture would have a dimension of  $n \times m \times 2$ .

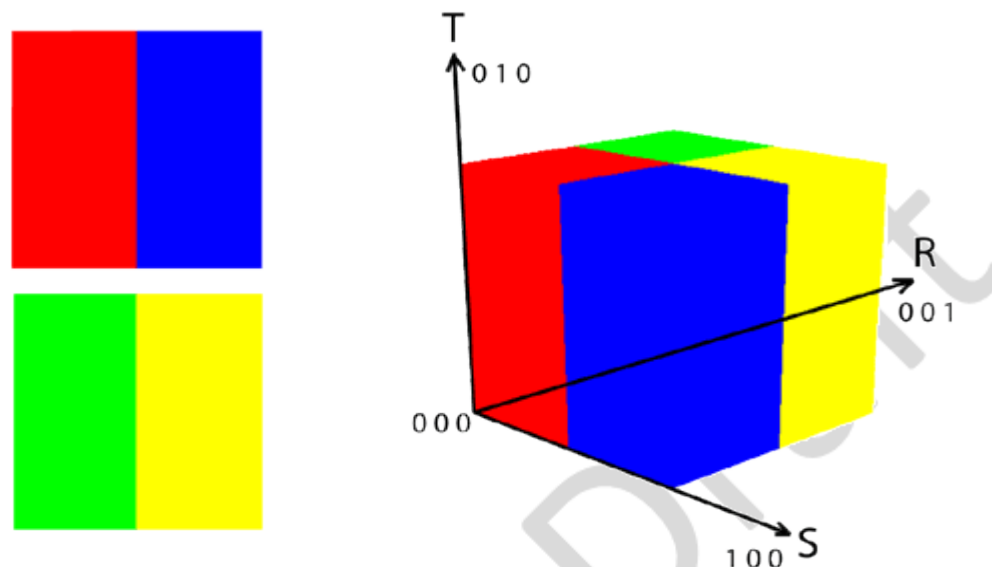


Figure 33.1 — Formation of 3D texture from two 2D textures

### 33.2.3 Texture coordinates

The coordinate system of the texture is a right-handed coordinate system as defined in [Figure 33.1](#). The coordinate components are defined to be (s,t,r) as values along the S, T, and R axes defined by [Figure 33.1](#).

### 33.2.4 Texture coordinate generation for primitive objects

Some geometry nodes are not capable of having 3D texture coordinates set by the user (e.g., Box and Cone). For these cases, 3D textures coordinates are automatically generated based on the following rules:

- All coordinates are generated in the range [0, 1] for the given axis. 0 is for the minimum value of the coordinate vertex on that axis, and 1 is assigned to the maximum value of the coordinate vertex on that axis.
- Orientation is oriented along the z-axis, looking in the -Z direction with a zero angle aligned with the axis.
- S coordinate is generated from left to right based on the maximum extents of the X-axis vertex values.
- T coordinate is generated from top to bottom based on the maximum extents of the Y-axis vertex values.
- R coordinate is generated from front (+Z) to back (-Z) based on the maximum extents of the Z-axis vertex values.

The *default 3D texture coordinate generation* described above is performed only when the *Appearance.texture* contains a node derived from *X3DTexture3DNode*, or when it contains *MultiTexture* and the first child of it is *X3DTexture3DNode*. In particular, it means that using 3D texture has no effect on the default texture coordinate generation algorithm when this 3D texture is used:

- as a non-first child of *MultiTexture*, which is then placed in *Appearance.texture*

- as a texture referenced by a material field, like *Material.diffuseTexture*

In the above two cases, the *default coordinate generation* still follows the standard algorithm (described at each particular geometry node, best suited for 2D textures). The reason for this is that a node may use multiple textures, both 3D and 2D, and in the above cases it's impossible for the browser to know which texture generation scheme (best suited for 3D or 2D texture) is a better default.

## 33.2.5 Texture map image formats

Node types specifying 3D texture maps may supply data with a number of color components between one and four. The valid types and interpretations of 3D textures are identical to that for 2D textures. The definition of texture formats is defined in [18.2.1 Texture map formats](#).

## 33.3 Abstract types

### 33.3.1 X3DTexture3DNode

```
X3DTexture3DNode : X3DSingleTextureNode {
X3DTexture3DNode : X3DTextureNode {
  SFNode [in,out] metadata      NULL [X3DMetadataObject]
  SFBool []    repeatS          FALSE
  SFBool []    repeatT          FALSE
  SFBool []    repeatR          FALSE
  SFNode []    textureProperties NULL [TextureProperties]
}
```

This abstract node type is the base type for all node types that specify 3D sources for texture images.

NOTE The base node type diverges from the standard X3D textures by making the default repeat modes `FALSE`, rather than `TRUE`. This is because 3D textures are almost never used in a repeated rendering mode, and because repeat mode `TRUE` for 3D textures can produce odd rendering artifacts.

## 33.4 Node reference

### 33.4.1 ComposedTexture3D

```
ComposedTexture3D : X3DTexture3DNode {
  SFNode [in,out] metadata      NULL [X3DMetadataObject]
  MFNode [in,out] texture       [] [X3DTexture2DNode]
  SFNode []    textureProperties NULL [TextureProperties]
  SFBool []    repeatS          FALSE
  SFBool []    repeatR          FALSE
  SFBool []    repeatT          FALSE
}
```

The `ComposedTexture3D` node defines a 3D image-based texture map as a collection of 2D texture sources at various depths and parameters controlling tiling repetition of the texture onto geometry.

The texture values are interpreted with the first image being at depth 0 and each following image representing an increasing depth value in the R direction. A user shall provide  $2^n$  source textures in this array. The individual source textures will ignore their *repeat* field values.

See [33.2 Concepts](#), for a general description of texture maps.



See [18 Texturing component](#) for a general description of the [X3DTexture2DNode](#) abstract type and interpretation of rendering for 2D images. When used as a source for cubic environment maps, the fields *repeatS* and *repeatT* fields shall be ignored.

### 33.4.2 ImageTexture3D

```
ImageTexture3D : X3DTexture3DNode, X3DUrlObject {
  SFString [in,out] description ""
  SFBool [in,out] load TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTime [in,out] refresh 0.0 [0,∞)
  MFString [in,out] url [] [URI]
  SFBool [] repeatS FALSE
  SFBool [] repeatT FALSE
  SFBool [] repeatR FALSE
  SFNode [] textureProperties NULL [TextureProperties]
}
```

The ImageTexture3D node defines a texture map by specifying a single image file that contains complete 3D data and general parameters for mapping texels to geometry.

The texture is read from the URL specified by the *url* field. When the *url* field contains no values ([ ]), texturing is disabled. The *url* field is defined in [9.2.1 URLs](#). While there are no required file formats, it is recommended that one of the following formats be supported:

- DDS (see [\[DDS\]](#)),h
- DICOM (see [2.\[DICOM\]](#)),
- NRRD (see [\[NRRD\]](#)), and/or
- .vol (see [\[VOL\]](#)).

See [33.2 Concepts](#) for a general description of texture maps.

### 33.4.3 PixelTexture3D

```
PixelTexture3D : X3DTexture3DNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFInt32 [in,out] image [0 0 0 0]
  SFBool [] repeatS FALSE
  SFBool [] repeatR FALSE
  SFBool [] repeatT FALSE
  SFNode [] textureProperties NULL [TextureProperties]
}
```

The PixelTexture3D node defines a 3D image-based texture map as an explicit array of pixel values (image field) and parameters controlling tiling repetition of the texture onto geometry.

The *image* field describes the raw data to be used for this 3D texture. The first value of the array is the number of components to the image and shall be a value between 0 and 4. The following three numbers are the size of the texture: width, height and depth, respectively. The remaining values of the array are treated as the pixels for the image. There shall be at least width × height × depth number of pixel values provided. Each of the width, height and depth values is required to be a power of two.

See [33.2 Concepts](#) for a general description of 3D texture maps.

See [17 Lighting component](#) for a description of how the texture values interact with the appearance of the geometry. [5.7 SFImage and MFImage](#) describes the specification of

an image.

### 33.4.4 TextureCoordinate3D

```
TextureCoordinate3D : X3DSingleTextureCoordinateNode {
TextureCoordinate3D : X3DTextureCoordinateNode {
  SFString [in,out] mapping ""
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFVec3f [in,out] point [] (-∞,∞)
}
```

The TextureCoordinate3D node is a geometry property node that specifies a set of 3D texture coordinates used by vertex-based geometry nodes (e.g., [IndexedFaceSet](#) and [ElevationGrid](#)) to map 3D textures to vertices.

Providing 3D texture coordinates to objects that only have 2D textures defined shall result in implementation dependent rendering.

### 33.4.5 TextureCoordinate4D

```
TextureCoordinate4D : X3DSingleTextureCoordinateNode {
TextureCoordinate4D : X3DTextureCoordinateNode {
  SFString [in,out] mapping ""
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFVec4f [in,out] point [] (-∞,∞)
}
```

The TextureCoordinate4D node is a geometry property node that specifies a set of 4D (homogeneous 3D) texture coordinates used by vertex-based geometry nodes (e.g., [IndexedFaceSet](#) and [ElevationGrid](#)) to map 3D textures to vertices.

Providing 4D texture coordinates to objects that only have 2D textures defined shall result in implementation dependent rendering.

### 33.4.6 TextureTransformMatrix3D

```
TextureTransformMatrix3D : X3DSingleTextureTransformNode {
TextureTransformMatrix3D : X3DTextureTransformNode {
  SFString [in,out] mapping ""
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFMatrix4f [in,out] matrix 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 (-∞,∞)
}
```

The *matrix* field specifies a generalized, unfiltered 4×4 transformation matrix that can be used to modify the texture. Any set of values is permitted.

### 33.4.7 TextureTransform3D

```
TextureTransform3D : X3DSingleTextureTransformNode {
TextureTransform3D : X3DTextureTransformNode {
  SFString [in,out] mapping ""
  SFVec3f [in,out] center 0 0 0 (-∞,∞)
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFRotation [in,out] rotation 0 0 1 0 (-∞,∞)
  SFVec3f [in,out] scale 1 1 1 (-∞,∞)
  SFVec3f [in,out] translation 0 0 0 (-∞,∞)
}
```

The TextureTransform3D node specifies a 3D transformation that is applied to texture coordinates (see [33.4.4 TextureCoordinate3D](#)). This node affects the way texture coordinates are applied to the geometric surface. The transformation consists of (in order):

- a. a translation;

- b. a rotation about the centre point; and
- c. a non-uniform scale about the centre point.

These parameters support changes to the size, orientation, and position of textures on shapes. These operations appear reversed when viewed on the surface of geometry.

**EXAMPLE** A scale value of (1 2 2) will scale the texture coordinates and have the net effect of shrinking the texture size by a factor of 2 (texture coordinates are twice as large and thus cause the texture to repeat) in the T and R dimensions and leave the S dimension unscaled. A translation of (0.5 0.0 0.0) translates the texture coordinates +0.5 units along the S-axis and has the net effect of translating the texture -0.5 along the S-axis on the geometry's surface. A rotation of  $\pi/2$  of the texture coordinates results in a  $-\pi/2$  rotation of the texture on the geometry.

The *center* field specifies a translation offset in texture coordinate space about which the *rotation* and *scale* fields are applied. The *scale* field specifies a scaling factor in S, T and R of the texture coordinates about the center point. All *scale* values shall be in the range  $(-\infty, \infty)$ . The *rotation* field specifies a rotation of the texture coordinates about the center point after the scale has been applied. A positive rotation value makes the texture coordinates rotate counterclockwise about the centre, thereby rotating the appearance of the texture itself clockwise. The *translation* field specifies a translation of the texture coordinates.

A 3D transform may be applied to 2D textures. The results are implementation dependent.

## 33.5 Support levels

The 3D Texturing component defines two levels of support as specified in [Table 33.2](#).

**Table 33.2 — 3D texturing component support levels**

Level	Prerequisites	Nodes/Features	Support
<b>1</b>	Core 1 Grouping 1 Shape 1 Rendering 1 Texturing 1		
		<i>X3DTexture3DNode</i>	n/a
		<del>TextureMatrixTransform</del> TextureTransformMatrix3D	All fields fully supported.
		TextureTransform3D	All fields fully supported.
		TextureCoordinate3D	All fields fully supported.

		TextureCoordinate4D	All fields fully supported.
		ComposedTexture3D	All fields fully supported.
		PixelTexture3D	All fields fully supported.
<b>2</b>	Core 1 Grouping 1 Shape 1 Rendering 1 Texturing 1		
		ImageTexture3D	All fields fully supported.



Draft



# Extensible 3D (X3D) Part 1: Architecture and base components

## Annex L (normative)

### MedicalInterchange profile

---

#### L.1 General

This annex defines the X3D components that comprise the MedicalInterchange profile. This annex includes not only the nodes that shall be supported but also which fields in the supported nodes may be ignored.

This profile is targeted towards:

- Exchange of polygonal geometry, volumetric data and accompanying documentation between medical imaging systems.
- Possible implementation in industry-specific applications that use X3D as an interchange format, but link to proprietary databases and hardware.

#### L.2 Topics

[Table L.1](#) provides links to the major topics in this annex.

**Table L.1 — Topics**

- |  |
|--|
| <ul style="list-style-type: none"><li>• <a href="#">L.1 General</a></li><li>• <a href="#">L.2 Topics in this annex</a></li><li>• <a href="#">L.3 Component support</a></li><li>• <a href="#">L.4 Conformance criteria</a></li><li>• <a href="#">L.5 Node set</a></li><li>• <a href="#">L.6 Other limitations</a></li><li>• <a href="#">Table L.1 — Topics</a></li><li>• <a href="#">Table L.2 — Components and levels</a></li><li>• <a href="#">Table L.3 — Nodes for conforming to the MedicalInterchange profile</a></li></ul> |
|--|

- [Table L.4 — Other limitations](#)

## L.3 Component support

[Table L.2](#) lists the components and their levels that shall be supported in the MedicalInterchange profile. [Table L.3](#) and [Table L.4](#) describe limitations on required support for nodes and fields contained within these components.

**Table L.2 — Components and levels**

Component	Level	Reference
Core	1	<a href="#">7.5 Support levels</a>
Time	1	<a href="#">8.5 Support levels</a>
Networking	2	<a href="#">9.5 Support levels</a>
Grouping	3	<a href="#">10.5 Support levels</a>
Rendering	5	<a href="#">11.5 Support levels</a>
Shape	3	<a href="#">12.5 Support levels</a>
Geometry3D	2	<a href="#">13.4 Support levels</a>
Geometry2D	2	<a href="#">14.4 Support levels</a>
Text	1	<a href="#">15.5 Support levels</a>
Lighting	1	<a href="#">17.5 Support levels</a>
Texturing	2	<a href="#">18.5 Support levels</a>
Interpolation	2	<a href="#">19.5 Support levels</a>
Navigation	3	<a href="#">23.4 Support levels</a>
Environmental effects	1	<a href="#">24.5 Support levels</a>
Event utilities	1	<a href="#">30.5 Support levels</a>
Texturing3D	2	<a href="#">33.5 Support levels</a>
Volume rendering	4	<a href="#">41.5 Support levels</a>

## L.4 Conformance criteria

Conformance to this profile shall include conformance criteria defined by the

specifications for those components and levels listed in [Table L.2](#).

In Tables L.3 and L.4, the first column defines the item for which conformance is being defined. In some cases, general limits are defined but are later overridden in specific cases by more restrictive limits. The second column defines the requirements for an X3D file conforming to the MedicalInterchange profile. If an X3D file contains any items that exceed these limits, it may not be possible for an X3D browser conforming to the MedicalInterchange profile to successfully parse that X3D file. The third column defines the minimum complexity for an X3D scene that an X3D browser conforming to the MedicalInterchange profile shall be able to present to the user. Fields flagged as "not supported" may optionally be supported by browsers which conform to the MedicalInterchange profile. The word "ignore" in the minimum browser support column refers only to the display of the item; in particular, *set\_* events to ignored inputOutput fields shall still generate corresponding *\_changed* events.

## L.5 Node set

[Table L.3](#) lists the nodes which shall be supported in the MedicalInterchange profile and specifies any fields in these nodes for which this profile requires less than full support.

**Table L.3 — Nodes for conforming to the MedicalInterchange profile**

Item	X3D File Limit	Minimum Browser Support
Anchor	No restrictions.	Full support.
Arc2D	No restrictions.	Full support.
ArcClose2D	No restrictions.	Full support.
Appearance	No restrictions.	Full support.
Background	No restrictions.	<i>groundAngle</i> and <i>groundColor</i> optionally supported. <i>backURL</i> , <i>frontURL</i> , <i>leftURL</i> , <i>rightURL</i> , <i>topURL</i> optionally supported. <i>skyAngle</i> optionally



		supported. At least one <i>skyColor</i> supported.
Billboard	Restrictions as for all groups.	Full support except as for all groups.
BlendedVolumeStyle	No restrictions.	Full support.
BooleanFilter	No restrictions.	Full support.
BooleanSequencer	No restrictions.	Full support.
BooleanToggle	No restrictions.	Full support.
BooleanTrigger	No restrictions.	Full support.
BoundaryEnhancementVolumeStyle	No restrictions.	Full support.
Box	No restrictions.	Full support
CartoonVolumeStyle	No restrictions.	Full support.
Circle2D	No restrictions.	Full support.
ClipPlane	No restrictions.	Full support.
Collision	Restrictions as for all groups.	Full support except as for all groups. Any navigation behaviour acceptable when collision occurs.
Color	15,000 colours.	15,000 colours.
ColorInterpolator	No restrictions.	Full support.
ColorRGBA	15,000	15,000 colours

	colours.	
ComposedVolumeStyle	No restrictions.	Full support.
CompositeTexture3D	Minimum 512 textures.	Full support.
Cone	No restrictions.	Full support.
Coordinate	65,535 points.	65,535 points.
CoordinateDouble	65,535 points.	65,535 points.
CoordinateInterpolator	No restrictions.	Full support.
Cylinder	No restrictions.	Full support.
DirectionalLight	No restrictions.	Not scoped by parent Group or Transform.
Disk2D	No restrictions.	Full support.
EdgeEnhancementVolumeStyle	No restrictions.	Full support.
FillProperties	No restrictions.	Full support.
FontStyle	No restrictions.	If the values of the text aspects character set, <i>family</i> , <i>style</i> cannot be simultaneously supported, the order of precedence shall be: 1) character set 2) <i>family</i> 3) <i>style</i> . Browser shall display all characters in Table 2 (Basic Latin) and

		Table 3 (Latin-1 Supplement) of ISO/IEC 10646 (see <a href="#">ISO/IEC 10646</a> ).
Group	Restrictions as for all groups.	<i>addChildren</i> optionally supported. <i>removeChildren</i> optionally supported. Otherwise as for all groups.
ImageTexture	JPEG ( <a href="#">2. [JPEG]</a> ) and PNG ( <a href="#">(ISO/IEC 15948)</a> ) format.	JPEG ( <a href="#">2. [JPEG]</a> ) and PNG ( <a href="#">(ISO/IEC 15948)</a> ) format.
ImageTexture3D	DICOM, JPEG ( <a href="#">2. [JPEG]</a> ) and PNG ( <a href="#">(ISO/IEC 15948)</a> ) format.	Full support. Minimum texture size of 256x256x256 pixels
IndexedFaceSet	10 vertices per face. 5000 faces. Less than 15,000 indices.	10 vertices per face. 5000 faces. 15,000 indices in any index field.
IndexedLineSet	15,000 total vertices. 15,000 indices in any index field.	15,000 total vertices. 15,000 indices in any index field.
IndexedTriangleFanSet	5,000 total faces. 15,000 indices in any index field.	5,000 total faces. 15,000 indices in any index field.
IndexedTriangleSet	5,000 total faces. 15,000 indices in	5,000 total faces. 15,000 indices in any

	any index field.	index field.
IndexedTriangleStripSet	5,000 total faces. 15,000 indices in any index field.	5,000 total faces. 15,000 indices in any index field.
Inline	No restrictions	All fields except <i>load</i> which is optionally supported.
IntegerSequencer	No restrictions.	Full support.
IntegerTrigger	No restrictions.	Full support.
IsoSurfaceVolumeData	Minimum dimensions: 512 width, 512 height, 512 depth.	Full support.
LineProperties	No restrictions.	Full support.
LineSet	15,000 total vertices.	15,000 total vertices.
LOD	Restrictions as for all groups.	At least first 4 <i>level/range</i> combinations interpreted, and support as for all groups.
Material	No restrictions.	Full support.
MetadataBoolean	No restrictions.	Full support.
MetadataDouble	No restrictions.	Full support.
MetadataFloat	No restrictions.	Full support.
MetadataInteger	No restrictions.	Full support.

MetadataSet	No restrictions.	Full support.
MetadataString	No restrictions.	Full support.
MultiTexture	No restrictions.	At least one texture displayed per node with any number specified. Full support.
MultiTextureCoordinate	15,000 coordinates.	15,000 coordinates.
MultiTextureTransform	Restrictions as for all groups.	Full support.
NavigationInfo	No restrictions.	<i>avatarSize</i> optionally supported. <i>speed</i> optionally supported. <i>type</i> optionally supported. <i>visibilityLimit</i> optionally supported.
Normal	15,000 normals.	15,000 normals.
NormalInterpolator	No restrictions.	Full support.
OctTree	No restrictions.	Full support.
OpacityMapVolumeStyle	No restrictions.	Full support. 3D transfer functions shall be supported.
OrientationInterpolator	No restrictions.	Full support.
OrthoViewpoint	No restrictions.	Full support.
		512 width. 512

PixelTexture	512 width. 512 height.	height. Display fully transparent and fully opaque pixels.
PixelTexture3D	256 width. 256 height. 256 depth.	256 width. 256 height. 256 depth. Display fully transparent and fully opaque pixels.
PointSet	5,000 points.	5,000 points.
Polyline2D	5,000 points.	5,000 points.
Polypoint2D	5,000 points.	5,000 points.
PositionInterpolator	No restrictions.	Full support.
ProjectionVolumeStyle	No restrictions.	Full support.
Rectangle2D	No restrictions.	Full support.
ScalarInterpolator	No restrictions.	Full support.
SegmentedVolumeData	Minimum dimensions: 512 width, 512 height, 512 depth.	Full support.
ShadedVolumeStyle	No restrictions.	A fields fully supported except shadows. Shadows supported with at least Phong shading. Henyey-Greenstein phase function not required.
Shape	No	Full support.

	restrictions.	
SilhouetteEnhancementVolumeStyle	No restrictions.	Full support.
Sphere	No restrictions.	Full support.
StaticGroup	No restrictions.	Full support.
Switch	No restrictions.	Full support.
Text	100 characters per string. 100 strings.	100 characters per string. 100 strings.
TextureCoordinate	65,535 coordinates.	65,535 coordinates.
TextureCoordinate3D	65,535 coordinates.	65,535 coordinates.
TextureCoordinate4D	65,535 coordinates.	65,535 coordinates.
TextureCoordinateGenerator	No restrictions.	Full support.
TextureMatrixTransformTextureTransformMatrix3D	No restrictions.	Full support.
TextureProperties	No restrictions.	Full support.
TextureTransform	No restrictions.	Full support.
TextureTransform3D	No restrictions.	Full support.
TimeSensor	No restrictions.	<i>pause, isPaused, resumeTime</i> optionally supported.
TimeTrigger	No restrictions.	Full support.
ToneMappedVolumeStyle	No restrictions.	Full support.



Transform	Restrictions as for all groups.	<i>addChildren</i> optionally supported. <i>removeChildren</i> optionally supported. Otherwise, full support except as for all groups.
TriangleFanSet	5,000 triangles per fan. 15,000 total triangles.	5,000 triangles per fan. 15,000 total triangles.
TriangleSet	15,000 triangles	15,000 triangles
TriangleStripSet	5,000 triangles per strip. 15,000 total triangles	5,000 triangles per strip. 15,000 total triangles.
Viewpoint	No restrictions.	Full support.
ViewpointGroup	No restrictions.	Full support.
VolumeData	Minimum dimensions: 512 width, 512 height, 512 depth.	Full support.
WorldInfo	No restrictions.	Full support.

## L.6 Other limitations

[Table L.4](#) specifies other aspects of X3D functionality which are supported by this profile. Note that general items refer only to those specific nodes listed in [Table L.3](#).

**Table L.4 — Other limitations**

Item	X3D File Limit	Minimum Browser Support
All groups	500 children.	500 children. Optionally ignore <i>bboxCenter</i> and <i>bboxSize</i> .

All lights	8 simultaneous lights.	8 simultaneous lights.
Names for DEF/field	50 utf8 octets.	50 utf8 octets.
All <i>url</i> fields	10 URLs.	10 URLs. URN's ignored.
SFBool	No restrictions.	Full support.
SFColor	No restrictions.	Full support.
SFColorRGBA	No restrictions.	Full support.
SFDouble	No restrictions.	Full support. Range $\pm 1e\pm 12$ . Precision $1e-7$ .
SFFloat	No restrictions.	Full support.
SFImage	512 width. 512 height.	512 width. 512 height.
SFInt32	No restrictions.	Full support.
SFMatrix4d	No restrictions.	Full support.
SFMatrix4f	No restrictions.	Full support.
SFNode	No restrictions.	Full support.
SFRotation	No restrictions.	Full support.
SFString	30,000 utf8 octets.	30,000 utf8 octets.
SFTime	No restrictions.	Full support.
SFVec2d	No restrictions.	Full support.
SFVec2f	No restrictions.	Full support.
SFVec3d	No restrictions.	Full support.
SFVec3f	No restrictions.	Full support.
SFVec4d	No restrictions.	Full support.
SFVec4f	No restrictions.	Full support.
MFCColor	15,000 values.	15,000 values.
MFCColorRGBA	15,000 values.	15,000 values.
MFDouble	1000 values.	1000 values.
MFFloat	1,000 values.	1,000 values.

MFInt32	20,000 values.	20,000 values.
MFNode	500 values.	500 values.
MFRotation	1,000 values.	1,000 values.
MFString	30,000 utf8 octets per string, 10 strings.	30,000 utf8 octets per string, 10 strings.
MFTime	1,000 values.	1,000 values.
MFVec2d	15,000 values.	15,000 values.
MFVec2f	15,000 values.	15,000 values.
MFVec3d	15,000 values.	15,000 values.
MFVec3f	15,000 values.	15,000 values.
MFVec4d	15,000 values.	15,000 values.
MFVec4f	15,000 values.	15,000 values.





## Extensible 3D (X3D) Part 1: Architecture and base components

### 13 Geometry3D component

---



#### 13.1 Introduction

##### 13.1.1 Name

The name of this component is "Geometry3D". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 13.1.2 Overview

This clause describes the Geometry3D component of this part of ISO/IEC 19775. This includes how 3D geometry is specified and what shapes are available. [Table 13.1](#) provides links to the major topics in this clause.

**Table 13.1 — Topics**

- [13.1 Introduction](#)
  - [13.1.1 Name](#)
  - [13.1.2 Overview](#)
- [13.2 Concepts](#)
  - [13.2.1 Overview of geometry](#)
  - [13.2.2 Shape and geometry nodes](#)
  - [13.2.3 Geometric property nodes](#)
  - [13.2.4 Appearance nodes](#)
  - [13.2.5 Common geometry fields](#)
- [13.3 Node reference](#)
  - [13.3.1 Box](#)
  - [13.3.2 Cone](#)
  - [13.3.3 Cylinder](#)
  - [13.3.4 ElevationGrid](#)
  - [13.3.5 Extrusion](#)
    - [13.3.5.1 Syntax](#)
    - [13.3.5.2 Overview](#)

- 13.3.5.3 Algorithmic description
    - [13.3.5.4 Special cases](#)
        - [13.3.5.4.1 Overview](#)
        - [13.3.5.4.2 Number of scale or orientation values](#)
        - [13.3.5.4.3 Collinear spine points](#)
        - [13.3.5.4.4 Coincident spine points](#)
        - [13.3.5.4.5 Number of distinct spine points](#)
      - [13.3.5.5 Common cases](#)
      - [13.3.5.6 Other fields](#)
    - [13.3.6 IndexedFaceSet](#)
    - [13.3.7 Sphere](#)
- [13.4 Support levels](#)
- [Figure 13.1 — Box node](#)
- [Figure 13.2 — Cone node](#)
- [Figure 13.3 — Cylinder node](#)
- [Figure 13.4 — ElevationGrid node](#)
- [Figure 13.5 — Spine-aligned cross-section plane at a spine point](#)
- [Figure 13.6 — IndexedFaceSet texture default mapping](#)
- [Figure 13.7 — ImageTexture for IndexedFaceSet in Figure 13.6](#)
- [Figure 13.8 — Sphere node](#)
- [Table 13.1 — Topics](#)
- [Table 13.2 — Geometry3D component support levels](#)

## 13.2 Concepts

### 13.2.1 Overview of geometry

The geometry component consists of four types of nodes: shape, geometry, geometry property, and appearance. Together, these node types are used to describe the visual elements of a X3D world.

### 13.2.2 Shape and geometry nodes

The [Shape](#) node associates a geometry node with nodes that define that geometry's appearance. Shape nodes must be part of the transformation hierarchy to have any visible result, and the transformation hierarchy must contain Shape nodes for any geometry to be visible (the only nodes that render visible results are Shape nodes and the background nodes in [24 Environmental effects](#)). A Shape node contains exactly one geometry node in its *geometry* field, which is of type [X3DGeometryNode](#). For more on the Shape node, see [12 Shape component](#).

Other components may define additional geometry node types.

### 13.2.3 Geometric property nodes

Several geometry nodes contain geometric property nodes such as [Coordinate](#), [Color](#), [ColorRGBA](#), and/or [Normal](#). These nodes are specified in [11 Rendering component](#). The [X3DTextureCoordinate](#) nodes specified in [18 Texturing component](#) are also geometry property nodes.

### 13.2.4 Appearance nodes

[Shape](#) nodes may specify an [Appearance](#) node that describes the appearance properties (material and texture) to be applied to the Shape's geometry. Appearance is described in [12 Shape component](#).

### 13.2.5 Common geometry fields

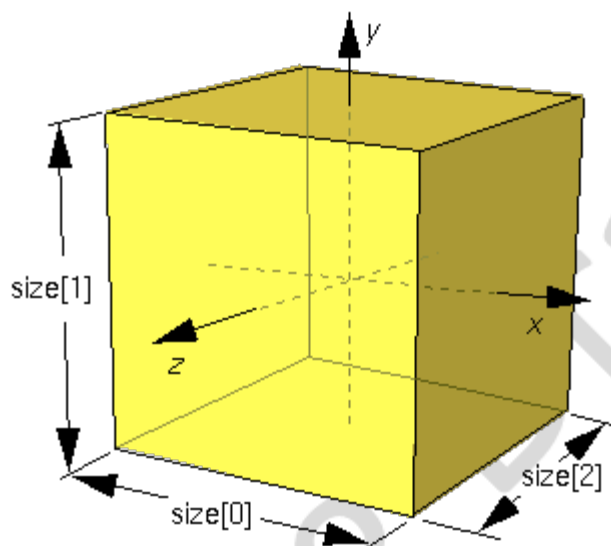
Several 3D geometry nodes share common fields to describe attributes. These fields specify the vertex ordering, if the shape is solid, if the shape contains convex faces, and at what angle a crease appears between faces, and are named *ccw*, *solid*, *convex* and *creaseAngle*, respectively. Common 3D geometry fields are described in [11 Rendering component](#).

## 13.3 Node reference

### 13.3.1 Box

```
Box : X3DGeometryNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFVec3f [] size 2 2 2 (0,∞)
  SFBool [] solid TRUE
}
```

The Box node specifies a rectangular parallelepiped box centred at (0, 0, 0) in the local coordinate system and aligned with the local coordinate axes. By default, the box measures 2 units in each dimension, from -1 to +1. The *size* field specifies the extents of the box along the X-, Y-, and Z-axes respectively and each component value shall be greater than zero. [Figure 13.1](#) illustrates the Box node.



## Figure 13.1 — Box node

Textures are applied individually to each face of the box. On the front (+Z), back (-Z), right (+X), and left (-X) faces of the box, when viewed from the outside with the +Y-axis up, the texture is mapped onto each face with the same orientation as if the image were displayed normally in 2D. On the top face of the box (+Y), when viewed from above and looking down the Y-axis toward the origin with the -Z-axis as the view up direction, the texture is mapped onto the face with the same orientation as if the image were displayed normally in 2D. On the bottom face of the box (-Y), when viewed from below looking up the Y-axis toward the origin with the +Z-axis as the view up direction, the texture is mapped onto the face with the same orientation as if the image were displayed normally in 2D. [TextureTransform](#) affects the texture coordinates of the Box (see [18.4.8 TextureTransform](#)).

The *solid* field determines whether the box is visible when viewed from the inside. [11.2.3 Common geometry fields](#) provides a complete description of the *solid* field.

### 13.3.2 Cone

```

Cone : X3DGeometryNode {
  SFNode [in,out] metadata  NULL [X3DMetadataObject]
  SFBool [in,out] bottom    TRUE
  SFFloat [] bottomRadius  1 (0,∞)
  SFFloat [] height        2 (0,∞)
  SFBool [in,out] side      TRUE
  SFBool [] solid          TRUE
}

```

The Cone node specifies a cone which is centred in the local coordinate system and whose central axis is aligned with the local Y-axis. The *bottomRadius* field specifies the radius of the cone's base, and the *height* field specifies the height of the cone from the centre of the base to the apex. By default, the cone has a radius of 1.0 at the bottom and a height of 2.0, with its apex at  $y = height/2$  and its bottom at  $y = -height/2$ . Both *bottomRadius* and *height* shall be greater than zero. [Figure 13.2](#) illustrates the Cone node.

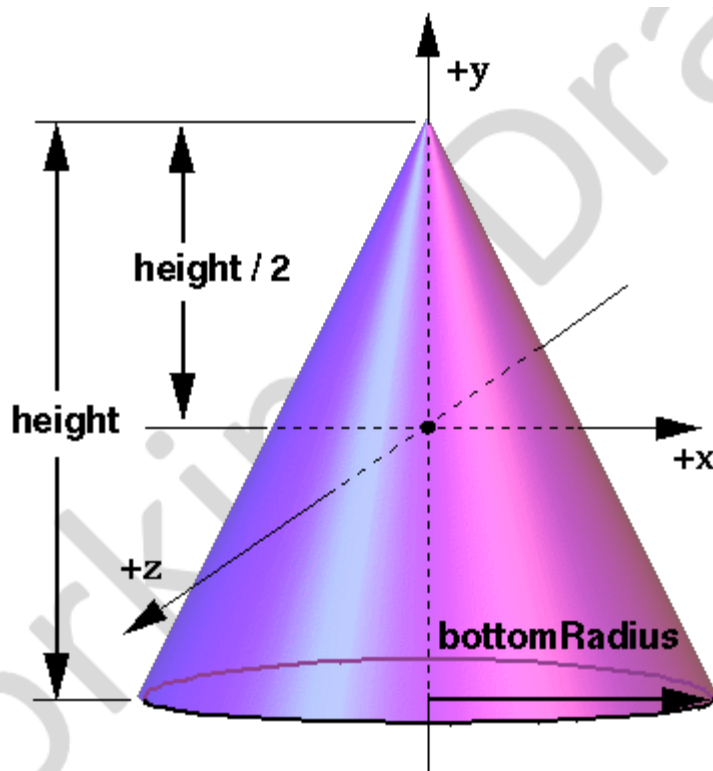


Figure 13.2 — Cone node

The *side* field specifies whether sides of the cone are created and the *bottom* field specifies whether the bottom cap of the cone is created. A value of `TRUE` specifies that this part of the cone exists, while a value of `FALSE` specifies that this part does not exist (not rendered or eligible for collision or sensor intersection tests).

When a texture is applied to the sides of the cone, the texture wraps counterclockwise (from above) starting at the back of the cone. The texture has a vertical seam at the back in the  $X=0$  plane, from the apex  $(0, height/2, 0)$  to the point  $(0, -height/2, -bottomRadius)$ . For the bottom cap, a circle is cut out of the texture square centred at  $(0, -height/2, 0)$  with dimensions  $(2 \times bottomRadius)$  by  $(2 \times bottomRadius)$ . The bottom cap texture appears right side up when the top of the cone is rotated towards the  $-Z$ -axis. [TextureTransform](#) affects the texture coordinates of the Cone (see [18.4.8 TextureTransform](#)).

The *solid* field determines whether the cone is visible when viewed from the inside. [11.2.3 Common geometry fields](#) provides a complete description of the *solid* field.

This geometry node is fundamentally a mathematical representation. Displayed geometry shall have sufficient rendering quality that surface and silhouette edges appear smooth, including when textures are applied.

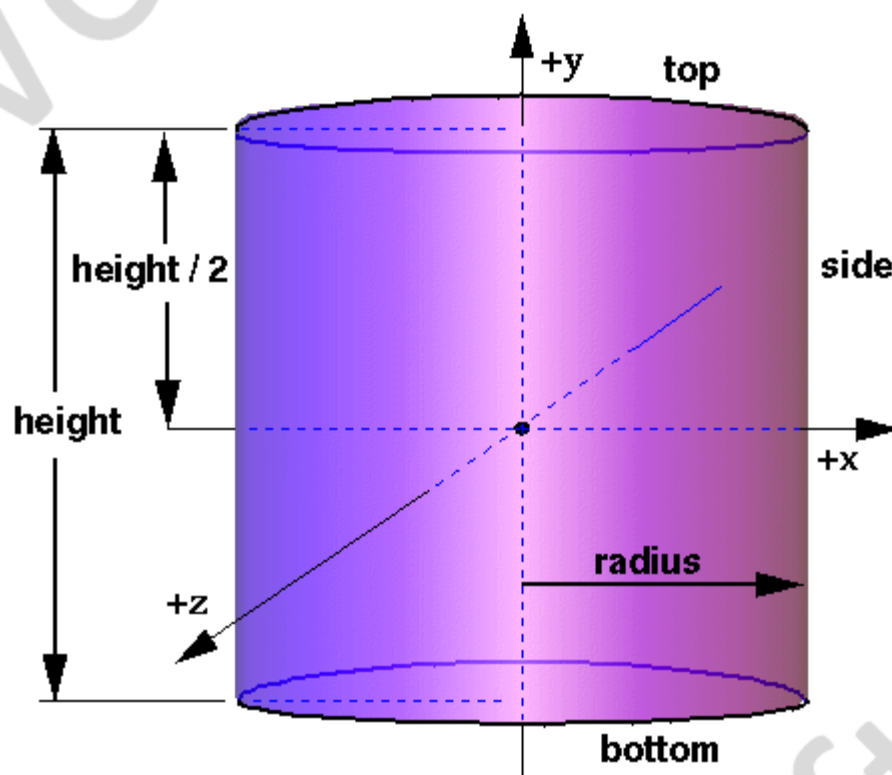
### 13.3.3 Cylinder

```
Cylinder : X3DGeometryNode {
  SFNode [in.out] metadata NULL [X3DMetadataObject]
  SFBool [in.out] bottom TRUE
  SFFloat [] height 2 (0,∞)
  SFFloat [] radius 1 (0,∞)
  SFBool [in.out] side TRUE
  SFBool [] solid TRUE
  SFBool [in.out] top TRUE
}
```



The Cylinder node specifies a capped cylinder centred at (0,0,0) in the local coordinate system and with a central axis oriented along the local Y-axis. By default, the cylinder is sized at "-1" to "+1" in all three dimensions. The *radius* field specifies the radius of the cylinder and the *height* field specifies the height of the cylinder along the central axis. Both *radius* and *height* shall be greater than zero. [Figure 13.3](#) illustrates the Cylinder node.

The cylinder has three *parts*: the *side*, the *top* ( $Y = +\text{height}/2$ ) and the *bottom* ( $Y = -\text{height}/2$ ). Each part has an associated SFBool field that indicates whether the part exists (`TRUE`) or does not exist (`FALSE`). Parts which do not exist are not rendered and not eligible for intersection tests (EXAMPLE collision detection or sensor activation).



**Figure 13.3 — Cylinder node**

When a texture is applied to a cylinder, it is applied differently to the sides, top, and bottom. On the sides, the texture wraps counterclockwise (from above) starting at the back of the cylinder. The texture has a vertical seam at the back, intersecting the  $X=0$  plane. For the top and bottom caps, a circle is cut out of the unit texture squares centred at  $(0, \pm\text{height}/2, 0)$  with dimensions  $2 \times \text{radius}$  by  $2 \times \text{radius}$ . The top texture appears right side up when the top of the cylinder is tilted toward the  $+Z$ -axis, and the bottom texture appears right side up when the top of the cylinder is tilted toward the  $-Z$ -axis. [TextureTransform](#) affects the texture coordinates of the Cylinder node (see [18.4.8 TextureTransform](#)).

The *solid* field determines whether the cylinder is visible when viewed from the inside. [11.2.3 Common geometry fields](#) provides a complete description of the *solid* field.

This geometry node is fundamentally a mathematical representation. Displayed geometry shall have sufficient rendering quality that surface and silhouette edges

appear smooth, including when textures are applied.

### 13.3.4 ElevationGrid

```

ElevationGrid : X3DGeometryNode {
  MFFloat [in]  set_height
  MFNode [in,out] attrib [] [X3DVertexAttributeNode]
  SFNode [in,out] color NULL [X3DColorNode]
  SFNode [in,out] fogCoord NULL [FogCoordinate]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFNode [in,out] normal NULL [X3DNormalNode]
  SFNode [in,out] texCoord NULL [X3DTextureCoordinateNode]
  SFBool [] ccw TRUE
  SFBool [] colorPerVertex TRUE
  SFFloat [] creaseAngle 0 [0,∞)
  MFFloat [] height [] (-∞,∞)
  SFBool [] normalPerVertex TRUE
  SFBool [] solid TRUE
  SFInt32 [] xDimension 0 [0,∞)
  SFFloat [] xSpacing 1.0 (0,∞)
  SFInt32 [] zDimension 0 [0,∞)
  SFFloat [] zSpacing 1.0 (0,∞)
}

```

The `ElevationGrid` node specifies a uniform rectangular grid of varying height in the  $Y=0$  plane of the local coordinate system. The geometry is described by a scalar array of height values that specify the height of a surface above each point of the grid.

The `xDimension` and `zDimension` fields indicate the number of elements of the grid `height` array in the X and Z directions. Both `xDimension` and `zDimension` shall be greater than or equal to zero. If either the `xDimension` or the `zDimension` is less than two, the `ElevationGrid` contains no quadrilaterals. The vertex locations for the rectangles are defined by the `height` field and the `xSpacing` and `zSpacing` fields:

- The `height` field is an `xDimension` by `zDimension` array of scalar values representing the height above the grid for each vertex.
- The `xSpacing` and `zSpacing` fields indicate the distance between vertices in the X and Z directions respectively, and shall be greater than zero.

Thus, the vertex corresponding to the point  $P[i, j]$  on the grid is placed at:

$$P[i,j].x = xSpacing \times i$$

$$P[i,j].y = height[i + j \times xDimension]$$

$$P[i,j].z = zSpacing \times j$$

where  $0 \leq i < xDimension$  and  $0 \leq j < zDimension$ ,  
and  $P[0,0]$  is `height[0]` units above/below the origin of the local coordinate system

If the rendering algorithm being used requires tessellation, the quadrilaterals are split into triangles along the seam starting at the initial vertex of the quadrilateral and proceeding to the opposite vertex. The positive direction for the normal of each triangle shall be on the same side of the quadrilateral. The triangles are defined either counterclockwise or clockwise depending on the value of the `ccw` field.

**EXAMPLE** In [Figure 13.4](#) with the `ccw` field set to `TRUE`, the first polygon is split into triangles with vertices  $[0, 5, 6]$  and vertices  $[0, 6, 1]$ .

The `set_height` inputOnly field allows the height `MFFloat` field to be changed to support animated `ElevationGrid` nodes.

The `color` field specifies per-vertex or per-quadrilateral colours for the `ElevationGrid` node depending on the value of `colorPerVertex`. If the `color` field is `NULL`, the

ElevationGrid node is rendered with the overall attributes of the [Shape](#) node enclosing the ElevationGrid node (see [12 Shape component](#)).

The *colorPerVertex* field determines whether colours specified in the *color* field are applied to each vertex or each quadrilateral of the ElevationGrid node. If *colorPerVertex* is `FALSE` and the *color* field is not `NULL`, the *color* field shall specify a node derived from [X3DColorNode](#) containing at least  $(xDimension-1) \times (zDimension-1)$  colours; one for each quadrilateral, ordered as follows:

$$\text{QuadColor}[i,j] = \text{Color}[i + j \times (xDimension-1)]$$

where  $0 \leq i < xDimension-1$  and  $0 \leq j < zDimension-1$ ,  
and  $\text{QuadColor}[i,j]$  is the colour for the quadrilateral defined by  $\text{height}[i+j \times xDimension]$ ,  $\text{height}[(i+1)+j \times xDimension]$ ,  $\text{height}[(i+1)+(j+1) \times xDimension]$  and  $\text{height}[i+(j+1) \times xDimension]$

If *colorPerVertex* is `TRUE` and the *color* field is not `NULL`, the *color* field shall specify a node derived from [X3DColorNode](#) containing at least  $xDimension \times zDimension$  colours, one for each vertex, ordered as follows:

$$\text{VertexColor}[i,j] = \text{Color}[i + j \times xDimension]$$

where  $0 \leq i < xDimension$  and  $0 \leq j < zDimension$ ,  
and  $\text{VertexColor}[i,j]$  is the colour for the vertex defined by  $\text{height}[i+j \times xDimension]$

The *normal* field specifies per-vertex or per-quadrilateral normals for the ElevationGrid node. If the *normal* field is `NULL`, the browser shall automatically generate normals, using the *creaseAngle* field to determine if and how normals are smoothed across the surface (see [11.2.3 Common geometry fields](#)).

The *normalPerVertex* field determines whether normals are applied to each vertex or each quadrilateral of the ElevationGrid node depending on the value of *normalPerVertex*. If *normalPerVertex* is `FALSE` and the *normal* node is not `NULL`, the *normal* field shall specify a node derived from [X3DNormalNode](#) containing at least  $(xDimension-1) \times (zDimension-1)$  normals; one for each quadrilateral, ordered as follows:

$$\text{QuadNormal}[i,j] = \text{Normal}[i + j \times (xDimension-1)]$$

where  $0 \leq i < xDimension-1$  and  $0 \leq j < zDimension-1$ ,  
and  $\text{QuadNormal}[i,j]$  is the normal for the quadrilateral defined by  $\text{height}[i+j \times xDimension]$ ,  $\text{height}[(i+1)+j \times xDimension]$ ,  $\text{height}[(i+1)+(j+1) \times xDimension]$  and  $\text{height}[i+(j+1) \times xDimension]$

If *normalPerVertex* is `TRUE` and the *normal* field is not `NULL`, the *normal* field shall specify a node derived from [X3DNormalNode](#) containing at least  $xDimension \times zDimension$  normals; one for each vertex, ordered as follows:

$$\text{VertexNormal}[i,j] = \text{Normal}[i + j \times xDimension]$$

where  $0 \leq i < xDimension$  and  $0 \leq j < zDimension$ ,  
and  $\text{VertexNormal}[i,j]$  is the normal for the vertex defined by  $\text{height}[i+j \times xDimension]$

The *texCoord* field specifies per-vertex texture coordinates for the ElevationGrid node. If *texCoord* is `NULL`, default texture coordinates are applied to the geometry. The default texture coordinates range from (0,0) at the first vertex to (1,1) at the last vertex. The S texture coordinate is aligned with the positive X-axis, and the T texture coordinate with positive Z-axis. If *texCoord* is not `NULL`, it shall specify a node derived from [X3DTextureCoordinateNode](#) containing at least  $(xDimension) \times (zDimension)$  texture coordinates; one for each vertex, ordered as follows:

VertexTexCoord[i,j] = TextureCoordinate[ i + j × xDimension]

where  $0 \leq i < xDimension$  and  $0 \leq j < zDimension$ ,  
and VertexTexCoord[i,j] is the texture coordinate for the vertex  
defined by height[i+j × xDimension]

The *ccw*, *solid*, and *creaseAngle* fields are described in [11.2.3 Common geometry fields](#).

By default, the quadrilaterals are defined with a counterclockwise ordering. Hence, the Y-component of the normal is positive. Setting the *ccw* field to `FALSE` reverses the normal direction. Backface culling is enabled when the *solid* field is `TRUE`.

See [Figure 13.4](#) for a depiction of the ElevationGrid node.

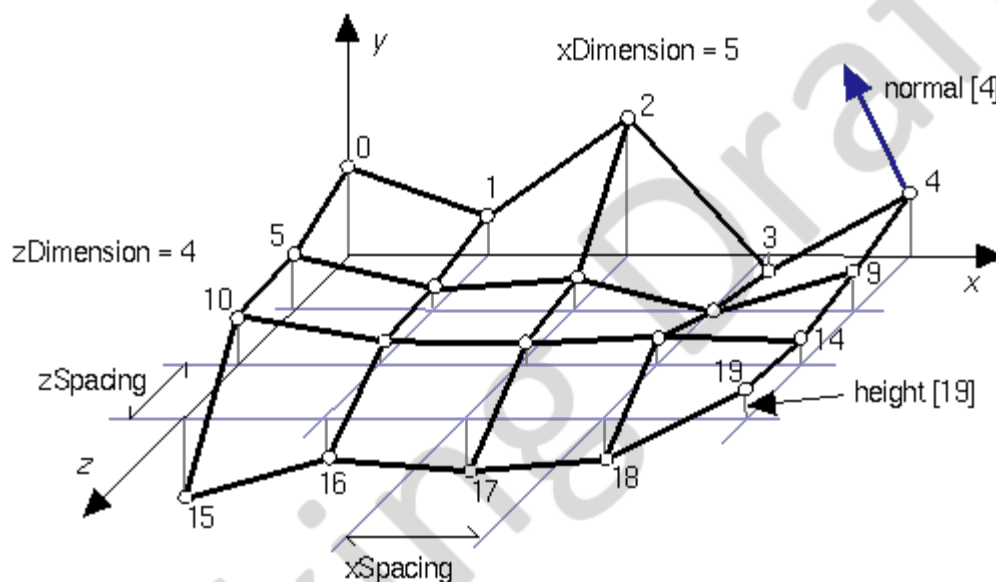


Figure 13.4 — ElevationGrid node

## 13.3.5 Extrusion

### 13.3.5.1 Syntax

```

Extrusion : X3DGeometryNode {
  MFVec2f [in]  set_crossSection
  MFRotation [in]  set_orientation
  MFVec2f [in]  set_scale
  MFVec3f [in]  set_spine
  SFNode [in,out] metadata          NULL          [X3DMetadataObject]
  SFBool []    beginCap             TRUE
  SFBool []    ccw                  TRUE
  SFBool []    convex               TRUE
  SFFloat []   creaseAngle           0              [0,∞)
  MFVec2f []   crossSection          [1 1 1 -1 -1 -1 1 1 1] (-∞,∞)
  SFBool []   endCap                TRUE
  MFRotation [] orientation           0 0 1 0         [-1,1] or (-∞,∞)
  MFVec2f []   scale                 1 1           (0,∞)
  SFBool []   solid                 TRUE
  MFVec3f []   spine                 [0 0 0 1 0]   (-∞,∞)
}

```

### 13.3.5.2 Overview

The Extrusion node specifies geometric shapes based on a two dimensional cross-section extruded along a three dimensional spine in the local coordinate system. The cross-section can be scaled and rotated at each spine point to produce a wide variety of

shapes.

An Extrusion node is defined by:

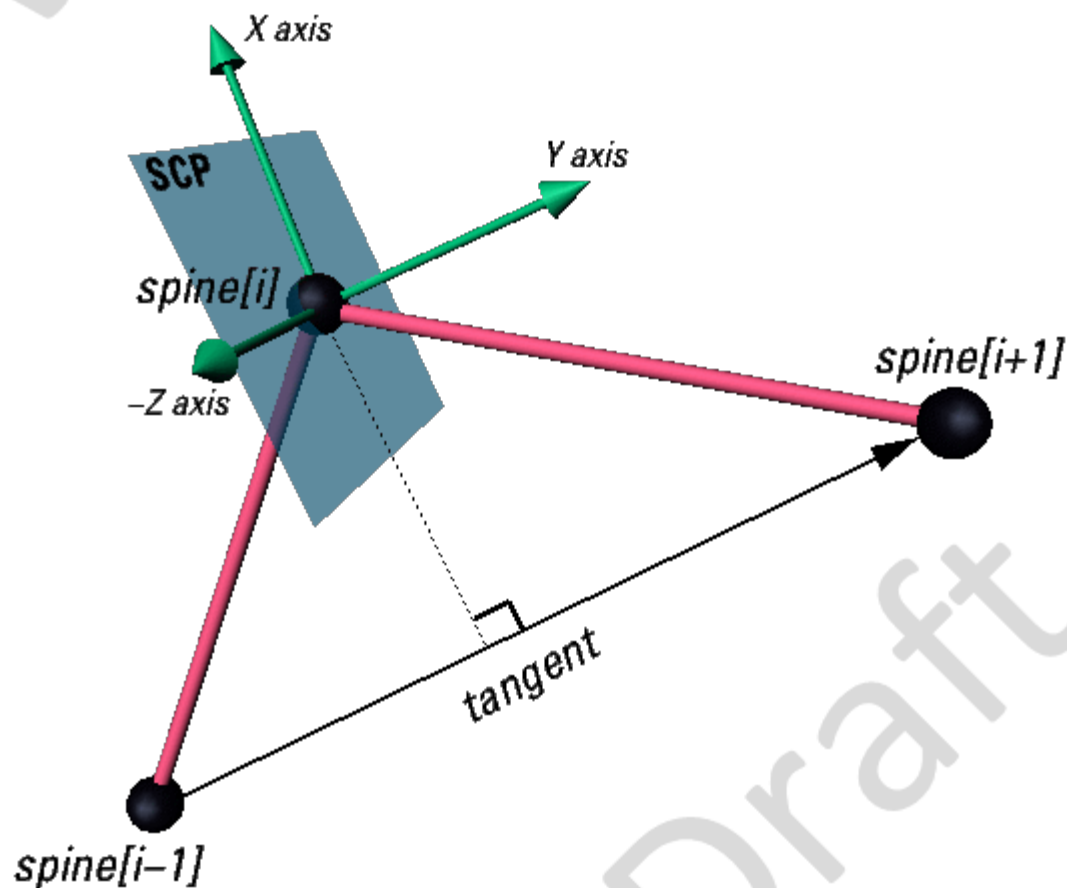
- a. a 2D *crossSection* piecewise linear curve (described as a series of connected vertices);
- b. a 3D *spine* piecewise linear curve (also described as a series of connected vertices);
- c. a list of 2D *scale* parameters;
- d. a list of 3D *orientation* parameters.

The *scale* values shall be positive.

### 13.3.5.3 Algorithmic description

Shapes are constructed as follows. The cross-section curve, which starts as a curve in the  $Y=0$  plane, is first scaled about the origin by the first *scale* parameter (first value scales in  $X$ , second value scales in  $Z$ ). It is then translated by the first spine point and oriented using the first *orientation* parameter (as explained later). The same procedure is followed to place a cross-section at the second spine point, using the second scale and orientation values. Corresponding vertices of the first and second cross-sections are then connected, forming a quadrilateral polygon between each pair of vertices. This same procedure is then repeated for the rest of the spine points, resulting in a surface extrusion along the spine.

The final orientation of each cross-section is computed by first orienting it relative to the spine segments on either side of point at which the cross-section is placed. This is known as the *spine-aligned cross-section plane* (SCP), and is designed to provide a smooth transition from one spine segment to the next (see [Figure 13.5](#)).



**Figure 13.5 — Spine-aligned cross-section plane (SCP) at a spine point.**

The SCP for each point is determined by first computing its Y-axis and Z-axis, then taking the cross product of these to determine the X-axis. These three axes are then used to determine the rotation value needed to rotate the  $Y=0$  plane to the SCP. This results in a normal to the plane that is the approximate tangent of the spine at the point, as shown in Figure 13.5. First the Y-axis is determined, as follows:

Let  $n$  be the number of spines and let  $i$  be the index variable satisfying  $0 \leq i < n$ :

- a. For all points other than the first or last: The Y-axis for  $spine[i]$  is found by normalizing the vector defined by  $(spine[i+1] - spine[i-1])$ .
- b. If the spine curve is closed: The SCP for the first and last points is the same and is found using  $(spine[1] - spine[n-2])$  to compute the Y-axis.
- c. If the spine curve is not closed: The Y-axis used for the first point is the vector from  $spine[0]$  to  $spine[1]$ , and for the last it is the vector from  $spine[n-2]$  to  $spine[n-1]$ .

The Z-axis is determined as follows:

- d. For all points other than the first or last: Take the following cross-product:

$$Z = (spine[i+1] - spine[i]) \times (spine[i-1] - spine[i])$$

- e. If the spine curve is closed: The SCP for the first and last points is the same and is



found by taking the following cross-product:

$$Z = (\text{spine}[1] - \text{spine}[0]) \times (\text{spine}[n-2] - \text{spine}[0])$$

- f. *If the spine curve is not closed:* The Z-axis used for the first spine point is the same as the Z-axis for *spine[1]*. The Z-axis used for the last spine point is the same as the Z-axis for *spine[n-2]*.
- g. After determining the Z-axis, its dot product with the Z-axis of the previous spine point is computed. If this value is negative, the Z-axis is flipped (multiplied by  $-1$ ). In most cases, this prevents small changes in the spine segment angles from flipping the cross-section 180 degrees.

Once the Y- and Z-axes have been computed, the X-axis can be calculated as their cross-product.

Each SCP is then rotated by the corresponding orientation value. This rotation is performed relative to the SCP itself. For example, to impart twist in the cross-section, a rotation about the local Y-axis (0 1 0) would be used. Other orientation values are valid and may rotate the cross-section out of the plane of the original SCP.

### 13.3.5.4 Special cases

#### 13.3.5.4.1 Overview

There are a number of special cases require specific handling. These concern the numbers of values or points, and collinear or coincident spine points.

#### 13.3.5.4.2 Number of scale or orientation values

If the number of *scale* or *orientation* values is greater than the number of spine points, the excess values are ignored. If they contain one value, it is applied at all spine points. The results are undefined if the number of scale or orientation values is greater than one but less than the number of spine points.

#### 13.3.5.4.3 Collinear spine points

If the three points used in computing the Z-axis are collinear, the cross-product is zero so the value from the previous point is used instead.

If the Z-axis of the first point is undefined (because the spine is not closed and the first two spine segments are collinear) then the Z-axis for the first spine point with a defined Z-axis is used.

If the entire spine is collinear, the SCP **for all the spine points** is computed by finding the rotation of a vector along the positive Y-axis ( $\mathbf{v}_1$ ) to the vector ( $\mathbf{v}_2$ ) **defined by (spine[n] - spine [0]), where spine[n] is the first spine point not coincident with spine [0]. If  $\mathbf{v}_2$  is parallel to and in the direction of the negative-Y axis, the rotation will be a 180 degree rotation about the Z-axis.** The  $Y=0$  plane is then rotated by this value.

#### 13.3.5.4.4 Coincident spine points

If two or more sequential points in a spine array are coincident, they are each treated as a single point when computing the corresponding SCP, and each will have an identical SCP.

Note: This case is useful when animating the spine array without needing to simultaneously modify the corresponding orientation and scale arrays.

If each coincident point has a different orientation value, the surface is constructed by connecting edges of the cross-sections as normal. This is useful in creating revolved surfaces.

Note: Combining coincident and non-coincident spine segments, as well as other combinations, can lead to interpenetrating surfaces which the extrusion algorithm makes no attempt to avoid.

#### 13.3.5.4.5 Number of distinct spine points

If only 2 distinct, non-coincident, spine points are provided, the corresponding SCP planes for each are perpendicular to the vector defined by these two points.

If fewer than 2 non-coincident spine points are provided, the extrusion is not well defined and no results are rendered.

#### 13.3.5.5 Common cases

The following common cases are among the effects which are supported by the Extrusion node:

##### *Surfaces of revolution:*

If the cross-section is an approximation of a circle and the spine is straight, the Extrusion is equivalent to a surface of revolution, where the *scale* parameters define the size of the cross-section along the spine.

##### *Uniform extrusions:*

If the *scale* is (1, 1) and the spine is straight, the cross-section is extruded uniformly without twisting or scaling along the spine. The result **forms a parallelepiped** with a uniform cross section.

##### *Bend/twist/taper objects:*

These shapes are the result of using all fields. The spine curve bends the extruded shape defined by the cross-section, the orientation parameters (given as rotations about the Y-axis) twist it around the spine, and the scale parameters taper it (by scaling about the spine).

#### 13.3.5.6 Other fields

Extrusion has three *parts*: the *sides*, the *beginCap* (the surface at the initial end of the spine) and the *endCap* (the surface at the final end of the spine). The caps have an associated SFBool field that indicates whether each exists (`TRUE`) or doesn't exist (`FALSE`).

When the *beginCap* or *endCap* fields are specified as `TRUE`, planar cap surfaces will be generated regardless of whether the *crossSection* is a closed curve. If *crossSection* is not a closed curve, the caps are generated by adding a final point to *crossSection* that is equal to the initial point. An open surface can still have a cap, resulting (for a simple



case) in a shape analogous to a soda can sliced in half vertically. These surfaces are generated even if *spine* is also a closed curve. If a field value is `FALSE`, the corresponding cap is not generated.

Texture coordinates are automatically generated by Extrusion nodes. Textures are mapped so that the coordinates range in the U direction from 0 to 1 along the *crossSection* curve (with 0 corresponding to the first point in *crossSection* and 1 to the last) and in the V direction from 0 to 1 along the *spine* curve (with 0 corresponding to the first listed *spine* point and 1 to the last). If either the *endCap* or *beginCap* exists, the *crossSection* curve is uniformly scaled and translated so that the larger dimension of the cross-section (X or Z) produces texture coordinates that range from 0.0 to 1.0. The *beginCap* and *endCap* textures' S and T directions correspond to the X and Z directions in which the *crossSection* coordinates are defined.

The browser shall automatically generate normals for the Extrusion node, using the *creaseAngle* field to determine if and how normals are smoothed across the surface. Normals for the caps are generated along the Y-axis of the SCP, with the ordering determined by viewing the cross-section from above (looking along the negative Y-axis of the SCP). By default, a *beginCap* with a counterclockwise ordering shall have a normal along the negative Y-axis. An *endCap* with a counterclockwise ordering shall have a normal along the positive Y-axis.

Each quadrilateral making up the sides of the extrusion are ordered from the bottom cross-section (the one at the earlier spine point) to the top. So, one quadrilateral has the points:

```
spine[0](crossSection[0], crossSection[1])
spine[1](crossSection[1], crossSection[0])
```

in that order. By default, normals for the sides are generated as described in [13.2.2 Shape and geometry nodes](#).

For instance, a circular crossSection with counter-clockwise ordering and the default spine form a cylinder. With *solid* `TRUE` and *ccw* `TRUE`, the cylinder is visible from the outside. Changing *ccw* to `FALSE` makes it visible from the inside.

The *ccw*, *solid*, *convex*, and *creaseAngle* fields are described in [11.2.3 Common geometry fields](#).

### 13.3.6 IndexedFaceSet

```
IndexedFaceSet : X3DComposedGeometryNode {
  MFInt32 [in]  set_colorIndex
  MFInt32 [in]  set_coordIndex
  MFInt32 [in]  set_normalIndex
  MFInt32 [in]  set_texCoordIndex
  MFNode [in,out] attrib      [] [X3DVertexAttributeNode]
  SFNode [in,out] color      NULL [X3DColorNode]
  SFNode [in,out] coord      NULL [X3DCoordinateNode]
  SFNode [in,out] fogCoord    NULL [FogCoordinate]
  SFNode [in,out] metadata    NULL [X3DMetadataObject]
  SFNode [in,out] normal      NULL [X3DNormalNode]
  SFNode [in,out] texCoord    NULL [X3DTextureCoordinateNode]
  SFBool []    ccw            TRUE
  MFInt32 []   colorIndex     [] [0,∞) or -1
  SFBool []   colorPerVertex  TRUE
  SFBool []   convex          TRUE
  MFInt32 []  coordIndex      [] [0,∞) or -1
  SFFloat []  creaseAngle     0 [0,∞)
  MFInt32 []  normalIndex     [] [0,∞) or -1
  SFBool []   normalPerVertex TRUE
  SFBool []   solid           TRUE
  MFInt32 []  texCoordIndex   [] [-1,∞)
}
```

The IndexedFaceSet node represents a 3D shape formed by constructing faces (polygons) from vertices listed in the *coord* field. The *coord* field contains a Coordinate node that defines the 3D vertices referenced by the *coordIndex* field. IndexedFaceSet uses the indices in its *coordIndex* field to specify the polygonal faces by indexing into the coordinates in the [Coordinate](#) node. An index of "-1" indicates that the current face has ended and the next one begins. The last face may be (but does not have to be) followed by a "-1" index. If the greatest index in the *coordIndex* field is N, the Coordinate node shall contain N+1 coordinates (indexed as 0 to N). Each face of the IndexedFaceSet shall have:

- a. at least three non-coincident vertices;
- b. vertices that define a planar polygon;
- c. vertices that define a non-self-intersecting polygon.

Otherwise, The results are undefined.

The IndexedFaceSet node is specified in the local coordinate system and is affected by the transformations of its ancestors.

Descriptions of the *coord*, *normal*, and *texCoord* fields are provided in [Coordinate](#), [X3DNormalNode](#), and [X3DTextureCoordinateNode](#), respectively.

Details on lighting equations and the interaction between *color* field, *normal* field, textures, materials, and geometries are provided in [11 Rendering component](#) and [12 Shape component](#).

If the *color* field is not `NULL`, it shall contain a node derived from [X3DColorNode](#) whose colours are applied to the vertices or faces of the IndexedFaceSet as follows:

- d. If *colorPerVertex* is `FALSE`, colours are applied to each face, as follows:
  1. If the *colorIndex* field is not empty, one colour is used for each face of the IndexedFaceSet. There shall be at least as many indices in the *colorIndex* field as there are faces in the IndexedFaceSet. If the greatest index in the *colorIndex* field is N, there shall be N+1 colours in the *X3DColorNode*. The *colorIndex* field shall not contain any negative entries.
  2. If the *colorIndex* field is empty, the colours in the *X3DColorNode* node are applied to each face of the IndexedFaceSet in order. There shall be at least as many colours in the *X3DColorNode* node as there are faces.
- e. If *colorPerVertex* is `TRUE`, colours are applied to each vertex, as follows:
  1. If the *colorIndex* field is not empty, colours are applied to each vertex of the IndexedFaceSet in exactly the same manner that the *coordIndex* field is used to choose coordinates for each vertex from the Coordinate node. The *colorIndex* field shall contain at least as many indices as the *coordIndex* field, and shall contain end-of-face markers (-1) in exactly the same places as the *coordIndex* field. If the greatest index in the *colorIndex* field is N, then there shall be N+1 colours in the *X3DColorNode* node.
  2. If the *colorIndex* field is empty, the *coordIndex* field is used to choose colours from the *X3DColorNode* node. If the greatest index in the *coordIndex* field is N, then there shall be N+1 colours in the *X3DColorNode* node.

If the *color* field is `NULL`, the geometry shall be rendered normally using the Material and texture defined in the Appearance node (see [12 Shape component](#) for details).

If the *normal* field is not `NULL`, it shall contain a node derived from *X3DNormalNode* whose normals are applied to the vertices or faces of the IndexedFaceSet in a manner exactly equivalent to that described above for applying colours to vertices/faces (where *normalPerVertex* corresponds to *colorPerVertex* and *normalIndex* corresponds to *colorIndex*). If the *normal* field is `NULL`, the browser shall automatically generate normals, using *creaseAngle* to determine if and how normals are smoothed across shared vertices (see [11.2.3 Common geometry fields](#)).

If the *texCoord* field is not `NULL`, it shall contain a node derived from *X3DTextureCoordinateNode*. The texture coordinates in that node are applied to the vertices of the IndexedFaceSet as follows:

- f. If the *texCoordIndex* field is not empty, then it is used to choose texture coordinates for each vertex of the IndexedFaceSet in exactly the same manner that the *coordIndex* field is used to choose coordinates for each vertex from the Coordinate node. The *texCoordIndex* field shall contain at least as many indices as the *coordIndex* field, and shall contain end-of-face markers (-1) in exactly the same places as the *coordIndex* field. If the greatest index in the *texCoordIndex* field is N, then there shall be N+1 texture coordinates in the *X3DTextureCoordinateNode*.
- g. If the *texCoordIndex* field is empty, then the *coordIndex* array is used to choose texture coordinates from the *X3DTextureCoordinateNode* node. If the greatest index in the *coordIndex* field is N, then there shall be N+1 texture coordinates in the *X3DTextureCoordinateNode* node.

If the *texCoord* field is `NULL`, a default texture coordinate mapping is calculated using the local coordinate system bounding box of the shape. The longest dimension of the bounding box defines the S coordinates, and the next longest defines the T coordinates. If two or all three dimensions of the bounding box are equal, ties shall be broken by choosing the X, Y, or Z dimension in that order of preference. The value of the S coordinate ranges from 0 to 1, from one end of the bounding box to the other. The T coordinate ranges between 0 and the ratio of the second greatest dimension of the bounding box to the greatest dimension. [Figure 13.6](#) illustrates the default texture coordinates for a simple box shaped IndexedFaceSet with an X dimension twice as large as the Z dimension and four times as large as the Y dimension. [Figure 13.7](#) illustrates the original texture image used on the IndexedFaceSet used in [Figure 13.6](#).

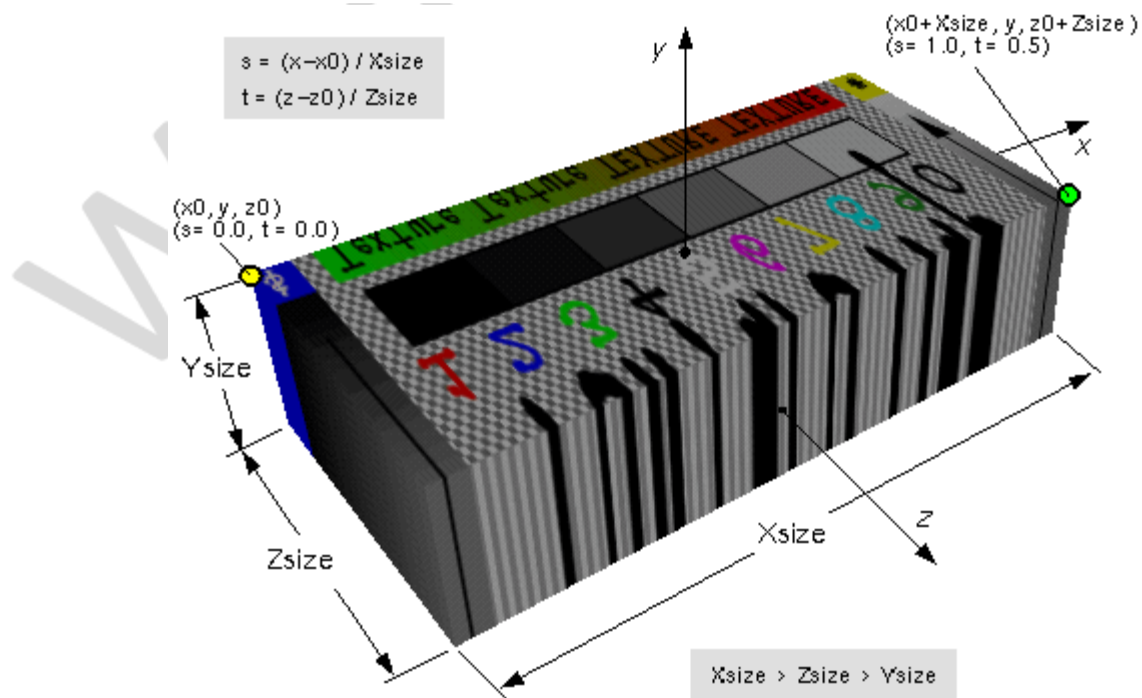


Figure 13.6 — IndexedFaceSet texture default mapping

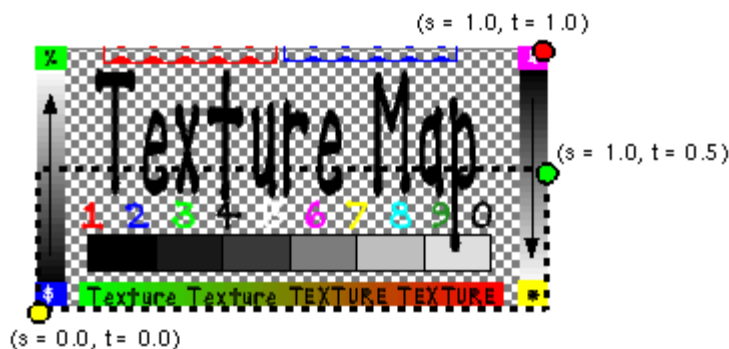


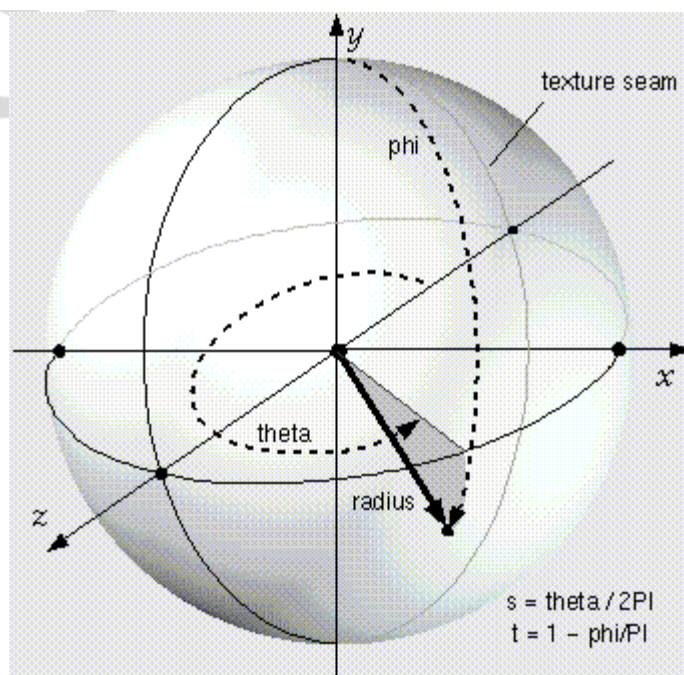
Figure 13.7 — ImageTexture for IndexedFaceSet in Figure 13.6

[11.2.3 Common geometry fields](#), provides a description of the ccw, solid, convex, and creaseAngle fields.

### 13.3.7 Sphere

```
Sphere : X3DGeometryNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFFloat [] radius 1 (0,∞)
  SFBool [] solid TRUE
}
```

The Sphere node specifies a sphere centred at (0, 0, 0) in the local coordinate system. The *radius* field specifies the radius of the sphere and shall be greater than zero. [Figure 13.8](#) depicts the fields of the Sphere node.



**Figure 13.8 — Sphere node**

When a texture is applied to a sphere, the texture covers the entire surface, wrapping counterclockwise from the back of the sphere (*i.e.*, longitudinal arc intersecting the  $-Z$ -axis) when viewed from the top of the sphere. The texture has a seam at the back where the  $X=0$  plane intersects the sphere and  $Z$  values are negative.

[TextureTransform](#) affects the texture coordinates of the Sphere (see [18.4.8 TextureTransform](#)).

The *solid* field determines whether the sphere is visible when viewed from the inside. [11.2.3 Common geometry fields](#) provides a complete description of the *solid* field.

This geometry node is fundamentally a mathematical representation. Displayed geometry shall have sufficient rendering quality that surface and silhouette edges appear smooth, including when textures are applied.

## 13.4 Geometry3D component support levels.

The Geometry3D component provides three levels of support as specified in [Table 13.2](#). Level 1 provides the basic indexed geometry types with limited support for some fields, as well as the geometric primitives and the [Shape](#) node. Level 2 adds support for the [IndexedFaceSet](#) node. Level 3 adds support for the [ElevationGrid](#) node to enable lightweight terrain and data visualization and supports all fields in all nodes supported at Level 3. Level 4 adds support for the [Extrusion](#) node.

**Table 13.2 — Geometry3D component support levels**

Level	Prerequisites	Nodes/Features	Support
1	Core 1 Grouping 1 Rendering 1		

	Shape 1		
		Box	All fields fully supported.
		Cone	All fields fully supported.
		Cylinder	All fields fully supported.
		Sphere	All fields fully supported.
<b>2</b>	Core 1 Grouping 1 Rendering 1 Shape 1		
		All Level 1 geometry nodes	All fields as supported in Level 1.
		IndexedFaceSet	<p><i>ccw</i> optionally supported. <i>set_colorIndex</i> optionally supported. <i>set_normalIndex</i> optionally supported. <i>normal</i> optionally supported. Only convex indexed face sets supported. Hence, <i>convex</i> optionally supported. For <i>creaseAngle</i>, only 0 and <math>\pi</math> radians supported (or the equivalent if a different angle base unit has been specified). <i>normalIndex</i> optionally supported.</p> <p>Face list shall be well-defined as follows:</p> <ol style="list-style-type: none"> <li>1. Each face is terminated with -1, including the last face in the array.</li> <li>2. Each face contains at least three non-coincident vertices.</li> <li>3. A given <i>coordIndex</i> is not repeated in a face.</li> <li>4. The vertices of a face shall define a planar polygon.</li> <li>5. The vertices of a face shall not define a self-intersecting polygon.</li> </ol>
<b>3</b>	Core 1 Grouping 1 Rendering 1 Shape 1		

		All Level 2 geometry nodes	All fields as supported in Level 2.
		ElevationGrid	<i>ccw</i> optionally supported.
<b>4</b>	Core 1 Grouping 1 Rendering 1 Shape 1		
		All Level 3 geometry nodes	All fields fully supported.
		Extrusion	All fields fully supported.



Draft





## Extensible 3D (X3D) Part 1: Architecture and base components

### 34 Cube map environmental texturing component



#### 34.1. Introduction

##### 34.1.1 Name

The name of this component is "CubeMapTexturing". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 COMPONENT statement](#)).

##### 34.1.2 Overview

This clause describes the cube map environmental texturing component of this part of ISO/IEC 19775. This includes how additional texturing effects are defined to produce environmental effects such as reflections from objects. [Table 34.1](#) provides lists the major topics in this clause.

**Table 34.1 — Topics**

- [34.1 Introduction](#)
  - [34.1.1 Name](#)
  - [34.1.2 Overview](#)
- [34.2 Concepts](#)
  - [34.2.1 Overview](#)
  - [34.2.2 Texture map formats](#)
  - [34.2.3 Texture map image formats](#)
  - [34.2.4 Texture coordinates](#)
  - [34.2.5 Texture orientation](#)
- [34.3 Abstract types](#)
  - [34.3.1 X3DEnvironmentTextureNode](#)
- [34.4. Node reference](#)
  - [34.4.1 ComposedCubeMapTexture](#)
  - [34.4.2 GeneratedCubeMapTexture](#)
  - [34.4.3 ImageCubeMapTexture](#)
- [34.5 Support levels](#)



- [Figure 34.1 — Mapping texture sides to the texture coordinate axes](#)
- [Table 34.1 — Topics](#)
- [Table 34.2 — Environment Texturing component support levels](#)

## 34.2 Concepts

### 34.2.1 Overview

Cube map environmental texturing provides cubic environmental texture mapping capabilities within X3D. Cubic environment maps support reflection and specular highlighting in a simple way, often in combination with automatic texture coordinate generation (see [18.2.3 Texture coordinates](#)). This component may be combined with the multitexture abilities of the Texturing component (see [18.2.4 Multitexturing](#)) to provide advanced visual effects.

Cubic environment maps ignore most of the normal texture settings (*e.g.*, there are no repeat fields) but they can be mipmapped. The sources can be drawn from any 2D texture source whether dynamically generated or provided from somewhere else as images or pixel arrays.

### 34.2.2 Texture Map Formats

Cubic environment mapping nodes defined as part of this component use a collection of 2D texture maps to define each side of the cube. These may contain from one to four component colour values. The interpretation of the image shall follow the description in [18.2.1 Texture map formats](#).

All source images shall be square and provide source data in powers of two numbers of pixels. Source images in a cubic environment map shall have identical sizes. Providing differently sized images or rectangular images shall be an error.

EXAMPLE It is not valid to define the front image as a 64×64 image and the left side image as 128×128 pixels.

### 34.2.3 Texture Map Image Formats

Texture nodes that require support for 2D images file formats shall follow the description defined in [18.2.2 Texture map image formats](#).

### 34.2.4 Texture Coordinates

For each texture, the three-dimensional texture coordinates (s,t,r) are treated as a direction vector from the local origin. Each texel drawn onto the geometry is treated as the texel in the environment map that is "seen" from this direction vector.

Texture coordinates for using cubic environment mapped textures are usually dynamically generated as this is far easier to handle for the content developer than providing explicit texture coordinates. It is recommended that an implementation shall also support a minimum of Level 2 Texturing component capabilities (see [18.5 Support](#)

[levels](#)) in addition to this component. Typically, the `CAMERASPACE` `NORMAL` OF `CAMERASPACE` `REFLECTIONVECTOR` modes are used.

To specify explicit texture coordinates, the [TextureCoordinate3D](#) node (see [33 Texture3D component](#)) shall be used.

## 34.2.5 Texture Orientation

Cubic environment maps define a single texture as consisting of six separate images, one for each side of a cube. This component defines the six sides as front, back, left, right, top and bottom. These sides shall be oriented as shown in [Figure 34.1](#).

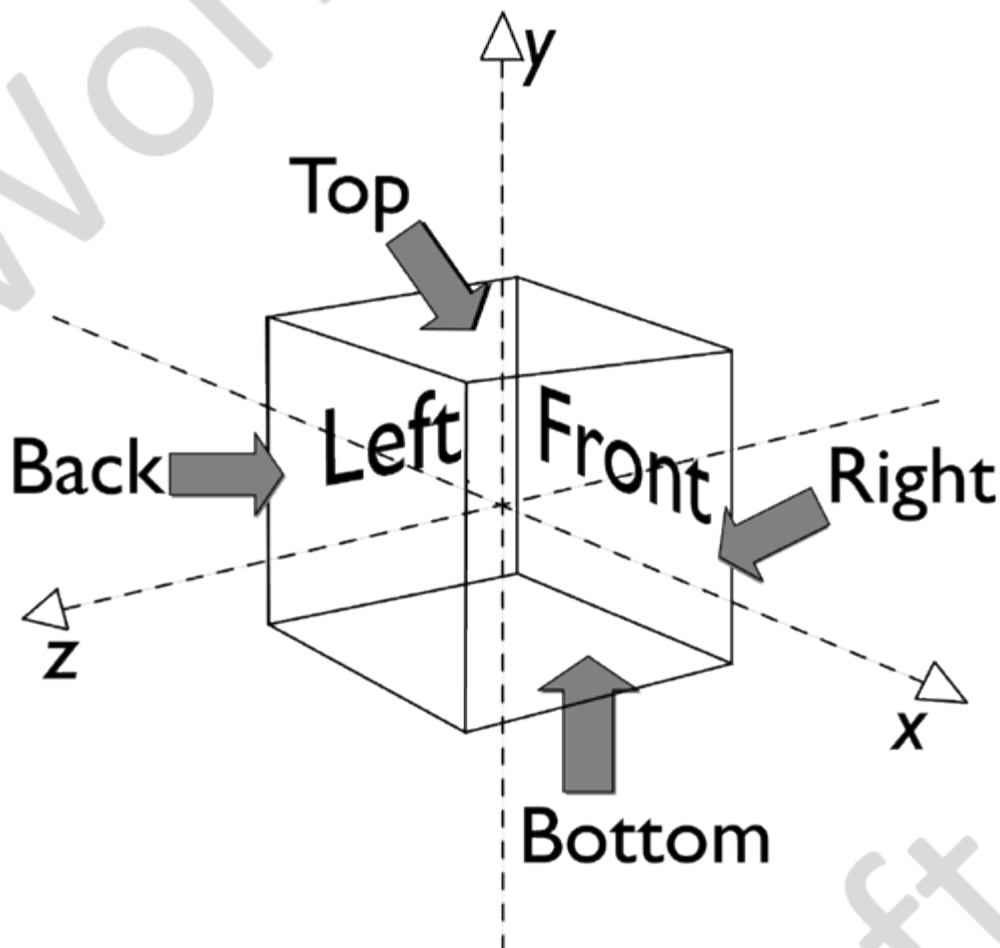


Figure 34.1 — Mapping texture sides to the texture coordinate axes

## 34.3 Abstract Types

### 34.3.1 *X3DEnvironmentTextureNode*

```
X3DEnvironmentTextureNode : X3DSingleTextureNode {
  X3DEnvironmentTextureNode : X3DTextureNode {
    SFNode [in,out] metadata NULL [X3DMetadataObject]
  }
}
```

This abstract node type is the base type for all node types that specify cubic environment map sources for texture images.

## 34.4 Node reference

### 34.4.1 ComposedCubeMapTexture

```
ComposedCubeMapTexture : X3DEnvironmentTextureNode {
  SFNode [in,out] back    NULL [X3DTexture2DNode]
  SFNode [in,out] bottom  NULL [X3DTexture2DNode]
  SFNode [in,out] front   NULL [X3DTexture2DNode]
  SFNode [in,out] left    NULL [X3DTexture2DNode]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFNode [in,out] right   NULL [X3DTexture2DNode]
  SFNode [in,out] top     NULL [X3DTexture2DNode]
}
```

The `ComposedCubeMapTexture` node defines a cubic environment map source as an explicit set of images drawn from individual 2D texture nodes.

See [34.2 Concepts](#) for a general description of cube map environmental texture maps.

See [18 Texturing component](#) for a general description of the `X3DTexture2DNode` abstract type and interpretation of rendering for 2D images. When used as a source for cubic environment maps, the fields *repeatS* and *repeatT* fields shall be ignored.

### 34.4.2 GeneratedCubeMapTexture

```
GeneratedCubeMapTexture : X3DEnvironmentTextureNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFString [in,out] update  "NONE" ["NONE"|"NEXT_FRAME_ONLY"|"ALWAYS"]
  SFInt32 [] size          128 (0,∞)
  SFNode [] textureProperties NULL [TextureProperties]
}
```

The `GeneratedCubeMapTexture` node defines a cubic environment map that sources its data from internally generated images, rendered from a virtual situated perspective in the scene.

The viewpoint of the generated texture is based on the location and orientation of the associated geometry in world space.

**NOTE** An object trying to render itself in the scene graph can cause infinite loops in the renderer implementation and is thus not permitted.

The field of view shall be  $\pi/2$  radians (or the equivalent angle base units) with an aspect ratio of 1:1.

The *size* field indicates the resolution of the generated images in number of pixels per side.

The *update* field can be used to request a regeneration of the texture. Setting this field to "ALWAYS" will cause the texture to be rendered every frame. A value of "NONE" will stop rendering so that no further updates are performed even if the contained scene graph changes. When the value is set to "NEXT\_FRAME\_ONLY", it is an instruction to render the texture at the end of this frame, and then not render it again. In this case, the update frame indicator is set to this frame; at the start of the next frame, the update value shall be automatically set back to "NONE" to indicate that the rendering has already taken place. Since this is a change of value for the *update* field, an output event is automatically generated.

### 34.4.3 ImageCubeMapTexture

```
ImageCubeMapTexture : X3DEnvironmentTextureNode, X3DUrlObject {
  SFString [in,out] description ""
  SFBool [in,out] load TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTime [in,out] refresh 0.0 [0,∞)
  MFString [in,out] url [] [URI]
  SFNode [] textureProperties NULL [TextureProperties]
}
```

The ImageCubeMapTexture node defines a cubic environment map source as a single file format that contains multiple images, one for each side.

The texture is read from the URL specified by the *url* field. When the *url* field contains no values, texturing is disabled. The *url* field is defined in [9.2.1 URLs](#). Browsers are not required to support any specific cube map environment texture format. It is recommended that browsers support the Microsoft DDS cube map environment texture file format (see [\[DDS\]](#)).

See [18.2 Concepts](#) for a general description of texture maps.

## 34.5 Support levels

The Cube map environmental texturing component defines three levels of support as specified in [Table 34.2](#).

**Table 34.2 — Cube map environmental texturing component support levels**

Level	Prerequisites	Nodes/Features	Support
1	Core 1 Grouping 1 Shape 1 Rendering 1 Texturing 1		
		<i>X3DEnvironmentTextureNode</i>	n/a
2	Core 1 Grouping 1 Shape 1 Rendering 1 Texturing 1	ComposedCubeMapTexture	All fields fully supported.
		ImageCubeMapTexture	All fields fully supported.
3	Core 1 Grouping 1 Shape 1 Rendering 1 Texturing 1		

		GeneratedCubeMapTexture	All fields fully supported.
--	--	-------------------------	-----------------------------





# Extensible 3D (X3D) Part 1: Architecture and base components

## Annex Z (normative)

### Version content



#### Z.1 General

This annex specifies the content supported by the specified versions of X3D. Conformance requirements are stated in [6 Conformance](#).

#### Z.2 Topics

[Table Z.1](#) provides links to the major topics in this annex.

**Table Z.1 — Topics**

- [Z.1 General](#)
- [Z.2 Topics](#)
- [Z.3 Version content](#)
- [Table Z.1 — Topics](#)
- [Table Z.2 — Version content \(nodes\)](#)
- [Table Z.3 — Version content \(statements\)](#)

#### Z.3 Version content

[Table Z.2](#) lists each node specified by this part of ISO/IEC 19775. For each node, the fields supported by each version are identified listed in the order specified by the node signature. Nodes will appear in multiple rows if fields have been added in subsequent versions.

**Table Z.2 — Version content (nodes)**

Index:	<a href="#">A</a>	<a href="#">B</a>	<a href="#">C</a>	<a href="#">D</a>	<a href="#">E</a>	<a href="#">F</a>	<a href="#">G</a>	<a href="#">H</a>	<a href="#">I</a>	<a href="#">K</a>	<a href="#">L</a>	<a href="#">M</a>	<a href="#">N</a>	<a href="#">O</a>	<a href="#">P</a>	<a href="#">Q</a>	<a href="#">R</a>	<a href="#">S</a>	<a href="#">T</a>	<a href="#">U</a>	<a href="#">V</a>	<a href="#">W</a>	<a href="#">X</a>				
Node	Fields																						3.0	3.1	3.2	3.3	4.0
	addChildren																						X	X	X	X	X

<a href="#">Anchor</a>	removeChildren	X	X	X	X	X
	children	X	X	X	X	X
	description	X	X	X	X	X
	metadata	X	X	X	X	X
	parameter	X	X	X	X	X
	url	X	X	X	X	X
	bboxCenter	X	X	X	X	X
	bboxSize	X	X	X	X	X
<a href="#">Appearance</a>	backMaterial					X
	fillProperties	X	X	X	X	X
	lineProperties	X	X	X	X	X
	material	X	X	X	X	X
	metadata	X	X	X	X	X
	shaders		X	X	X	X
	texture	X	X	X	X	X
	textureTransform	X	X	X	X	X
<a href="#">Arc2D</a>	metadata	X	X	X	X	X
	endAngle	X	X	X	X	X
	radius	X	X	X	X	X
	startAngle	X	X	X	X	X
<a href="#">ArcClose2D</a>	metadata	X	X	X	X	X
	closureType	X	X	X	X	X
	endAngle	X	X	X	X	X
	radius	X	X	X	X	X
	solid	X	X	X	X	X
	startAngle	X	X	X	X	X
<a href="#">AudioClip</a>	description	X	X	X	X	X
	loop	X	X	X	X	X
	metadata	X	X	X	X	X
	pauseTime	X	X	X	X	X
	pitch	X	X	X	X	X
	resumeTime	X	X	X	X	X
	startTime	X	X	X	X	X
	stopTime	X	X	X	X	X
	url	X	X	X	X	X
	duration_changed	X	X	X	X	X
	elapsedTime	X	X	X	X	X
	isActive	X	X	X	X	X
	isPaused	X	X	X	X	X
	set_bind	X	X	X	X	X
	groundAngle	X	X	X	X	X



<a href="#">Background</a>	groundColor	X	X	X	X	X
	backUrl	X	X	X	X	X
	bottomUrl	X	X	X	X	X
	frontUrl	X	X	X	X	X
	leftUrl	X	X	X	X	X
	metadata	X	X	X	X	X
	rightUrl	X	X	X	X	X
	topUrl	X	X	X	X	X
	skyAngle	X	X	X	X	X
	skyColor	X	X	X	X	X
	transparency			X	X	X
	bindTime	X	X	X	X	X
	isBound	X	X	X	X	X
<a href="#">BallJoint</a>	anchorPoint			X	X	X
	body1			X	X	X
	body2			X	X	X
	metadata			X	X	X
	mustOutput			X	X	X
	body1AnchorPoint			X	X	X
	body2AnchorPoint			X	X	X
<a href="#">Billboard</a>	addChildren	X	X	X	X	X
	removeChildren	X	X	X	X	X
	axisOfRotation	X	X	X	X	X
	children	X	X	X	X	X
	metadata	X	X	X	X	X
	bboxCenter	X	X	X	X	X
	bboxSize	X	X	X	X	X
<a href="#">BlendedVolumeStyle</a>	enabled				X	X
	metadata				X	X
	renderStyle				X	X
	voxels				X	X
	weightConstant1				X	X
	weightConstant2				X	X
	weightFunction1				X	X
	weightFunction2				X	X
	weightTransferFucntion1				X	X
	weightTransferFunction2				X	X
	bboxCenter				X	X
	bboxSize				X	X
	set_boolean	X	X	X	X	X
	metadata	X	X	X	X	X

<a href="#">BooleanFilter</a>	inputFalse	X	X	X	X	X
	inputNegate	X	X	X	X	X
	inputTrue	X	X	X	X	X
<a href="#">BooleanSequencer</a>	next	X	X	X	X	X
	previous	X	X	X	X	X
	set_fraction	X	X	X	X	X
	key	X	X	X	X	X
	keyValue	X	X	X	X	X
	metadata	X	X	X	X	X
	value_changed	X	X	X	X	X
<a href="#">BooleanToggle</a>	set_boolean	X	X	X	X	X
	metadata	X	X	X	X	X
	toggle	X	X	X	X	X
<a href="#">BooleanTrigger</a>	set_triggerTime	X	X	X	X	X
	metadata	X	X	X	X	X
	triggerTrue	X	X	X	X	X
<a href="#">BoundaryEnhancementVolumeStyle</a>	boundaryOpacity				X	X
	enabled				X	X
	metadata				X	X
	opacityFactor				X	X
	retainedOpacity				X	X
	transferFunction				X	X
<a href="#">BoundedPhysicsModel</a>	enabled			X	X	X
	geometry			X	X	X
	metadata			X	X	X
<a href="#">Box</a>	metadata	X	X	X	X	X
	size	X	X	X	X	X
	solid	X	X	X	X	X
<a href="#">CADAssembly</a>	addChildren		X	X	X	X
	removeChildren		X	X	X	X
	children		X	X	X	X
	metadata		X	X	X	X
	name		X	X	X	X
	bboxCenter		X	X	X	X
	bboxSize		X	X	X	X
<a href="#">CADFace</a>	metadata		X	X	X	X
	name		X	X	X	X
	shape		X	X	X	X
	bboxCenter		X	X	X	X
	bboxSize		X	X	X	X
	addChildren		X	X	X	X

<a href="#">CADLayer</a>	removeChildren		X	X	X	X
	children		X	X	X	X
	metadata		X	X	X	X
	name		X	X	X	X
	visible		X	X	X	X
	bboxCenter		X	X	X	X
	bboxSize		X	X	X	X
<a href="#">CADPart</a>	addChildren		X	X	X	X
	removeChildren		X	X	X	X
	center		X	X	X	X
	children		X	X	X	X
	metadata		X	X	X	X
	name		X	X	X	X
	rotation		X	X	X	X
	scale		X	X	X	X
	scaleOrientation		X	X	X	X
	translation		X	X	X	X
	bboxCenter		X	X	X	X
	bboxSize		X	X	X	X
<a href="#">CartoonVolumeStyle</a>	colorSteps				X	X
	enabled				X	X
	metadata				X	X
	orthogonalColor				X	X
	parallelColor				X	X
	surfaceNormals				X	X
<a href="#">Circle2D</a>	metadata	X	X	X	X	X
	radius	X	X	X	X	X
<a href="#">ClipPlane</a>	enabled			X	X	X
	metadata			X	X	X
	plane			X	X	X
<a href="#">CollidableOffset</a>	enabled			X	X	X
	metadata			X	X	X
	rotation			X	X	X
	translation			X	X	X
	bboxCenter			X	X	X
	bboxSize			X	X	X
	collidable			X	X	X
<a href="#">CollidableShape</a>	enabled			X	X	X
	metadata			X	X	X
	rotation			X	X	X
	translation			X	X	X

	bboxCenter			X	X	X
	bboxSize			X	X	X
	shape			X	X	X
<a href="#">Collision</a>	addChildren	X	X	X	X	X
	removeChildren	X	X	X	X	X
	enabled	X	X	X	X	X
	children	X	X	X	X	X
	metadata	X	X	X	X	X
	collideTime	X	X	X	X	X
	isActive	X	X	X	X	X
	bboxCenter	X	X	X	X	X
	bboxSize	X	X	X	X	X
	proxy	X	X	X	X	X
<a href="#">CollisionCollection</a>	appliedParameters			X	X	X
	bounce			X	X	X
	collidables			X	X	X
	enabled			X	X	X
	frictionCoefficients			X	X	X
	metadata			X	X	X
	minBounceSpeed			X	X	X
	slipFactors			X	X	X
	softnessConstantForceMix			X	X	X
	softnessErrorCorrection			X	X	X
	surfaceSpeed			X	X	X
<a href="#">CollisionSensor</a>	collidables			X	X	X
	enabled			X	X	X
	metadata			X	X	X
	intersections			X	X	X
	contacts			X	X	X
	isActive			X	X	X
<a href="#">CollisionSpace</a>	collidables			X	X	X
	enabled			X	X	X
	metadata			X	X	X
	useGeometry			X	X	X
	bboxCenter			X	X	X
	bboxSize			X	X	X
<a href="#">Color</a>	color	X	X	X	X	X
	metadata	X	X	X	X	X
	set_destination				X	X
	set_value				X	X
	metadata				X	X

<a href="#">ColorChaser</a>	isActive				X	X
	value_changed				X	X
	duration				X	X
	initialDestination				X	X
	initialValue				X	X
<a href="#">ColorDamper</a>	set_destination			X	X	X
	set_value			X	X	X
	metadata			X	X	X
	tau			X	X	X
	tolerance			X	X	X
	isActive			X	X	X
	value_changed			X	X	X
	initialDestination			X	X	X
	initialValue			X	X	X
	order			X	X	X
	<a href="#">ColorInterpolator</a>	set_fraction	X	X	X	X
key		X	X	X	X	X
keyValue		X	X	X	X	X
metadata		X	X	X	X	X
value_changed		X	X	X	X	X
<a href="#">ColorRGBA</a>	color	X	X	X	X	X
	metadata	X	X	X	X	X
<a href="#">ComposedCubeMapTexture</a>	back		X	X	X	X
	bottom		X	X	X	X
	front		X	X	X	X
	left		X	X	X	X
	metadata		X	X	X	X
	right		X	X	X	X
	top		X	X	X	X
<a href="#">ComposedShader</a>	activate		X	X	X	X
	metadata		X	X	X	X
	parts		X	X	X	X
	isSelected		X	X	X	X
	isValid		X	X	X	X
	language		X	X	X	X
<a href="#">ComposedTexture3D</a>	metadata		X	X	X	X
	repeatS		X	X	X	X
	repeatT		X	X	X	X
	repeatR		X	X	X	X
	texture		X	X	X	X
	textureProperties				X	X

<a href="#">ComposedVolumeStyle</a>	enabled				X	X
	metadata				X	X
	renderStyle				X	X
<a href="#">Cone</a>	metadata	X	X	X	X	X
	bottom	X	X	X	X	X
	bottomRadius	X	X	X	X	X
	height	X	X	X	X	X
	side	X	X	X	X	X
	solid	X	X	X	X	X
<a href="#">ConeEmitter</a>	angle			X	X	X
	direction			X	X	X
	metadata			X	X	X
	position			X	X	X
	speed			X	X	X
	variation			X	X	X
	mass			X	X	X
	surfaceArea			X	X	X
<a href="#">Contact</a>	appliedParameters			X	X	X
	body1			X	X	X
	body2			X	X	X
	bounce			X	X	X
	enabled			X	X	X
	depth			X	X	X
	frictionCoefficients			X	X	X
	frictionDirection			X	X	X
	geometry1			X	X	X
	geometry2			X	X	X
	metadata			X	X	X
	minBounceSpeed			X	X	X
	position			X	X	X
	slipCoefficients			X	X	X
	softnessConstantForceMix			X	X	X
	softnessErrorCorrection			X	X	X
	surfaceSpeed			X	X	X
<a href="#">Contour2D</a>	addChildren	X	X	X	X	X
	removeChildren	X	X	X	X	X
	children	X	X	X	X	X
	metadata	X	X	X	X	X
<a href="#">ContourPolyline2D</a>	metadata	X	X	X	X	X
	controlPoint	X	X	X	X	X
<a href="#">Coordinate</a>	metadata	X	X	X	X	X

	point	X	X	X	X	X
<a href="#">CoordinateChaser</a>	set_destination				X	X
	set_value				X	X
	metadata				X	X
	isActive				X	X
	value_changed				X	X
	duration				X	X
	initialDestination				X	X
	initialValue				X	X
<a href="#">CoordinateDamper</a>	set_destination			X	X	X
	set_value			X	X	X
	metadata			X	X	X
	tau			X	X	X
	tolerance			X	X	X
	isActive			X	X	X
	value_changed			X	X	X
	initialDestination			X	X	X
	initialValue			X	X	X
	order			X	X	X
<a href="#">CoordinateDouble</a>	metadata	X	X	X	X	X
	point	X	X	X	X	X
<a href="#">CoordinateInterpolator</a>	set_fraction	X	X	X	X	X
	key	X	X	X	X	X
	keyValue	X	X	X	X	X
	metadata	X	X	X	X	X
	value_changed	X	X	X	X	X
<a href="#">CoordinateInterpolator2D</a>	set_fraction	X	X	X	X	X
	key	X	X	X	X	X
	keyValue	X	X	X	X	X
	metadata	X	X	X	X	X
	value_changed	X	X	X	X	X
<a href="#">Cylinder</a>	metadata	X	X	X	X	X
	bottom	X	X	X	X	X
	height	X	X	X	X	X
	radius	X	X	X	X	X
	side	X	X	X	X	X
	solid	X	X	X	X	X
	top	X	X	X	X	X
	autoOffset	X	X	X	X	X
	description	X	X	X	X	X
	diskAngle	X	X	X	X	X

<a href="#">CylinderSensor</a>	enabled	X	X	X	X	X
	maxAngle	X	X	X	X	X
	metadata	X	X	X	X	X
	minAngle	X	X	X	X	X
	offset	X	X	X	X	X
	isActive	X	X	X	X	X
	isOver	X	X	X	X	X
	rotation_changed	X	X	X	X	X
	trackPoint_changed	X	X	X	X	X
<a href="#">DirectionalLight</a>	ambientIntensity	X	X	X	X	X
	color	X	X	X	X	X
	direction	X	X	X	X	X
	global		X	X	X	X
	intensity	X	X	X	X	X
	metadata	X	X	X	X	X
	on	X	X	X	X	X
<a href="#">DISEntityManager</a>	address			X	X	X
	applicationID			X	X	X
	mapping			X	X	X
	metadata			X	X	X
	port			X	X	X
	siteID			X	X	X
	addedEntities			X	X	X
	removedEntities			X	X	X
<a href="#">DISEntityTypeMapping</a>	metadata			X	X	X
	url			X	X	X
	category			X	X	X
	country			X	X	X
	domain			X	X	X
	extra			X	X	X
	kind			X	X	X
	specific			X	X	X
	subcategory			X	X	X
<a href="#">Disk2D</a>	metadata	X	X	X	X	X
	innerRadius	X	X	X	X	X
	outerRadius	X	X	X	X	X
	solid	X	X	X	X	X
	anchorPoint			X	X	X
	axis1			X	X	X
	axis2			X	X	X
	body1			X	X	X



<a href="#">DoubleAxisHingeJoint</a>	body2			X	X	X
	desiredAngularVelocity1			X	X	X
	desiredAngularVelocity2			X	X	X
	maxAngle1			X	X	X
	maxTorque1			X	X	X
	maxTorque2			X	X	X
	metadata			X	X	X
	minAngle1			X	X	X
	mustOutput			X	X	X
	stopBounce1			X	X	X
	stopConstantForceMix1			X	X	X
	stopErrorCorrection1			X	X	X
	suspensionErrorCorrection			X	X	X
	suspensionForce			X	X	X
	body1AnchorPoint			X	X	X
	body2AnchorPoint			X	X	X
	body1Axis			X	X	X
	body2Axis			X	X	X
	hinge1Angle			X	X	X
	hinge1AngleRate			X	X	X
hinge2Angle			X	X	X	
hinge2AngleRate			X	X	X	
<a href="#">EaseInEaseOut</a>	set_fraction			X	X	X
	easeInEaseOut			X	X	X
	key			X	X	X
	metadata			X	X	X
	modifiedFraction_changed			X	X	X
<a href="#">EdgeEnhancementVolumeStyle</a>	edgeColor				X	X
	enabled				X	X
	gradientThreshold				X	X
	metadataColor				X	X
	surfaceNormals				X	X
<a href="#">ElevationGrid</a>	set_height	X	X	X	X	X
	attrib		X	X	X	X
	color	X	X	X	X	X
	fogCoord		X	X	X	X
	metadata	X	X	X	X	X
	normal	X	X	X	X	X
	texCoord	X	X	X	X	X
	ccw	X	X	X	X	X
	colorPerVertex	X	X	X	X	X

creaseAngle	X	X	X	X	X
height	X	X	X	X	X
normalPerVertex	X	X	X	X	X
solid	X	X	X	X	X
xDimension	X	X	X	X	X
xSpacing	X	X	X	X	X
zDimension	X	X	X	X	X
zSpacing	X	X	X	X	X
addChildren	X	X	X	X	X
removeChildren	X	X	X	X	X
set_articulationParameterValue0	X	X	X	X	X
set_articulationParameterValue1	X	X	X	X	X
set_articulationParameterValue2	X	X	X	X	X
set_articulationParameterValue3	X	X	X	X	X
set_articulationParameterValue4	X	X	X	X	X
set_articulationParameterValue5	X	X	X	X	X
set_articulationParameterValue6	X	X	X	X	X
set_articulationParameterValue7	X	X	X	X	X
address	X	X	X	X	X
applicationID	X	X	X	X	X
articulationParameterCount	X	X	X	X	X
articulationParameterDesignatorArray	X	X	X	X	X
articulationParameterChangeIndicatorArray	X	X	X	X	X
articulationParameterIdPartAttachedToArray	X	X	X	X	X
articulationParameterTypeArray	X	X	X	X	X
articulationParameterArray	X	X	X	X	X
center	X	X	X	X	X
children	X	X	X	X	X
collisionType	X	X	X	X	X
deadReckoning	X	X	X	X	X
detonationLocation	X	X	X	X	X
detonationRelativeLocation	X	X	X	X	X
detonationResult	X	X	X	X	X
enabled		X	X	X	X
entityCategory	X	X	X	X	X
entityCountry	X	X	X	X	X
entityDomain	X	X	X	X	X
entityExtra	X	X	X	X	X
entityId	X	X	X	X	X
entityKind	X	X	X	X	X
entitySpecific	X	X	X	X	X

[EspduTransform](#)

entitySubCategory	X	X	X	X	X
eventApplicationID	X	X	X	X	X
eventEntityID	X	X	X	X	X
eventNumber	X	X	X	X	X
eventSiteID	X	X	X	X	X
fired1	X	X	X	X	X
fired2	X	X	X	X	X
fireMissionIndex	X	X	X	X	X
firingRange	X	X	X	X	X
firingRate	X	X	X	X	X
forceID	X	X	X	X	X
fuse	X	X	X	X	X
linearVelocity	X	X	X	X	X
linearAcceleration	X	X	X	X	X
marking	X	X	X	X	X
metadata	X	X	X	X	X
multicastRelayHost	X	X	X	X	X
multicastRelayPort	X	X	X	X	X
munitionApplicationID	X	X	X	X	X
munitionEndPoint	X	X	X	X	X
munitionEntityID	X	X	X	X	X
munitionQuantity	X	X	X	X	X
munitionSiteID	X	X	X	X	X
munitionStartPoint	X	X	X	X	X
networkMode	X	X	X	X	X
port	X	X	X	X	X
readInterval	X	X	X	X	X
rotation	X	X	X	X	X
scale	X	X	X	X	X
scaleOrientation	X	X	X	X	X
siteID	X	X	X	X	X
translation	X	X	X	X	X
warhead	X	X	X	X	X
writeInterval	X	X	X	X	X
articulationParameterValue0_changed	X	X	X	X	X
articulationParameterValue1_changed	X	X	X	X	X
articulationParameterValue2_changed	X	X	X	X	X
articulationParameterValue3_changed	X	X	X	X	X
articulationParameterValue4_changed	X	X	X	X	X
articulationParameterValue5_changed	X	X	X	X	X
articulationParameterValue6_changed	X	X	X	X	X
articulationParameterValue7_changed	X	X	X	X	X

	collideTime	X	X	X	X	X
	detonateTime	X	X	X	X	X
	firedTime	X	X	X	X	X
	isActive	X	X	X	X	X
	isCollided	X	X	X	X	X
	isDetonated	X	X	X	X	X
	isNetworkReader	X	X	X	X	X
	isRtpHeaderHeard	X	X	X	X	X
	isStandAlone	X	X	X	X	X
	timestamp	X	X	X	X	X
	bboxCenter	X	X	X	X	X
	bboxSize	X	X	X	X	X
	rtpHeaderExpected	X	X	X	X	X
<a href="#">ExplosionEmitter</a>	metadata			X	X	X
	position			X	X	X
	speed			X	X	X
	variation			X	X	X
	mass			X	X	X
	surfaceArea			X	X	X
<a href="#">Extrusion</a>	set_crossSection	X	X	X	X	X
	set_orientation	X	X	X	X	X
	set_scale	X	X	X	X	X
	set_spine	X	X	X	X	X
	metadata	X	X	X	X	X
	beginCap	X	X	X	X	X
	ccw	X	X	X	X	X
	convex	X	X	X	X	X
	creaseAngle	X	X	X	X	X
	crossSection	X	X	X	X	X
	endCap	X	X	X	X	X
	orientation	X	X	X	X	X
	scale	X	X	X	X	X
	solid	X	X	X	X	X
	spine	X	X	X	X	X
<a href="#">FillProperties</a>	filled	X	X	X	X	X
	hatchColor	X	X	X	X	X
	hatched	X	X	X	X	X
	hatchStyle	X	X	X	X	X
	metadata	X	X	X	X	X
	metadata		X	X	X	X
	value		X	X	X	X

<a href="#">FloatVertexAttribute</a>	name		X	X	X	X
	numComponents		X	X	X	X
<a href="#">Fog</a>	set_bind	X	X	X	X	X
	color	X	X	X	X	X
	fogType	X	X	X	X	X
	metadata	X	X	X	X	X
	visibilityRange	X	X	X	X	X
	bindTime	X	X	X	X	X
	isBound	X	X	X	X	X
<a href="#">FogCoordinate</a>	depth		X	X	X	X
	metadata		X	X	X	X
<a href="#">FontStyle</a>	metadata	X	X	X	X	X
	family	X	X	X	X	X
	horizontal	X	X	X	X	X
	justify	X	X	X	X	X
	language	X	X	X	X	X
	leftToRight	X	X	X	X	X
	size	X	X	X	X	X
	spacing	X	X	X	X	X
	style	X	X	X	X	X
	topToBottom	X	X	X	X	X
<a href="#">ForcePhysicsModel</a>	enabled			X	X	X
	force			X	X	X
	metadata			X	X	X
<a href="#">GeneratedCubeMapTexture</a>	metadata		X	X	X	X
	update		X	X	X	X
	size		X	X	X	X
<a href="#">GeoCoordinate</a>	metadata	X	X	X	X	X
	point	X	X	X	X	X
	geoOrigin	X	X	X	X	X
	geoSystem	X	X	X	X	X
	set_height	X	X	X	X	X
	color	X	X	X	X	X
	metadata	X	X	X	X	X
	normal	X	X	X	X	X
	texCoord	X	X	X	X	X
	yScale	X	X	X	X	X
	ccw	X	X	X	X	X
	colorPerVertex	X	X	X	X	X
	creaseAngle	X	X	X	X	X

<a href="#">GeoElevationGrid</a>	geoGridOrigin	X	X	X	X	X
	geoOrigin	X	X	X	X	X
	geoSystem	X	X	X	X	X
	height	X	X	X	X	X
	normalPerVertex	X	X	X	X	X
	solid	X	X	X	X	X
	xDimension	X	X	X	X	X
	xSpacing	X	X	X	X	X
	zDimension	X	X	X	X	X
	zSpacing	X	X	X	X	X
<a href="#">GeoLocation</a>	addChildren	X	X	X	X	X
	removeChildren	X	X	X	X	X
	children	X	X	X	X	X
	geoCoord	X	X	X	X	X
	metadata	X	X	X	X	X
	geoOrigin	X	X	X	X	X
	geoSystem	X	X	X	X	X
	bboxCenter	X	X	X	X	X
	bboxSize	X	X	X	X	X
<a href="#">GeoLOD</a>	metadata	X	X	X	X	X
	children	X	X	X	X	X
	center	X	X	X	X	X
	child1Url	X	X	X	X	X
	child2Url	X	X	X	X	X
	child3Url	X	X	X	X	X
	child4Url	X	X	X	X	X
	geoOrigin	X	X	X	X	X
	geoSystem	X	X	X	X	X
	level_changed		X	X	X	X
	range	X	X	X	X	X
	rootUrl	X	X	X	X	X
	rootNode	X	X	X	X	X
	bboxCenter	X	X	X	X	X
bboxSize	X	X	X	X	X	
<a href="#">GeoMetadata</a>	data	X	X	X	X	X
	metadata	X	X	X	X	X
	summary	X	X	X	X	X
	url	X	X	X	X	X
<a href="#">GeoOrigin</a> (omitted as of 3.3)	geoCoord	X	X	X		X
	metadata	X	X	X		X
	geoSystem	X	X	X		X

	rotateYUp	X	X	X		X
<a href="#">GeoPositionInterpolator</a>	set_fraction	X	X	X	X	X
	key	X	X	X	X	X
	keyValue	X	X	X	X	X
	metadata	X	X	X	X	X
	geovalue_changed	X	X	X	X	X
	value_changed	X	X	X	X	X
	geoOrigin	X	X	X	X	X
	geoSystem	X	X	X	X	X
<a href="#">GeoProximitySensor</a>	enabled			X	X	X
	geoCenter			X		
	center				X	X
	metadata			X	X	X
	size			X	X	X
	centerOfRotation_changed			X	X	X
	enterTime			X	X	X
	exitTime			X	X	X
	geoCoord_changed			X	X	X
	isActive			X	X	X
	orientation_changed			X	X	X
	position_changed			X	X	X
	geoOrigin			X	X	X
	geoSystem			X	X	X
<a href="#">GeoTouchSensor</a>	description	X	X	X	X	X
	enabled	X	X	X	X	X
	metadata	X	X	X	X	X
	hitNormal_changed	X	X	X	X	X
	hitPoint_changed	X	X	X	X	X
	hitTexCoord_changed	X	X	X	X	X
	hitGeoCoord_changed	X	X	X	X	X
	isActive	X	X	X	X	X
	isOver	X	X	X	X	X
	touchTime	X	X	X	X	X
	geoOrigin	X	X	X	X	X
geoSystem	X	X	X	X	X	
	addChildren			X	X	X
	removeChildren			X	X	X
	children			X	X	X
	geoCenter			X	X	X
	metadata			X	X	X
	rotation			X	X	X

<a href="#">GeoTransform</a>	scale			X	X	X
	scaleOrientation			X	X	X
	translation			X	X	X
	bboxCenter			X	X	X
	bboxSize			X	X	X
	geoOrigin			X	X	X
	geoSystem			X	X	X
<a href="#">GeoViewpoint</a>	set_bind	X	X	X	X	X
	set_orientation	X	X	X	X	X
	set_position	X	X	X	X	X
	description	X	X	X	X	X
	fieldOfView	X	X	X	X	X
	headlight	X	X	X		
	jump	X	X	X	X	X
	metadata	X	X	X	X	X
	navType	X	X	X		
	bindTime	X	X	X	X	X
	isBound	X	X	X	X	X
	geoOrigin	X	X	X	X	X
	geoSystem	X	X	X	X	X
	orientation	X	X	X	X	X
	position	X	X	X	X	X
speedFactor	X	X	X	X	X	
<a href="#">Group</a>	addchildren	X	X	X	X	X
	removeChildren	X	X	X	X	X
	children	X	X	X	X	X
	metadata	X	X	X	X	X
	bboxCenter	X	X	X	X	X
	bboxSize	X	X	X	X	X
<a href="#">HAnimDisplacer</a>	coordIndex	X	X	X	X	X
	displacements	X	X	X	X	X
	metadata	X	X	X	X	X
	name	X	X	X	X	X
	weight	X	X	X	X	X
	center	X	X	X	X	X
	info	X	X	X	X	X
	joints	X	X	X	X	X
	metadata	X	X	X	X	X
	name	X	X	X	X	X
	rotation	X	X	X	X	X
	scale	X	X	X	X	X



<a href="#">HAnimHumanoid</a>	scaleOrientation	X	X	X	X	X
	segments	X	X	X	X	X
	sites	X	X	X	X	X
	skeleton	X	X	X	X	X
	skin	X	X	X	X	X
	skinCoord	X	X	X	X	X
	skinNormal	X	X	X	X	X
	translation	X	X	X	X	X
	version	X	X	X	X	X
	viewpoints	X	X	X	X	X
	bboxCenter	X	X	X	X	X
	bboxSize	X	X	X	X	X
<a href="#">HAnimJoint</a>	addChildren	X	X	X	X	X
	removeChildren	X	X	X	X	X
	center	X	X	X	X	X
	children	X	X	X	X	X
	displacers	X	X	X	X	X
	limitOrientation	X	X	X	X	X
	llimit	X	X	X	X	X
	metadata	X	X	X	X	X
	name	X	X	X	X	X
	rotation	X	X	X	X	X
	scale	X	X	X	X	X
	scaleOrientation	X	X	X	X	X
	skinCoordIndex	X	X	X	X	X
	skinCoordWeight	X	X	X	X	X
	stiffness	X	X	X	X	X
	translation	X	X	X	X	X
	ulimit	X	X	X	X	X
	bboxCenter	X	X	X	X	X
bboxSize	X	X	X	X	X	
<a href="#">HAnimSegment</a>	addChildren	X	X	X	X	X
	removeChildren	X	X	X	X	X
	centerOfMass	X	X	X	X	X
	children	X	X	X	X	X
	coord	X	X	X	X	X
	displacers	X	X	X	X	X
	mass	X	X	X	X	X
	metadata	X	X	X	X	X
	momentsOfInertia	X	X	X	X	X
	name	X	X	X	X	X

	bboxCenter	X	X	X	X	X
	bboxSize	X	X	X	X	X
<a href="#">HAnimSite</a>	addChildren	X	X	X	X	X
	removeChildren	X	X	X	X	X
	center	X	X	X	X	X
	children	X	X	X	X	X
	metadata	X	X	X	X	X
	name	X	X	X	X	X
	rotation	X	X	X	X	X
	scale	X	X	X	X	X
	scaleOrientation	X	X	X	X	X
	translation	X	X	X	X	X
	bboxCenter	X	X	X	X	X
	bboxSize	X	X	X	X	X
	<a href="#">ImageCubeMapTexture</a>	metadata		X	X	X
url			X	X	X	X
<a href="#">ImageTexture</a>	metadata	X	X	X	X	X
	url	X	X	X	X	X
	repeatS	X	X	X	X	X
	repeatT	X	X	X	X	X
<a href="#">ImageTexture3D</a>	metadata		X	X	X	X
	url		X	X	X	X
	repeatS		X	X	X	X
	repeatT		X	X	X	X
	repeatR		X	X	X	X
<a href="#">IndexedFaceSet</a>	set_colorIndex	X	X	X	X	X
	set_coordIndex	X	X	X	X	X
	set_normalIndex	X	X	X	X	X
	set_texCoordIndex	X	X	X	X	X
	attrib		X	X	X	X
	color	X	X	X	X	X
	coord	X	X	X	X	X
	fogCoord		X	X	X	X
	metadata	X	X	X	X	X
	normal	X	X	X	X	X
	texCoord	X	X	X	X	X
	ccw	X	X	X	X	X
	colorIndex	X	X	X	X	X
	colorPerVertex	X	X	X	X	X
	convex	X	X	X	X	X
coordIndex	X	X	X	X	X	

	creaseAngle	X	X	X	X	X
	normalIndex	X	X	X	X	X
	normalPerVertex	X	X	X	X	X
	solid	X	X	X	X	X
	texCoordIndex	X	X	X	X	X
<a href="#">IndexedLineSet</a>	set_colorIndex	X	X	X	X	X
	set_coordIndex	X	X	X	X	X
	attrib		X	X	X	X
	color	X	X	X	X	X
	coord	X	X	X	X	X
	fogCoord		X	X	X	X
	metadata	X	X	X	X	X
	colorIndex	X	X	X	X	X
	colorPerVertex	X	X	X	X	X
	coordIndexd	X	X	X	X	X
<a href="#">IndexedQuadSet</a>	set_index		X	X	X	X
	attrib		X	X	X	X
	color		X	X	X	X
	coord		X	X	X	X
	fogCoord		X	X	X	X
	metadata		X	X	X	X
	normal		X	X	X	X
	texCoord		X	X	X	X
	ccw		X	X	X	X
	colorPerVertex		X	X	X	X
	normalPerVertex		X	X	X	X
	solid		X	X	X	X
	index		X	X	X	X
<a href="#">IndexedTriangleFanSet</a>	set_index	X	X	X	X	X
	attrib		X	X	X	X
	color	X	X	X	X	X
	coord	X	X	X	X	X
	fogCoord		X	X	X	X
	metadata	X	X	X	X	X
	normal	X	X	X	X	X
	texCoord	X	X	X	X	X
	ccw	X	X	X	X	X
	colorPerVertex	X	X	X	X	X
	normalPerVertex	X	X	X	X	X
	solid	X	X	X	X	X
	index	X	X	X	X	X

<a href="#">IndexedTriangleSet</a>	set_index	X	X	X	X	X
	attrib		X	X	X	X
	color	X	X	X	X	X
	coord	X	X	X	X	X
	fogCoord		X	X	X	X
	metadata	X	X	X	X	X
	normal	X	X	X	X	X
	texCoord	X	X	X	X	X
	ccw	X	X	X	X	X
	colorPerVertex	X	X	X	X	X
	normalPerVertex	X	X	X	X	X
	solid	X	X	X	X	X
	index	X	X	X	X	X
<a href="#">IndexedTriangleStripSet</a>	set_index	X	X	X	X	X
	attrib		X	X	X	X
	color	X	X	X	X	X
	coord	X	X	X	X	X
	creaseAngle	X	X	X	X	X
	fogCoord		X	X	X	X
	metadata	X	X	X	X	X
	normal	X	X	X	X	X
	texCoord	X	X	X	X	X
	ccw	X	X	X	X	X
	normalPerVertex	X	X	X	X	X
	solid	X	X	X	X	X
	index	X	X	X	X	X
<a href="#">Inline</a>	load	X	X	X	X	X
	metadata	X	X	X	X	X
	url	X	X	X	X	X
	bboxCenter	X	X	X	X	X
	bboxSize	X	X	X	X	X
<a href="#">IntegerSequencer</a>	next	X	X	X	X	X
	previous	X	X	X	X	X
	set_fraction	X	X	X	X	X
	key	X	X	X	X	X
	keyValue	X	X	X	X	X
	metadata	X	X	X	X	X
	value_changed	X	X	X	X	X
<a href="#">IntegerTrigger</a>	set_boolean	X	X	X	X	X
	integerKey	X	X	X	X	X
	metadata	X	X	X	X	X

	triggerValue	X	X	X	X	X
<a href="#">IsoSurfaceVolumeData</a>	contourStepSize				X	X
	dimensions				X	X
	gradients				X	X
	metadata				X	X
	renderStyle				X	X
	surfaceTolerance				X	X
	surfaceValues				X	X
	voxels				X	X
	bboxCenter				X	X
	bboxSize				X	X
<a href="#">KeySensor</a>	enabled	X	X	X	X	X
	metadata	X	X	X	X	X
	actionKeyPress	X	X	X	X	X
	actionKeyRelease	X	X	X	X	X
	altKey	X	X	X	X	X
	controlKey	X	X	X	X	X
	isActive	X	X	X	X	X
	keyPress	X	X	X	X	X
	keyRelease	X	X	X	X	X
	shiftKey	X	X	X	X	X
<a href="#">Layer</a>	addChildren			X	X	X
	removeChildren			X	X	X
	children			X	X	X
	pickable			X	X	X
	metadata			X	X	X
	viewport			X	X	X
<a href="#">LayerSet</a>	activeLayer			X	X	X
	layers			X	X	X
	metadata			X	X	X
	order			X	X	X
<a href="#">Layout</a>	align			X	X	X
	metadata			X	X	X
	offset			X	X	X
	offsetUnits			X	X	X
	scaleMode			X	X	X
	size			X	X	X
	sizeUnits			X	X	X
<a href="#">LayoutGroup</a>	addChildren			X	X	X
	removeChildren			X	X	X
	children			X	X	X

	layout			X	X	X
	metadata			X	X	X
	viewport			X	X	X
<a href="#">LayoutLayer</a>	addChildren			X	X	X
	removeChildren			X	X	X
	children			X	X	X
	pickable			X	X	X
	layout			X	X	X
	metadata			X	X	X
	viewport			X	X	X
<a href="#">LinePickSensor</a>	enabled			X	X	X
	metadata			X	X	X
	objectType			X	X	X
	pickingGeometry			X	X	X
	pickTarget			X	X	X
	isActive			X	X	X
	pickedGeometry			X	X	X
	pickedNormal			X	X	X
	pickedPoint			X	X	X
	pickedTextureCoordinate			X	X	X
	intersectionType			X	X	X
sortOrder			X	X	X	
<a href="#">LineProperties</a>	applied	X	X	X	X	X
	linetype	X	X	X	X	X
	linewidthScaleFactor	X	X	X	X	X
	metadata	X	X	X	X	X
<a href="#">LineSet</a>	attrib		X	X	X	X
	color	X	X	X	X	X
	coord	X	X	X	X	X
	fogCoord		X	X	X	X
	metadata	X	X	X	X	X
	vertexCount	X	X	X	X	X
<a href="#">LoadSensor</a>	enabled	X	X	X	X	X
	metadata	X	X	X	X	X
	timeOut	X	X	X	X	X
	watchList	X	X	X	X	X
	isActive	X	X	X	X	X
	isLoading	X	X	X	X	X
	loadTime	X	X	X	X	X
	progress	X	X	X	X	X
	color		X	X	X	X

<a href="#">LocalFog</a>	enabled		X	X	X	X
	fogType		X	X	X	X
	metadata		X	X	X	X
	visibilityRange		X	X	X	X
<a href="#">LOD</a>	addChildren	X	X	X	X	X
	removeChildren	X	X	X	X	X
	children	X	X	X	X	X
	metadata	X	X	X	X	X
	level_changed		X	X	X	X
	bboxCenter	X	X	X	X	X
	bboxSize	X	X	X	X	X
	center	X	X	X	X	X
	forceTransitions		X	X	X	X
	range	X	X	X	X	X
<a href="#">Material</a>	ambientIntensity	X	X	X	X	X
	ambientTexture					X
	ambientTextureMapping					X
	diffuseColor	X	X	X	X	X
	diffuseTexture					X
	diffuseTextureMapping					X
	emissiveColor	X	X	X	X	X
	emissiveTexture					X
	emissiveTextureMapping					X
	metadata	X	X	X	X	X
	normalTexture					X
	normalTextureMapping					X
	normalScale					X
	occlusionStrength					X
	occlusionTexture					X
	occlusionTextureMapping					X
	shininess	X	X	X	X	X
	shininessTexture					X
	shininessTextureMapping					X
	specularColor	X	X	X	X	X
specularTexture					X	
specularTextureMapping					X	
transparency	X	X	X	X	X	
<a href="#">Matrix3VertexAttribute</a>	metadata		X	X	X	X
	value		X	X	X	X
	name		X	X	X	X
	metadata		X	X	X	X

<a href="#">Matrix4VertexAttribute</a>	value		X	X	X	X
	name		X	X	X	X
<a href="#">MetadataBoolean</a>	metadata				X	X
	name				X	X
	reference				X	X
	value				X	X
<a href="#">MetadataDouble</a>	metadata	X	X	X	X	X
	name	X	X	X	X	X
	reference	X	X	X	X	X
	value	X	X	X	X	X
<a href="#">MetadataFloat</a>	metadata	X	X	X	X	X
	name	X	X	X	X	X
	reference	X	X	X	X	X
	value	X	X	X	X	X
<a href="#">MetadataInteger</a>	metadata	X	X	X	X	X
	name	X	X	X	X	X
	reference	X	X	X	X	X
	value	X	X	X	X	X
<a href="#">MetadataSet</a>	metadata	X	X	X	X	X
	name	X	X	X	X	X
	reference	X	X	X	X	X
	value	X	X	X	X	X
<a href="#">MetadataString</a>	metadata	X	X	X	X	X
	name	X	X	X	X	X
	reference	X	X	X	X	X
	value	X	X	X	X	X
<a href="#">MotorJoint</a>	axis1Angle			X	X	X
	axis1Torque			X	X	X
	axis2Angle			X	X	X
	axis2Torque			X	X	X
	axis3Angle			X	X	X
	axis3Torque			X	X	X
	body1			X	X	X
	body2			X	X	X
	enabledAxes			X	X	X
	metadata			X	X	X
	motor1Axis			X	X	X
	motor2Axis			X	X	X
	motor3Axis			X	X	X
	mustOutput			X	X	X
stop1Bounce			X	X	X	



	stop1ErrorCorrection			X	X	X
	stop2Bounce			X	X	X
	stop2ErrorCorrection			X	X	X
	stop3Bounce			X	X	X
	stop3ErrorCorrection			X	X	X
	motor1Angle			X	X	X
	motor1AngleRate			X	X	X
	motor2Angle			X	X	X
	motor2AngleRate			X	X	X
	motor3Angle			X	X	X
	motor3AngleRate			X	X	X
	autoCalc			X	X	X
<a href="#">MovieTexture</a>	loop	X	X	X	X	X
	metadata	X	X	X	X	X
	pauseTime	X	X	X	X	X
	pitch				X	X
	resumeTime	X	X	X	X	X
	speed	X	X	X	X	X
	startTime	X	X	X	X	X
	stopTime	X	X	X	X	X
	url	X	X	X	X	X
	repeatS	X	X	X	X	X
	repeatT	X	X	X	X	X
	duration_changed	X	X	X	X	X
	elapsedTime	X	X	X	X	X
	isActive	X	X	X	X	X
	isPaused	X	X	X	X	X
<a href="#">MultiTexture</a>	alpha	X	X	X	X	X
	color	X	X	X	X	X
	function	X	X	X	X	X
	metadata	X	X	X	X	X
	mode	X	X	X	X	X
	source	X	X	X	X	X
	texture	X	X	X	X	X
<a href="#">MultiTextureCoordinate</a>	metadata	X	X	X	X	X
	texCoord	X	X	X	X	X
<a href="#">MultiTextureTransform</a>	metadata	X	X	X	X	X
	textureTransform	X	X	X	X	X
	set_bind	X	X	X	X	X
	avatarSize	X	X	X	X	X
	headlight	X	X	X	X	X

<a href="#">NavigationInfo</a>	metadata	X	X	X	X	X
	speed	X	X	X	X	X
	transitionTime		X	X	X	X
	transitionType	X	X	X	X	X
	type	X	X	X	X	X
	visibilityLimit	X	X	X	X	X
	bindTime	X	X	X	X	X
	isBound	X	X	X	X	X
	transitionComplete		X	X	X	X
<a href="#">Normal</a>	metadata	X	X	X	X	X
	vector	X	X	X	X	X
<a href="#">NormalInterpolator</a>	set_fraction	X	X	X	X	X
	key	X	X	X	X	X
	keyValue	X	X	X	X	X
	metadata	X	X	X	X	X
	value_changed	X	X	X	X	X
<a href="#">NurbsCurve</a>	controlPoint	X	X	X	X	X
	metadata	X	X	X	X	X
	tessellation	X	X	X	X	X
	weight	X	X	X	X	X
	closed	X	X	X	X	X
	knot	X	X	X	X	X
	order	X	X	X	X	X
<a href="#">NurbsCurve2D</a>	controlPoint	X	X	X	X	X
	metadata	X	X	X	X	X
	tessellation	X	X	X	X	X
	weight	X	X	X	X	X
	knot	X	X	X	X	X
	order	X	X	X	X	X
	closed	X	X	X	X	X
<a href="#">NurbsOrientationInterpolator</a>	set_fraction	X	X	X	X	X
	controlPoint	X	X	X	X	X
	knot	X	X	X	X	X
	metadata	X	X	X	X	X
	order	X	X	X	X	X
	weight	X	X	X	X	X
	value_changed	X	X	X	X	X
	controlPoint	X	X	X	X	X
	metadata	X	X	X	X	X
	texCoord	X	X	X	X	X
	uTessellation	X	X	X	X	X

<a href="#">NurbsPatchSurface</a>	vTessellation	X	X	X	X	X
	weight	X	X	X	X	X
	solid	X	X	X	X	X
	uClosed	X	X	X	X	X
	uDimension	X	X	X	X	X
	uKnot	X	X	X	X	X
	uOrder	X	X	X	X	X
	vClosed	X	X	X	X	X
	vDimension	X	X	X	X	X
	vKnot	X	X	X	X	X
	vOrder	X	X	X	X	X
<a href="#">NurbsPositionInterpolator</a>	set_fraction	X	X	X	X	X
	controlPoint	X	X	X	X	X
	knot	X	X	X	X	X
	metadata	X	X	X	X	X
	order	X	X	X	X	X
	weight	X	X	X	X	X
	value_changed	X	X	X	X	X
<a href="#">NurbsSet</a>	addGeometry	X	X	X	X	X
	removeGeometry	X	X	X	X	X
	geometry	X	X	X	X	X
	metadata	X	X	X	X	X
	tessellationScale	X	X	X	X	X
	bboxCenter	X	X	X	X	X
	bboxSize	X	X	X	X	X
<a href="#">NurbsSurfaceInterpolator</a>	set_fraction	X	X	X	X	X
	controlPoint	X	X	X	X	X
	metadata	X	X	X	X	X
	weight	X	X	X	X	X
	position_changed	X	X	X	X	X
	normal_changed	X	X	X	X	X
	uDimension	X	X	X	X	X
	uKnot	X	X	X	X	X
	uOrder	X	X	X	X	X
	vDimension	X	X	X	X	X
	vKnot	X	X	X	X	X
vOrder	X	X	X	X	X	
<a href="#">NurbsSweptSurface</a>	crossSectionCurve	X	X	X	X	X
	metadata	X	X	X	X	X
	trajectoryCurve	X	X	X	X	X
	ccw	X	X	X	X	X

	solid	X	X	X	X	X
<a href="#">NurbsSwungSurface</a>	metadata	X	X	X	X	X
	profileCurve	X	X	X	X	X
	trajectoryCurve	X	X	X	X	X
	ccw	X	X	X	X	X
	solid	X	X	X	X	X
<a href="#">NurbsTextureCoordinate</a>	controlPoint	X	X	X	X	X
	metadata	X	X	X	X	X
	weight	X	X	X	X	X
	uDimension	X	X	X	X	X
	uKnot	X	X	X	X	X
	uOrder	X	X	X	X	X
	vDimension	X	X	X	X	X
	vKnot	X	X	X	X	X
<a href="#">NurbsTrimmedSurface</a>	addTrimmingContour	X	X	X	X	X
	removeTrimmingContour	X	X	X	X	X
	controlPoint	X	X	X	X	X
	metadata	X	X	X	X	X
	texCoord	X	X	X	X	X
	trimmingContour	X	X	X	X	X
	uTessellation	X	X	X	X	X
	vTessellation	X	X	X	X	X
	weight	X	X	X	X	X
	solid	X	X	X	X	X
	uClosed	X	X	X	X	X
	uDimension	X	X	X	X	X
	uKnot	X	X	X	X	X
	uOrder	X	X	X	X	X
	vClosed	X	X	X	X	X
	vDimension	X	X	X	X	X
<a href="#">OpacityMapVolumeStyle</a>	enabled				X	X
	metadata				X	X
	transferFunction				X	X
<a href="#">OrientationChaser</a>	set_destination			X	X	X
	set_value			X	X	X
	metadata			X	X	X
	isActive			X	X	X
	value_changed			X	X	X

	duration			X	X	X
	initialDestination			X	X	X
	initialValue			X	X	X
<a href="#">OrientationDamper</a>	set_destination			X	X	X
	set_value			X	X	X
	metadata			X	X	X
	tau			X	X	X
	tolerance			X	X	X
	isActive			X	X	X
	value_changed			X	X	X
	initialDestination			X	X	X
	initialValue			X	X	X
	order			X	X	X
<a href="#">OrientationInterpolator</a>	set_fraction	X	X	X	X	X
	key	X	X	X	X	X
	keyValue	X	X	X	X	X
	metadata	X	X	X	X	X
	value_changed	X	X	X	X	X
<a href="#">OrthoViewpoint</a>	set_bind			X	X	X
	centerOfRotation			X	X	X
	description			X	X	X
	fieldOfView			X	X	X
	jump			X	X	X
	metadata			X	X	X
	orientation			X	X	X
	position			X	X	X
	retainUserOffsets			X	X	X
	bindTime			X	X	X
	isBound			X	X	X
<a href="#">PackagedShader</a>	activate		X	X	X	X
	metadata		X	X	X	X
	url		X	X	X	X
	isSelected		X	X	X	X
	isValid		X	X	X	X
	language		X	X	X	X
	Any number of additional fields as specified in <a href="#">31.4.4 PackagedShader</a> .		X	X	X	X
	appearance			X	X	X
	createParticles			X	X	X
	geometry			X	X	X
	enabled			X	X	X

<a href="#">ParticleSystem</a>	lifetimeVariation			X	X	X
	maxParticles			X	X	X
	metadata			X	X	X
	particleLifetime			X	X	X
	particleSize			X	X	X
	isActive			X	X	X
	bboxCenter			X	X	X
	bboxSize			X	X	X
	colorRamp			X	X	X
	colorKey			X	X	X
	emitter			X	X	X
	geometryType			X	X	X
	physics			X	X	X
	texCoordRamp			X	X	X
texCoordKey			X	X	X	
<a href="#">PhysicalMaterial</a>	baseColor					X
	baseTexture					X
	baseTextureMapping					X
	metallic					X
	metallicRoughnessTexture					X
	metallicRoughnessTextureMapping					X
	emissiveColor					X
	emissiveTexture					X
	emissiveTextureMapping					X
	metadata					X
	normalTexture					X
	normalTextureMapping					X
	normalScale					X
	occlusionStrength					X
	occlusionTexture					X
	occlusionTextureMapping					X
roughness					X	
transparency					X	
<a href="#">PickableGroup</a>	addChildren			X	X	X
	removeChildren			X	X	X
	children			X	X	X
	metadata			X	X	X
	objectType			X	X	X
	pickable			X	X	X
	bboxCenter			X	X	X
	bboxSize			X	X	X

<a href="#">PixelTexture</a>	image	X	X	X	X	X
	metadata	X	X	X	X	X
	repeatS	X	X	X	X	X
	repeatT	X	X	X	X	X
<a href="#">PixelTexture3D</a>	metadata		X	X	X	X
	image		X	X	X	X
	repeatS		X	X	X	X
	repeatT		X	X	X	X
	repeatR		X	X	X	X
<a href="#">PlaneSensor</a>	autoOffset	X	X	X	X	X
	description	X	X	X	X	X
	enabled	X	X	X	X	X
	maxPosition	X	X	X	X	X
	metadata	X	X	X	X	X
	minPosition	X	X	X	X	X
	offset	X	X	X	X	X
	isActive	X	X	X	X	X
	isOver	X	X	X	X	X
	trackPoint_changed	X	X	X	X	X
	translation_changed	X	X	X	X	X
<a href="#">PointEmitter</a>	direction			X	X	X
	metadata			X	X	X
	position			X	X	X
	speed			X	X	X
	variation			X	X	X
	mass			X	X	X
	surfaceArea			X	X	X
<a href="#">PointLight</a>	ambientIntensity	X	X	X	X	X
	attenuation	X	X	X	X	X
	color	X	X	X	X	X
	global		X	X	X	X
	intensity	X	X	X	X	X
	location	X	X	X	X	X
	metadata	X	X	X	X	X
	on	X	X	X	X	X
	radius	X	X	X	X	X
<a href="#">PointPickSensor</a>	enabled			X	X	X
	metadata			X	X	X
	objectType			X	X	X
	pickingGeometry			X	X	X
	pickTarget			X	X	X

	isActive			X	X	X
	pickedGeometry			X	X	X
	pickedPoint			X	X	X
	intersectionType			X	X	X
	sortOrder			X	X	X
<a href="#">PointSet</a>	attrib		X	X	X	X
	color	X	X	X	X	X
	coord	X	X	X	X	X
	fogCoord		X	X	X	X
	metadata	X	X	X	X	X
<a href="#">PolylineEmitter</a>	set_coordIndex			X	X	X
	coord			X	X	X
	direction			X	X	X
	metadata			X	X	X
	speed			X	X	X
	variation			X	X	X
	coordIndex			X	X	X
	mass			X	X	X
	surfaceArea			X	X	X
<a href="#">Polyline2D</a>	metadata	X	X	X	X	X
	lineSegments	X	X	X	X	X
<a href="#">Polypoint2D</a>	metadata	X	X	X	X	X
	point	X	X	X	X	X
<a href="#">PositionChaser</a>	set_destination			X	X	X
	set_value			X	X	X
	metadata			X	X	X
	isActive			X	X	X
	value_changed			X	X	X
	duration			X	X	X
	initialDestination			X	X	X
	initialValue			X	X	X
<a href="#">PositionChaser2D</a>	set_destination			X	X	X
	set_value			X	X	X
	metadata			X	X	X
	isActive			X	X	X
	value_changed			X	X	X
	duration			X	X	X
	initialDestination			X	X	X
	initialValue			X	X	X
	set_destination			X	X	X
	set_value			X	X	X



<a href="#">PositionDamper</a>	metadata			X	X	X
	tau			X	X	X
	tolerance			X	X	X
	isActive			X	X	X
	value_changed			X	X	X
	initialDestination			X	X	X
	initialValue			X	X	X
	order			X	X	X
<a href="#">PositionDamper2D</a>	set_destination			X	X	X
	set_value			X	X	X
	metadata			X	X	X
	tau			X	X	X
	tolerance			X	X	X
	isActive			X	X	X
	value_changed			X	X	X
	initialDestination			X	X	X
<a href="#">PositionInterpolator</a>	initialValue			X	X	X
	order			X	X	X
	set_fraction	X	X	X	X	X
	key	X	X	X	X	X
	keyValue	X	X	X	X	X
<a href="#">PositionInterpolator2D</a>	metadata	X	X	X	X	X
	value_changed	X	X	X	X	X
	set_fraction	X	X	X	X	X
	key	X	X	X	X	X
	keyValue	X	X	X	X	X
<a href="#">PrimitivePickSensor</a>	metadata	X	X	X	X	X
	value_changed	X	X	X	X	X
	enabled			X	X	X
	metadata			X	X	X
	objectType			X	X	X
	pickingGeometry			X	X	X
	pickTarget			X	X	X
	isActive			X	X	X
	pickedGeometry			X	X	X
intersectionType			X	X	X	
<a href="#">ProgramShader</a>	sortOrder			X	X	X
	activate		X	X	X	X
	metadata		X	X	X	X
	programs		X	X	X	X
	isSelected		X	X	X	X

	isValid		X	X	X	X
	language		X	X	X	X
<a href="#">ProjectionVolumeStyle</a>	enabled				X	X
	metadata				X	X
	intensityThreshold				X	X
	type				X	X
<a href="#">ProximitySensor</a>	center	X	X	X	X	X
	enabled	X	X	X	X	X
	metadata	X	X	X	X	X
	size	X	X	X	X	X
	enterTime	X	X	X	X	X
	exitTime	X	X	X	X	X
	centerOfRotation_changed	X	X	X	X	X
	isActive	X	X	X	X	X
	orientation_changed	X	X	X	X	X
position_changed	X	X	X	X	X	
<a href="#">QuadSet</a>	attrib		X	X	X	X
	color		X	X	X	X
	coord		X	X	X	X
	fogCoord		X	X	X	X
	metadata		X	X	X	X
	normal		X	X	X	X
	texCoord		X	X	X	X
	ccw		X	X	X	X
	colorPerVertex		X	X	X	X
	normalPerVertex		X	X	X	X
	solid		X	X	X	X
	address	X	X	X	X	X
	applicationID		X	X	X	X
	enabled		X	X	X	X
	entityID		X	X	X	X
	metadata		X	X	X	X
	multicastRelayHost		X	X	X	X
	multicastRelayPort		X	X	X	X
	networkMode		X	X	X	X
	port		X	X	X	X
	radioID		X	X	X	X
	readInterval		X	X	X	X
	receivedPower		X	X	X	X
	receiverState		X	X	X	X
	rtpHeaderExpected		X	X	X	X

<a href="#">ReceiverPdu</a>	siteID		X	X	X	X
	transmitterApplicationID		X	X	X	X
	transmitterEntityID		X	X	X	X
	transmitterRadioID		X	X	X	X
	transmitterSiteID		X	X	X	X
	whichGeometry		X	X	X	X
	writeInterval		X	X	X	X
	isActive		X	X	X	X
	isNetworkReader		X	X	X	X
	isNetworkWriter		X	X	X	X
	isRtpHeaderHeard		X	X	X	X
	isStandAlone		X	X	X	X
	timestamp		X	X	X	X
	bboxCenter		X	X	X	X
	bboxSize	X	X	X	X	X
<a href="#">Rectangle2D</a>	metadata	X	X	X	X	X
	size	X	X	X	X	X
	solid	X	X	X	X	X
<a href="#">RigidBody</a>	angularDampingFactor			X	X	X
	angularVelocity			X	X	X
	autoDamp			X	X	X
	autoDisable			X	X	X
	centerOfMass			X	X	X
	disableAngularSpeed			X	X	X
	disableLinearSpeed			X	X	X
	disableTime			X	X	X
	enabled			X	X	X
	finiteRotationAxis			X	X	X
	fixed			X	X	X
	forces			X	X	X
	geometry			X	X	X
	inertia			X	X	X
	linearDampingFactor			X	X	X
	linearVelocity			X	X	X
	mass			X	X	X
	massDensityModel			X	X	X
	metadata			X	X	X
	orientation			X	X	X
	position			X	X	X
torques			X	X	X	
useFiniteRotation			X	X	X	

	useGlobalGravity			X	X	X
<a href="#">RigidBodyCollection</a>	set_contacts			X	X	X
	autoDisable			X	X	X
	bodies			X	X	X
	constantForceMix			X	X	X
	constantSurfaceThickness			X	X	X
	disableAngularSpeed			X	X	X
	disableLinearSpeed			X	X	X
	disableTime			X	X	X
	enabled			X	X	X
	errorCorrection			X	X	X
	gravity			X	X	X
	iterations			X	X	X
	joints			X	X	X
	maxCorrectionSpeed			X	X	X
	metadata			X	X	X
	preferAccuracy			X	X	X
collider			X	X	X	
<a href="#">ScalarChaser</a>	set_destination			X	X	X
	set_value			X	X	X
	metadata			X	X	X
	isActive			X	X	X
	value_changed			X	X	X
	duration			X	X	X
	initialDestination			X	X	X
	initialValue			X	X	X
<a href="#">ScalarDamper</a>	set_destination				X	X
	set_value				X	X
	metadata				X	X
	tau				X	X
	tolerance				X	X
	isActive				X	X
	value_changed				X	X
	initialDestination				X	X
	initialValue				X	X
	order				X	X
<a href="#">ScalarInterpolator</a>	set_fraction	X	X	X	X	X
	key	X	X	X	X	X
	keyValue	X	X	X	X	X
	metadata	X	X	X	X	X
	value_changed	X	X	X	X	X

<a href="#">ScreenFontStyle</a>	metadata			X	X	X
	family			X	X	X
	horizontal			X	X	X
	justify			X	X	X
	language			X	X	X
	leftToRight			X	X	X
	pointSize			X	X	X
	spacing			X	X	X
	style			X	X	X
	topToBottom			X	X	X
<a href="#">ScreenGroup</a>	addChildren			X	X	X
	removeChildren			X	X	X
	children			X	X	X
	metadata			X	X	X
	bboxCenter			X	X	X
	bboxSize			X	X	X
<a href="#">Script</a>	metadata	X	X	X	X	X
	url	X	X	X	X	X
	directOutput	X	X	X	X	X
	mustEvaluate	X	X	X	X	X
	Any number of additional fields as specified in <a href="#">29.4.1 Script</a> .	X	X	X	X	X
<a href="#">SegmentedVolumeData</a>	dimensions				X	X
	metadata				X	X
	renderStyle				X	X
	segmentEnabled				X	X
	segmentIdentifiers				X	X
	voxels				X	X
	bboxCenter				X	X
	bboxSize				X	X
<a href="#">ShadedVolumeStyle</a>	enabled				X	X
	lighting				X	X
	material				X	X
	metadata				X	X
	shadows				X	X
	surfaceNormals				X	X
	phaseFunction				X	X
<a href="#">ShaderPart</a>	metadata		X	X	X	X
	url		X	X	X	X
	type		X	X	X	X
	metadata		X	X	X	X

<a href="#">ShaderProgram</a>	url		X	X	X	X
	type		X	X	X	X
	Any number of additional fields as specified in <a href="#">31.4.7 ShaderProgram</a> .		X	X	X	X
<a href="#">Shape</a>	appearance	X	X	X	X	X
	geometry	X	X	X	X	X
	metadata	X	X	X	X	X
	bboxCenter	X	X	X	X	X
	bboxSize	X	X	X	X	X
<a href="#">SignalPdu</a>	address	X	X	X	X	X
	applicationID	X	X	X	X	X
	data	X	X	X	X	X
	dataLength	X	X	X	X	X
	enabled		X	X	X	X
	encodingScheme	X	X	X	X	X
	entityID	X	X	X	X	X
	metadata	X	X	X	X	X
	multicastRelayHost	X	X	X	X	X
	multicastRelayPort	X	X	X	X	X
	networkMode	X	X	X	X	X
	port	X	X	X	X	X
	radioid	X	X	X	X	X
	readInterval	X	X	X	X	X
	rtpHeaderExpected	X	X	X	X	X
	sampleRate	X	X	X	X	X
	samples	X	X	X	X	X
	siteID	X	X	X	X	X
	tdlType	X	X	X	X	X
	whichGeometry	X	X	X	X	X
	writeInterval	X	X	X	X	X
	isActive	X	X	X	X	X
	isNetworkReader	X	X	X	X	X
	isNetworkWriter	X	X	X	X	X
	isRtpHeaderHeard	X	X	X	X	X
	isStandAlone	X	X	X	X	X
	timestamp	X	X	X	X	X
bboxCenter	X	X	X	X	X	
bboxSize	X	X	X	X	X	
<a href="#">SilhouetteEnhancementVolumeStyle</a>	enabled				X	X
	metadata				X	X
	silhouetteBoundaryOpacity				X	X
	silhouetteRetainedOpacity				X	X

	silhouetteSharpness				X	X
	surfaceNormals				X	X
<a href="#">SingleAxisHingeJoint</a>	anchorPoint			X	X	X
	axis			X	X	X
	body1			X	X	X
	body2			X	X	X
	maxAngle			X	X	X
	metadata			X	X	X
	minAngle			X	X	X
	mustOutput			X	X	X
	stopBounce			X	X	X
	stopErrorCorrection			X	X	X
	angle			X	X	X
	angleRate			X	X	X
	body1AnchorPoint			X	X	X
	body2AnchorPoint			X	X	X
<a href="#">SliderJoint</a>	axis			X	X	X
	body1			X	X	X
	body2			X	X	X
	maxSeparation			X	X	X
	metadata			X	X	X
	minSeparation			X	X	X
	mustOutput			X	X	X
	stopBounce			X	X	X
	stopErrorCorrection			X	X	X
	separation			X	X	X
	separationRate			X	X	X
<a href="#">Sound</a>	direction	X	X	X	X	X
	intensity	X	X	X	X	X
	location	X	X	X	X	X
	maxBack	X	X	X	X	X
	maxFront	X	X	X	X	X
	metadata	X	X	X	X	X
	minBack	X	X	X	X	X
	minFront	X	X	X	X	X
	priority	X	X	X	X	X
	source	X	X	X	X	X
	spatialize	X	X	X	X	X
<a href="#">Sphere</a>	metadata	X	X	X	X	X
	radius	X	X	X	X	X
	solid	X	X	X	X	X

<a href="#">SphereSensor</a>	autoOffset	X	X	X	X	X
	description	X	X	X	X	X
	enabled	X	X	X	X	X
	metadata	X	X	X	X	X
	offset	X	X	X	X	X
	isActive	X	X	X	X	X
	isOver	X	X	X	X	X
	rotation_changed	X	X	X	X	X
	trackPoint_changed	X	X	X	X	X
<a href="#">SplinePositionInterpolator</a>	set_fraction			X	X	X
	closed			X	X	X
	key			X	X	X
	keyValue			X	X	X
	keyVelocity			X	X	X
	metadata			X	X	X
	normalizeVelocity			X	X	X
	value_changed			X	X	X
<a href="#">SplinePositionInterpolator2D</a>	set_fraction			X	X	X
	closed			X	X	X
	key			X	X	X
	keyValue			X	X	X
	keyVelocity			X	X	X
	metadata			X	X	X
	normalizeVelocity			X	X	X
	value_changed			X	X	X
<a href="#">SplineScalarInterpolator</a>	set_fraction			X	X	X
	closed			X	X	X
	key			X	X	X
	keyValue			X	X	X
	keyVelocity			X	X	X
	metadata			X	X	X
	normalizeVelocity			X	X	X
	value_changed			X	X	X
<a href="#">SpotLight</a>	ambientIntensity	X	X	X	X	X
	attenuation	X	X	X	X	X
	beamWidth	X	X	X	X	X
	color	X	X	X	X	X
	cutOffAngle	X	X	X	X	X
	direction	X	X	X	X	X
	global		X	X	X	X
	intensity	X	X	X	X	X



	location	X	X	X	X	X
	metadata	X	X	X	X	X
	on	X	X	X	X	X
	radius	X	X	X	X	X
<a href="#">SquadOrientationInterpolator</a>	set_fraction			X	X	X
	key			X	X	X
	keyValue			X	X	X
	metadata			X	X	X
	normalizeVelocity			X	X	X
	value_changed			X	X	X
<a href="#">StaticGroup</a>	metadata	X	X	X	X	X
	children	X	X	X	X	X
	bboxCenter	X	X	X	X	X
	bboxSize	X	X	X	X	X
<a href="#">StringSensor</a>	deletionAllowed	X	X	X	X	X
	enabled	X	X	X	X	X
	metadata	X	X	X	X	X
	enteredText	X	X	X	X	X
	finalText	X	X	X	X	X
	isActive	X	X	X	X	X
<a href="#">SurfaceEmitter</a>	set_coordIndex			X	X	X
	metadata			X	X	X
	speed			X	X	X
	variation			X	X	X
	coordIndex			X	X	X
	mass			X	X	X
	surface			X	X	X
	surfaceArea			X	X	X
<a href="#">Switch</a>	addChildren	X	X	X	X	X
	removeChildren	X	X	X	X	X
	children	X	X	X	X	X
	metadata	X	X	X	X	X
	whichChoice	X	X	X	X	X
	bboxCenter	X	X	X	X	X
	bboxSize	X	X	X	X	X
<a href="#">TexCoordChaser2D</a>	set_destination				X	X
	set_value				X	X
	metadata				X	X
	isActive				X	X
	value_changed				X	X
	duration				X	X

	initialDestination				X	X
	initialValue				X	X
<a href="#">TexCoordDamper2D</a>	set_destination			X	X	X
	set_value			X	X	X
	metadata			X	X	X
	tau			X	X	X
	tolerance			X	X	X
	isActive			X	X	X
	value_changed			X	X	X
	initialDestination			X	X	X
	initialValue			X	X	X
	order			X	X	X
<a href="#">Text</a>	fontStyle	X	X	X	X	X
	length	X	X	X	X	X
	maxExtent	X	X	X	X	X
	metadata	X	X	X	X	X
	string	X	X	X	X	X
	lineBounds		X	X	X	X
	textBounds		X	X	X	X
solid	X	X	X	X	X	
<a href="#">TextureBackground</a>	set_bind	X	X	X	X	X
	groundAngle	X	X	X	X	X
	groundColor	X	X	X	X	X
	backTexture	X	X	X	X	X
	bottomTexture	X	X	X	X	X
	frontTexture	X	X	X	X	X
	leftTexture	X	X	X	X	X
	metadata	X	X	X	X	X
	rightTexture	X	X	X	X	X
	topTexture	X	X	X	X	X
	skyAngle	X	X	X	X	X
	skyColor	X	X	X	X	X
	transparency			X	X	X
	bindTime	X	X	X	X	X
isBound	X	X	X	X	X	
<a href="#">TextureCoordinate</a>	mapping					X
	metadata	X	X	X	X	X
	point	X	X	X	X	X
<a href="#">TextureCoordinate3D</a>	mapping					X
	metadata		X	X	X	X
	point		X	X	X	X

<a href="#">TextureCoordinate4D</a>	mapping					X
	metadata		X	X	X	X
	point		X	X	X	X
<a href="#">TextureCoordinateGenerator</a>	mapping					X
	metadata	X	X	X	X	X
	mode	X	X	X	X	X
	parameter	X	X	X	X	X
<a href="#">TextureProperties</a>	anisotropicDegree			X	X	X
	borderColor			X	X	X
	borderWidth			X	X	X
	boundaryModeS			X	X	X
	boundaryModeT			X	X	X
	boundaryModeR			X	X	X
	magnificationFilter			X	X	X
	metadata			X	X	X
	minificationFilter			X	X	X
	textureCompression			X	X	X
	texturePriority			X	X	X
	generateMipMaps			X	X	X
<a href="#">TextureTransform</a>	center	X	X	X	X	X
	mapping					X
	metadata	X	X	X	X	X
	rotation	X	X	X	X	X
	scale	X	X	X	X	X
	translation	X	X	X	X	X
<a href="#">TextureTransform3D</a>	center		X	X	X	X
	mapping					X
	metadata		X	X	X	X
	rotation		X	X	X	X
	scale		X	X	X	X
	translation		X	X	X	X
<a href="#">TextureTransformMatrix3D</a>	mapping					X
	metadata		X	X	X	X
	matrix		X	X	X	X
<a href="#">TimeSensor</a>	cycleInterval	X	X	X	X	X
	enabled	X	X	X	X	X
	loop	X	X	X	X	X
	metadata	X	X	X	X	X
	pauseTime	X	X	X	X	X
	resumeTime	X	X	X	X	X
	startTime	X	X	X	X	X

	stopTime	X	X	X	X	X
	cycleTime	X	X	X	X	X
	elapsedTime	X	X	X	X	X
	fraction_changed	X	X	X	X	X
	isActive	X	X	X	X	X
	isPaused	X	X	X	X	X
	time	X	X	X	X	X
<a href="#">TimeTrigger</a>	set_boolean	X	X	X	X	X
	metadata	X	X	X	X	X
	triggerTime	X	X	X	X	X
<a href="#">ToneMappedVolumeStyle</a>	coolColor				X	X
	enabled				X	X
	metadata				X	X
	surfaceNormals				X	X
	warmColor				X	X
<a href="#">TouchSensor</a>	description	X	X	X	X	X
	enabled	X	X	X	X	X
	metadata	X	X	X	X	X
	hitNormal_changed	X	X	X	X	X
	hitPoint_changed	X	X	X	X	X
	hitTexCoord_changed	X	X	X	X	X
	isActive	X	X	X	X	X
	isOver	X	X	X	X	X
	touchTime	X	X	X	X	X
<a href="#">Transform</a>	addChildren	X	X	X	X	X
	removeChildren	X	X	X	X	X
	center	X	X	X	X	X
	children	X	X	X	X	X
	metadata	X	X	X	X	X
	rotation	X	X	X	X	X
	scale	X	X	X	X	X
	scaleOrientation	X	X	X	X	X
	translation	X	X	X	X	X
	bboxCenter	X	X	X	X	X
	bboxSize	X	X	X	X	X
<a href="#">TransformSensor</a>	center			X	X	X
	enabled			X	X	X
	metadata			X	X	X
	size			X	X	X
	targetObject			X	X	X
	enterTime			X	X	X

	exitTime			X	X	X
	isActive			X	X	X
	orientation_changed			X	X	X
	position_changed			X	X	X
<a href="#">TransmitterPdu</a>	address	X	X	X	X	X
	antennaLocation	X	X	X	X	X
	antennaPatternLength	X	X	X	X	X
	antennaPatternType	X	X	X	X	X
	applicationID	X	X	X	X	X
	cryptoKeyID	X	X	X	X	X
	cryptoSystem	X	X	X	X	X
	enabled		X	X	X	X
	entityID	X	X	X	X	X
	frequency	X	X	X	X	X
	inputSource	X	X	X	X	X
	lengthOfModulationParameters	X	X	X	X	X
	metadata	X	X	X	X	X
	modulationTypeDetail	X	X	X	X	X
	modulationTypeMajor	X	X	X	X	X
	modulationTypeSpreadSpectrum	X	X	X	X	X
	modulationTypeSystem	X	X	X	X	X
	multicastRelayHost	X	X	X	X	X
	multicastRelayPort	X	X	X	X	X
	networkMode	X	X	X	X	X
	port	X	X	X	X	X
	power	X	X	X	X	X
	radioEntityTypeCategory	X	X	X	X	X
	radioEntityTypeCountry	X	X	X	X	X
	radioEntityTypeDomain	X	X	X	X	X
	radioEntityTypeKind	X	X	X	X	X
	radioEntityTypeNomenclature	X	X	X	X	X
	radioEntityTypeNomenclatureVersion	X	X	X	X	X
	radioID	X	X	X	X	X
	readInterval	X	X	X	X	X
	relativeAntennaLocation	X	X	X	X	X
	rtpHeaderExpected	X	X	X	X	X
	siteID	X	X	X	X	X
	transmitFrequencyBandwidth	X	X	X	X	X
transmitState	X	X	X	X	X	
whichGeometry	X	X	X	X	X	
writeInterval	X	X	X	X	X	

	isActive	X	X	X	X	X
	isNetworkReader	X	X	X	X	X
	isNetworkWriter	X	X	X	X	X
	isRtpHeaderHeard	X	X	X	X	X
	isStandAlone	X	X	X	X	X
	timestamp	X	X	X	X	X
	bboxCenter	X	X	X	X	X
	bboxSize	X	X	X	X	X
<a href="#">TriangleFanSet</a>	attrib		X	X	X	X
	color	X	X	X	X	X
	coord	X	X	X	X	X
	fanCount	X	X	X	X	X
	fogCoord		X	X	X	X
	metadata	X	X	X	X	X
	normal	X	X	X	X	X
	texCoord	X	X	X	X	X
	ccw	X	X	X	X	X
	colorPerVertex	X	X	X	X	X
	normalPerVertex	X	X	X	X	X
	solid	X	X	X	X	X
<a href="#">TriangleSet</a>	attrib		X	X	X	X
	color	X	X	X	X	X
	coord	X	X	X	X	X
	fogCoord		X	X	X	X
	metadata	X	X	X	X	X
	normal	X	X	X	X	X
	texCoord	X	X	X	X	X
	ccw	X	X	X	X	X
	colorPerVertex	X	X	X	X	X
	normalPerVertex	X	X	X	X	X
	solid	X	X	X	X	X
<a href="#">TriangleSet2D</a>	metadata	X	X	X	X	X
	vertices	X	X	X	X	X
	solid	X	X	X	X	X
<a href="#">TriangleStripSet</a>	attrib		X	X	X	X
	color	X	X	X	X	X
	coord	X	X	X	X	X
	fogCoord		X	X	X	X
	metadata	X	X	X	X	X
	normal	X	X	X	X	X
	stripCount	X	X	X	X	X

	texCoord	X	X	X	X	X
	ccw	X	X	X	X	X
	colorPerVertex	X	X	X	X	X
	normalPerVertex	X	X	X	X	X
	solid	X	X	X	X	X
<a href="#">TwoSidedMaterial</a>	ambientIntensity			X	X	X
	backAmbientIntensity			X	X	X
	backDiffuseColor			X	X	X
	backEmissiveColor			X	X	X
	backShininess			X	X	X
	backSpecularColor			X	X	X
	backTransparency			X	X	X
	diffuseColor			X	X	X
	emissiveColor			X	X	X
	metadata			X	X	X
	shininess			X	X	X
	separateBackColor			X	X	X
	specularColor			X	X	X
	transparency			X	X	X
<a href="#">UnlitMaterial</a>	emissiveColor					X
	emissiveTexture					X
	emissiveTextureMapping					X
	metadata					X
	normalTexture					X
	normalTextureMapping					X
	normalScale					X
	transparency					X
<a href="#">UniversalJoint</a>	anchorPoint			X	X	X
	axis1			X	X	X
	axis2			X	X	X
	body1			X	X	X
	body2			X	X	X
	metadata			X	X	X
	mustOutput			X	X	X
	stop1Bounce			X	X	X
	stop1ErrorCorrection			X	X	X
	stop2Bounce			X	X	X
	stop2ErrorCorrection			X	X	X
	body1AnchorPoint			X	X	X
	body1Axis			X	X	X
body2AnchorPoint			X	X	X	

	body2Axis			X	X	X
<a href="#">Viewpoint</a>	set_bind	X	X	X	X	X
	centerOfRotation	X	X	X	X	X
	description	X	X	X	X	X
	fieldOfView	X	X	X	X	X
	jump	X	X	X	X	X
	metadata	X	X	X	X	X
	orientation	X	X	X	X	X
	retainUserOffsets			X	X	X
	position	X	X	X	X	X
	bindTime	X	X	X	X	X
	isBound	X	X	X	X	X
<a href="#">ViewpointGroup</a>	center			X	X	X
	children			X	X	X
	description			X	X	X
	displayed			X	X	X
	metadata			X	X	X
	retainUserOffsets			X	X	X
	size			X	X	X
<a href="#">Viewport</a>	addChildren			X	X	X
	removeChildren			X	X	X
	children			X	X	X
	clipBoundary			X	X	X
	metadata			X	X	X
	bboxCenter			X	X	X
	bboxSize			X	X	X
<a href="#">VisibilitySensor</a>	center	X	X	X	X	X
	enabled	X	X	X	X	X
	metadata	X	X	X	X	X
	size	X	X	X	X	X
	enterTime	X	X	X	X	X
	exitTime	X	X	X	X	X
	isActive	X	X	X	X	X
<a href="#">VolumeData</a>	dimensions				X	X
	metadata				X	X
	renderStyle				X	X
	voxels				X	X
	bboxCenter				X	X
	bboxSize				X	X
	set_coordIndex			X	X	X
	coord			X	X	X



<a href="#">VolumeEmitter</a>	direction			X	X	X
	metadata			X	X	X
	speed			X	X	X
	variation			X	X	X
	coordIndex			X	X	X
	internal			X	X	X
	mass			X	X	X
	surfaceArea			X	X	X
<a href="#">VolumePickSensor</a>	enabled			X	X	X
	metadata			X	X	X
	objectType			X	X	X
	pickingGeometry			X	X	X
	pickTarget			X	X	X
	isActive			X	X	X
	pickedGeometry			X	X	X
	intersectionType			X	X	X
sortOrder			X	X	X	
<a href="#">WindPhysicsModel</a>	direction			X	X	X
	enabled			X	X	X
	gustiness			X	X	X
	metadata			X	X	X
	speed			X	X	X
	turbulence			X	X	X
<a href="#">WorldInfo</a>	metadata	X	X	X	X	X
	info	X	X	X	X	X
	title	X	X	X	X	X
<a href="#">X3DAppearanceChildNode</a>	metadata	X	X	X	X	X
<a href="#">X3DAppearanceNode</a>	metadata	X	X	X	X	X
<a href="#">X3DBackgroundNode</a>	set_bind	X	X	X	X	X
	groundAngle	X	X	X	X	X
	groundColor	X	X	X	X	X
	metadata	X	X	X	X	X
	skyAngle	X	X	X	X	X
	skyColor	X	X	X	X	X
	transparency			X	X	X
	bindTime	X	X	X	X	X
	isBound	X	X	X	X	X
	set_bind	X	X	X	X	X
	metadata	X	X	X	X	X

<a href="#"><i>X3DBindableNode</i></a>	bindTime	X	X	X	X	X
	isBound	X	X	X	X	X
<a href="#"><i>X3DBoundedObject</i></a>	bboxCenter	X	X	X	X	X
	bboxSize	X	X	X	X	X
<a href="#"><i>X3DChaserNode</i></a>	set_destination			X	X	X
	set_value			X	X	X
	metadata			X	X	X
	isActive			X	X	X
	value_changed			X	X	X
	duration			X	X	X
	initialDestination			X	X	X
	initialValue			X	X	X
<a href="#"><i>X3DChildNode</i></a>	metadata	X	X	X	X	X
<a href="#"><i>X3DColorNode</i></a>	metadata	X	X	X	X	X
<a href="#"><i>X3DComposableVolumeRenderStyleNode</i></a>	enabled				X	X
	metadata				X	X
<a href="#"><i>X3DComposedGeometryNode</i></a>	attrib		X	X	X	X
	color	X	X	X	X	X
	coord	X	X	X	X	X
	fogCoord		X	X	X	X
	metadata	X	X	X	X	X
	normal	X	X	X	X	X
	texCoord	X	X	X	X	X
	ccw	X	X	X	X	X
	colorPerVertex	X	X	X	X	X
	normalPerVertex	X	X	X	X	X
	solid	X	X	X	X	X
<a href="#"><i>X3DCoordinateNode</i></a>	metadata	X	X	X	X	X
<a href="#"><i>X3DDamperNode</i></a>	set_destination			X	X	X
	set_value			X	X	X
	metadata			X	X	X
	tau			X	X	X
	tolerance			X	X	X
	isActive			X	X	X
	value_changed			X	X	X
	initialDestination			X	X	X
	initialValue			X	X	X

	order			X	X	X
<a href="#">X3DDragSensorNode</a>	autoOffset	X	X	X	X	X
	description	X	X	X	X	X
	enabled	X	X	X	X	X
	metadata	X	X	X	X	X
	isActive	X	X	X	X	X
	isOver	X	X	X	X	X
	trackPoint_changed	X	X	X	X	X
<a href="#">X3DEnvironmentalSensorNode</a>	center	X	X	X	X	X
	enabled	X	X	X	X	X
	metadata	X	X	X	X	X
	size	X	X	X	X	X
	enterTime	X	X	X	X	X
	exitTime	X	X	X	X	X
	isActive	X	X	X	X	X
<a href="#">X3DEnvironmentTextureNode</a>	metadata		X	X	X	X
<a href="#">X3DFollowerNode</a>	set_destination			X	X	X
	set_value			X	X	X
	metadata			X	X	X
	isActive			X	X	X
	value_changed			X	X	X
	initialDestination			X	X	X
	initialValue			X	X	X
<a href="#">X3DFogObject</a>	color		X	X	X	X
	fogType		X	X	X	X
	visibilityRange		X	X	X	X
<a href="#">X3DFontStyleNode</a>	metadata	X	X	X	X	X
<a href="#">X3DGeometricPropertyNode</a>	metadata	X	X	X	X	X
<a href="#">X3DGeometryNode</a>	metadata	X	X	X	X	X
<a href="#">X3DGroupingNode</a>	addChildren	X	X	X	X	X
	removeChildren	X	X	X	X	X
	children	X	X	X	X	X
	metadata	X	X	X	X	X
	bboxCenter	X	X	X	X	X
	bboxSize	X	X	X	X	X
<a href="#">X3DInfoNode</a>	metadata	X	X	X	X	X

<a href="#"><i>X3DInterpolatorNode</i></a>	set_fraction	X	X	X	X	X
	key	X	X	X	X	X
	keyValue	X	X	X	X	X
	metadata	X	X	X	X	X
	value_changed	X	X	X	X	X
<a href="#"><i>X3DKeyDeviceSensorNode</i></a>	enabled	X	X	X	X	X
	metadata	X	X	X	X	X
	isActive	X	X	X	X	X
<a href="#"><i>X3DLayerNode</i></a>	pickable			X	X	X
	metadata			X	X	X
	viewport			X	X	X
<a href="#"><i>X3DLayoutNode</i></a>	metadata			X	X	X
<a href="#"><i>X3DLightNode</i></a>	ambientIntensity	X	X	X	X	X
	color	X	X	X	X	X
	global	X	X	X	X	X
	intensity	X	X	X	X	X
	metadata	X	X	X	X	X
	on	X	X	X	X	X
<a href="#"><i>X3DMaterialNode</i></a>	metadata	X	X	X	X	X
<a href="#"><i>X3DMetadataObject</i></a>	name	X	X	X	X	X
	reference	X	X	X	X	X
<a href="#"><i>X3DNBodyCollidableNode</i></a>	enabled			X	X	X
	metadata			X	X	X
	rotation			X	X	X
	translation			X	X	X
	bboxCenter			X	X	X
	bboxSize			X	X	X
<a href="#"><i>X3DNBodyCollisionSpaceNode</i></a>	enabled			X	X	X
	metadata			X	X	X
	bboxCenter			X	X	X
	bboxSize			X	X	X
<a href="#"><i>X3DNetworkSensorNode</i></a>	enabled	X	X	X	X	X
	metadata	X	X	X	X	X
	isActive	X	X	X	X	X
<a href="#"><i>X3DNode</i></a>	metadata	X	X	X	X	X
<a href="#"><i>X3DNormalNode</i></a>	metadata	X	X	X	X	X

<a href="#"><u>X3DNurbsControlCurveNode</u></a>	controlPoint	X	X	X	X	X
	metadata	X	X	X	X	X
<a href="#"><u>X3DNurbsSurfaceGeometryNode</u></a>	controlPoint	X	X	X	X	X
	metadata	X	X	X	X	X
	texCoord	X	X	X	X	X
	uTessellation	X	X	X	X	X
	vTessellation	X	X	X	X	X
	weight	X	X	X	X	X
	solid	X	X	X	X	X
	uClosed	X	X	X	X	X
	uDimension	X	X	X	X	X
	uKnot	X	X	X	X	X
	uOrder	X	X	X	X	X
	vClosed	X	X	X	X	X
	vDimension	X	X	X	X	X
	vKnot	X	X	X	X	X
vOrder	X	X	X	X	X	
<a href="#"><u>X3DOneSidedMaterialNode</u></a>	emissiveColor					X
	emissiveTexture					X
	emissiveTextureMapping					X
	metadata					X
	normalTexture					X
	normalTextureMapping					X
	normalScale					X
<a href="#"><u>X3DParametricGeometryNode</u></a>	metadata	X	X	X	X	X
<a href="#"><u>X3DParticleEmitterNode</u></a>	metadata			X	X	X
	speed			X	X	X
	variation			X	X	X
	mass			X	X	X
	surfaceArea			X	X	X
<a href="#"><u>X3DParticlePhysicsModelNode</u></a>	enabled			X	X	X
	metadata			X	X	X
<a href="#"><u>X3DPickableObject</u></a>	enabled			X	X	X
	metadata			X	X	X
<a href="#"><u>X3DPickSensorNode</u></a>	enabled			X	X	X
	metadata			X	X	X
	matchCriterion			X	X	X
	objectType			X	X	X
	pickingGeometry			X	X	X
	pickTarget			X	X	X

	pickedGeometry			X	X	X
	isActive			X	X	X
	intersectionType			X	X	X
	sortOrder			X	X	X
<a href="#">X3DPointingDeviceSensorNode</a>	description	X	X	X	X	X
	enabled	X	X	X	X	X
	metadata	X	X	X	X	X
	isActive	X	X	X	X	X
	isOver	X	X	X	X	X
<a href="#">X3DProductStructureChildNode</a>	metadata		X	X	X	X
	name		X	X	X	X
<a href="#">X3DProgrammableShaderObject</a>	none		X	X	X	X
<a href="#">X3DPrototypeInstance</a>	metadata	X	X	X	X	X
<a href="#">X3DRigidJointNode</a>	body1			X	X	X
	body2			X	X	X
	metadata			X	X	X
	mustOutput			X	X	X
<a href="#">X3DScriptNode</a>	metadata			X	X	X
<a href="#">X3DSensorNode</a>	enabled	X	X	X	X	X
	metadata	X	X	X	X	X
	isActive	X	X	X	X	X
<a href="#">X3DSequencerNode</a>	next	X	X	X	X	X
	previous	X	X	X	X	X
	set_fraction	X	X	X	X	X
	key	X	X	X	X	X
	keyValue	X	X	X	X	X
	metadata	X	X	X	X	X
	value_changed	X	X	X	X	X
<a href="#">X3DShaderNode</a>	activate		X	X	X	X
	metadata		X	X	X	X
	isSelected		X	X	X	X
	isValid		X	X	X	X
	language		X	X	X	X
<a href="#">X3DShapeNode</a>	appearance	X	X	X	X	X
	geometry	X	X	X	X	X
	metadata	X	X	X	X	X
	bboxCenter	X	X	X	X	X
	bboxSize	X	X	X	X	X

<a href="#">X3DSoundNode</a>	metadata	X	X	X	X	X
<a href="#">X3DSoundSourceNode</a>	description	X	X	X	X	X
	loop	X	X	X	X	X
	metadata	X	X	X	X	X
	pauseTime	X	X	X	X	X
	pitch	X	X	X	X	X
	resumeTime	X	X	X	X	X
	startTime	X	X	X	X	X
	stopTime	X	X	X	X	X
	duration_changed	X	X	X	X	X
	elapsedTime	X	X	X	X	X
	isActive	X	X	X	X	X
	isPaused	X	X	X	X	X
<a href="#">X3DSingleTextureCoordinateNode</a>	mapping					X
	metadata					X
<a href="#">X3DSingleTextureCoordinateNode</a>	metadata					X
<a href="#">X3DSingleTextureCoordinateNode</a>	mapping					X
	metadata					X
<a href="#">X3DTexture2DNode</a>	metadata	X	X	X	X	X
	repeatS	X	X	X	X	X
	repeatT	X	X	X	X	X
<a href="#">X3DTexture3DNode</a>	metadata		X	X	X	X
	repeatS		X	X	X	X
	repeatT		X	X	X	X
	repeatR		X	X	X	X
<a href="#">X3DTextureCoordinateNode</a>	metadata	X	X	X	X	X
<a href="#">X3DTextureNode</a>	metadata	X	X	X	X	X
<a href="#">X3DTextureTransformNode</a>	metadata	X	X	X	X	X
<a href="#">X3DTimeDependentNode</a>	loop	X	X	X	X	X
	metadata	X	X	X	X	X
	pauseTime	X	X	X	X	X
	resumeTime	X	X	X	X	X
	startTime	X	X	X	X	X
	stopTime	X	X	X	X	X
	elapsedTime	X	X	X	X	X
	isActive	X	X	X	X	X

	isPaused	X	X	X	X	X
<a href="#">X3DTouchSensorNode</a>	description	X	X	X	X	X
	enabled	X	X	X	X	X
	metadata	X	X	X	X	X
	isActive	X	X	X	X	X
	isOver	X	X	X	X	X
	touchTime	X	X	X	X	X
<a href="#">X3DTriggerNode</a>	metadata	X	X	X	X	X
<a href="#">X3DUriObject</a>	url	X	X	X	X	X
<a href="#">X3DVertexAttributeNode</a>	metadata		X	X	X	X
	name		X	X	X	X
<a href="#">X3DViewpointNode</a>	set_bind			X	X	X
	centerOfRotation			X	X	X
	description			X	X	X
	jump			X	X	X
	metadata			X	X	X
	orientation			X	X	X
	position			X	X	X
	retainUserOffsets			X	X	X
	bindTime			X	X	X
	isBound			X	X	X
<a href="#">X3DViewportNode</a>	addChildren			X	X	X
	removeChildren			X	X	X
	children			X	X	X
	metadata			X	X	X
	bboxCenter			X	X	X
	bboxSize			X	X	X
<a href="#">X3DVolumeDataNode</a>	dimensions				X	X
	metadata				X	X
	bboxCenter				X	X
	bboxSize				X	X
<a href="#">X3DVolumeRenderStyleNode</a>	enabled				X	X
	metadata				X	X

[Table Z.3](#) lists each statement specified by this part of ISO/IEC 19775. For each statement, the parameters supported by each version are identified listed in the order specified by the statement definition. Statements will appear in multiple rows if parameters have been added in subsequent versions.

**Table Z.3 — Version content (statements)**



Statement	Parameters	3.0	3.1	3.2	3.3	4.0
<a href="#">COMPONENT</a>	name	X	X	X	X	X
	level	X	X	X	X	X
<a href="#">EXTERNPROTO</a>	externprotoName	X	X	X	X	X
	externprotoInterfaceDeclaration	X	X	X	X	X
	externprotoURL	X	X	X	X	X
<a href="#">header</a>	standard	X	X	X	X	X
	version	X	X	X	X	X
	character encoding	X	X	X	X	X
	comments	X	X	X	X	X
<a href="#">META</a>	key	X	X	X	X	X
	data	X	X	X	X	X
<a href="#">PROFILE</a>	name	X	X	X	X	X
<a href="#">PROTO</a>	protoName	X	X	X	X	X
	protoInterfaceDeclaration	X	X	X	X	X
	protoDefinition	X	X	X	X	X
<a href="#">ROUTE</a>	fromNodeName	X	X	X	X	X
	fromFieldName	X	X	X	X	X
	toNodeName	X	X	X	X	X
	toFieldName	X	X	X	X	X
<a href="#">UNIT</a>	category				X	X
	name				X	X
	conversionFactor				X	X



## Extensible 3D (X3D) Part 1: Architecture and base components

### 14 Geometry2D component



#### 14.1 Introduction

##### 14.1.1 Name

The name of this component is "Geometry2D". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 14.1.2 Overview

This clause describes the Geometry2D component of this part of ISO/IEC 19775. This includes how two-dimensional geometry is specified and what shapes are available. [Table 14.1](#) provides links to the major topics in this clause.

**Table 14.1 — Topics**

- [14.1 Introduction](#)
  - [14.1.1 Name](#)
  - [14.1.2 Overview](#)
- [14.2 Concepts](#)
  - [14.2.1 Overview of geometry](#)
  - [14.2.2 Shape and geometry nodes](#)
  - [14.2.3 Geometric property nodes](#)
  - [14.2.4 Appearance nodes](#)
- [14.3 Node Reference](#)
  - [14.3.1 Arc2D](#)
  - [14.3.2 ArcClose2D](#)
  - [14.3.3 Circle2D](#)
  - [14.3.4 Disk2D](#)
  - [14.3.5 Polyline2D](#)
  - [14.3.6 Polypoint2D](#)
  - [14.3.7 Rectangle2D](#)
  - [14.3.8 TriangleSet2D](#)

#### 14.4 Support levels

- [Figure 14.1 — Arc2D node](#)
- [Figure 14.2 — ArcClose2D node \("PIE" closure\)](#)
- [Figure 14.2 — ArcClose2D node \("CHORD" closure\)](#)
- [Figure 13.4 — Circle2D node](#)
- [Figure 14.5 — Disk2D node](#)
- [Figure 14.6 — Polyline2D node](#)
- [Figure 14.7 — Polypoint2D node](#)
- [Figure 14.8 — Rectangle2D node](#)
- [Figure 14.9 — TriangleSet2D node](#)
  
- [Table 14.1 — Topics](#)
- [Table 14.2 — Geometry2D component support levels](#)

## 14.2 Concepts

### 14.2.1 Overview of geometry

The geometry2D component consists of only geometry nodes since it uses the shape, geometry property, and appearance nodes defined in the Shape component. The geometry2D nodes may be considered to be planar objects.

The two-dimensional coordinate system in which all 2D nodes are specified is defined to be the  $z=0$  plane of the current 3D coordinate system with  $x$ - and  $y$ -axes coincident with those of the current 3D coordinate system. The origin of the 2D coordinate system is defined to be the origin of the 3D coordinate system. The unspecified  $z$ -component of a 2D coordinate is defined to always have value zero. The position and orientation of 2D nodes are affected by all transformations whether 2D or 3D.

Each face in a 2D node is coplanar with the  $z=0$  plane of the coordinate system in which it is defined. Faces have both a front and a back face. The front face is defined to be that on the positive side of the  $z=0$  plane. Faces are subject to culling as defined elsewhere in this standard for 3D geometry.

When 2D nodes are viewed edge-on, they disappear as they have no depth.

### 14.2.2 Shape and geometry nodes

The [Shape](#) node is defined in [12 Shape component](#).

### 14.2.3 Geometric property nodes

Several geometry nodes contain geometric property nodes such as [Coordinate](#), [Color](#), [ColorRGBA](#), and/or [Normal](#). These nodes are specified in [11 Rendering component](#). The [X3DTextureCoordinate](#) nodes specified in [18 Texturing component](#) are also geometry property nodes.

### 14.2.4 Appearance nodes

[Shape](#) nodes may specify an [Appearance](#) node that describes the appearance properties (material and texture) to be applied to the Shape's geometry. Appearance is described in [12 Shape component](#)

The interaction between the appearance properties and properties specific to geometry nodes is described in [12 Shape component](#).

## 14.3 Node reference

### 14.3.1 Arc2D

```
Arc2D : X3DGeometryNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFFloat [] endAngle  $\pi/2$  [-2 $\pi$ ,2 $\pi$ ]
  SFFloat [] radius 1 (0, $\infty$ )
  SFFloat [] startAngle 0 [-2 $\pi$ ,2 $\pi$ ]
}
```

The Arc node specifies a linear circular arc whose center is at (0,0) and whose angles are measured starting at the positive x-axis and sweeping towards the positive y-axis. The *radius* field specifies the radius of the circle of which the arc is a portion. The arc extends from the *startAngle* counterclockwise to the *endAngle*. The values of *startAngle* and *endAngle* shall be in the range  $[-2\pi, 2\pi]$  radians (or the equivalent if a different angle base unit has been specified). If *startAngle* and *endAngle* have the same value, a circle is specified.

See [Figure 14.1](#) for a depiction of the Arc node.



Figure 14.1 — Arc2D node

### 14.3.2 ArcClose2D

```
ArcClose2D : X3DGeometryNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFString [] closureType "PIE" ["PIE"|"CHORD"]
  SFFloat [] endAngle  $\pi/2$  [-2 $\pi$ ,2 $\pi$ ]
  SFFloat [] radius 1 (0, $\infty$ )
  SFBool [] solid FALSE
  SFFloat [] startAngle 0 [-2 $\pi$ ,2 $\pi$ ]
}
```

The ArcClose node specifies a portion of a circle whose center is at (0,0) and whose angles are measured starting at the positive x-axis and sweeping towards the positive y-axis. The end points of the arc specified are connected as defined by the *closureType* field. The *radius* field specifies the radius of the circle of which the arc is a portion. The arc extends from the *startAngle* counterclockwise to the *endAngle*. The value of *radius* shall be greater than zero. The values of *startAngle* and *endAngle* shall be in the range  $[-2\pi, 2\pi]$  radians (or the equivalent if a different default angle base unit has been

specified). If *startAngle* and *endAngle* have the same value, a circle is specified and *closureType* is ignored. If the absolute difference between *startAngle* and *endAngle* is greater than or equal to  $2\pi$ , a complete circle is produced with no chord or radial line(s) drawn from the center.

A *closureType* of "PIE" connects the end point to the start point by defining two straight line segments first from the end point to the center and then the center to the start point. This forms a pie wedge as shown in [Figure 14.2](#).



**Figure 14.2 — ArcClose2D node ("PIE" closure)**

A *closureType* of "CHORD" connects the end point to the start point by defining a straight line segment from the end point to the start point. This forms an arc segment as shown in [Figure 14.3](#).



**Figure 14.3 — ArcClose2D node ("CHORD" closure)**

Textures are applied individually to each face of the ArcClose2D. On the front (+Z) and back (-Z) faces of the ArcClose2D, when viewed from the +Z-axis, the texture is mapped onto each face with the same orientation as if the image were displayed normally in 2D. TextureTransform affects the texture coordinates of the ArcClose2D (see [18.4.9 TextureTransform](#)).

[11.2.3 Common geometry fields](#) provides a complete description of the *solid* field.

### 14.3.3 Circle2D

```
Circle2D : X3DGeometryNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFFloat [] radius 1 (0,∞)
}
```

The Circle2D node specifies a circle centred at (0,0) in the local 2D coordinate system. The *radius* field specifies the radius of the Circle2D. The value of *radius* shall be greater than zero. [Figure 14.4](#) illustrates the Circle2D node with a dashed linetype applied.



Figure 14.4 — Circle2D node

### 14.3.4 Disk2D

```

Disk2D : X3DGeometryNode {
  SFNode [in,out] metadata  NULL [X3DMetadataObject]
  SFFloat []   innerRadius 0  (0,∞)
  SFFloat []   outerRadius 1  (0,∞)
  SFBool []    solid      FALSE
}

```

The Disk2D node specifies a circular disk which is centred at (0, 0) in the local coordinate system. The *outerRadius* field specifies the radius of the outer dimension of the Disk2D. The *innerRadius* field specifies the inner dimension of the Disk2D. The value of *outerRadius* shall be greater than zero. The value of *innerRadius* shall be greater than or equal to zero and less than or equal to *outerRadius*. If *innerRadius* is zero, the Disk2D is completely filled. Otherwise, the area within the *innerRadius* forms a hole in the Disk2D. If *innerRadius* is equal to *outerRadius*, a solid circular line shall be drawn using the current line properties. [Figure 14.5](#) illustrates the Disk2D node containing a non-zero *innerRadius*.



Figure 14.5 — Disk2D node

Textures are applied individually to each face of the Disk2D. On the front (+Z) and back (-Z) faces of the Disk2D, when viewed from the +Z-axis, the texture is mapped onto each face with the same orientation as if the image were displayed normally in 2D. [TextureTransform](#) affects the texture coordinates of Disk2D nodes (see [18.4.9 TextureTransform](#)).

[11.2.3 Common geometry fields](#) provides a complete description of the *solid* field.

### 14.3.5 Polyline2D

```

Polyline2D : X3DGeometryNode {
  SFNode [in,out] metadata  NULL [X3DMetadataObject]
  MFVec2f []   lineSegments [] (-∞,∞)
}

```

The Polyline2D node specifies a series of contiguous line segments in the local 2D coordinate system connecting the specified vertices. The *lineSegments* field specifies the vertices to be connected. [Figure 14.6](#) illustrates the Polyline2D node.

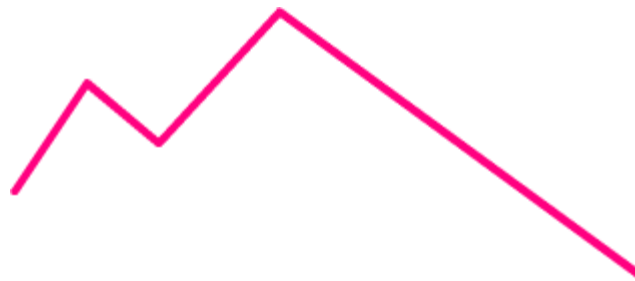


Figure 14.6 — Polyline2D node

### 14.3.6 Polypoint2D

```
Polypoint2D : X3DGeometryNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFVec2f [in,out] point [] (-∞,∞)
}
```

The Polyline2D node specifies a set of vertices in the local 2D coordinate system at each of which is displayed a point. The *points* field specifies the vertices to be displayed. [Figure 14.7](#) illustrates the Polypoint2D node by depicting the line in [Figure 14.6](#) (with points augmented for illustrative purposes).



Figure 14.7 — Polypoint2D node

### 14.3.7 Rectangle2D

```
Rectangle2D : X3DGeometryNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFVec2f [] size 2 2 (0,∞)
  SFBool [] solid FALSE
}
```

The Rectangle2D node specifies a rectangle centred at (0, 0) in the current local 2D coordinate system and aligned with the local coordinate axes. By default, the box measures 2 units in each dimension, from -1 to +1. The *size* field specifies the extents of the box along the X-, and Y-axes respectively and each component value shall be greater than zero. [Figure 14.8](#) illustrates the Rectangle2D node with a [FillProperties](#) node defining a hatch style.



**Figure 14.8 — Rectangle2D node**

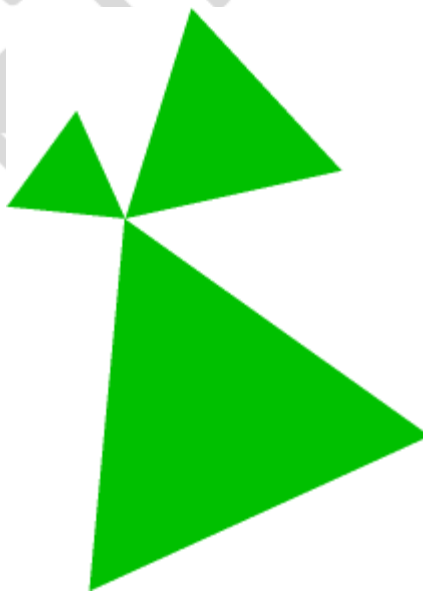
Textures are applied individually to each face of the Rectangle2D. On the front (+Z) and back (-Z) faces of the Rectangle2D, when viewed from the +Z-axis, the texture is mapped onto each face with the same orientation as if the image were displayed normally in 2D. [TextureTransform](#) affects the texture coordinates of the Rectangle2D (see [18.4.9 TextureTransform](#)).

[11.2.3 Common geometry fields](#) provides a complete description of the *solid* field.

### 14.3.8 TriangleSet2D

```
TriangleSet2D : X3DGeometryNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFVec2f [in,out] vertices [] (-∞,∞)
  SFBool [] solid FALSE
}
```

The TriangleSet2D node specifies a set of triangles in the local 2D coordinate system. The *vertices* field specifies the triangles to be displayed. The number of vertices provided shall be evenly divisible by three. Excess vertices shall be ignored. [Figure 14.9](#) illustrates the TriangleSet2D node.



**Figure 14.9 — TriangleSet2D node**



[11.2.3 Common geometry fields](#) provides a complete description of the *solid* field.

Textures are applied individually to each face of the TriangleSet2D. On the front (+Z) and back (-Z) faces of the TriangleSet2D, when viewed from the +Z-axis, the texture is mapped onto each face with the same orientation as if the image were displayed normally in 2D. [TextureTransform](#) affects the texture coordinates of the TriangleSet2D (see [18.4.9 TextureTransform](#)).

## 14.4 Support levels

The Geometry2D component provides two levels of support as specified in [Table 14.2](#). Level 1 provides the basic support for two-dimensional geometry with straight sides. Level 2 adds support for two-dimensional geometry with non-straight sides.

**Table 14.2 — Geometry2D component support levels**

Level	Prerequisites	Nodes/Features	Support
1	Core 1 Grouping 1 Shape 1 Rendering 1		
		Polyline2D	All fields fully supported.
		Polypoint2D	All fields fully supported.
		Rectangle2D	All fields fully supported.
		TriangleSet2D	All fields fully supported.
2	Core 1 Grouping 1 Shape 1 Rendering 1		
		All Level 1 Geometry2D nodes	All fields fully supported.
		Arc2D	All fields fully supported.
		ArcClose2D	All fields fully supported.
		Circle2D	All fields fully supported.
			All fields fully

		Disk2D	supported.
--	--	--------	------------



draft



## Extensible 3D (X3D) Part 1: Architecture and base components

### 35 Layering component

---



#### 35.1 Introduction

##### 35.1.1 Name

The name of this component is "Layering". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 35.1.2 Overview

This subclause describes the Layering component of this International Standard. This includes how to layer a set of subscene layers into a composite scene. [Table 35.1](#) provides links to the major topics in this subclause.

**Table 35.1 — Topics**

- [35.1 Introduction](#)
  - [35.1.1 Name](#)
  - [35.1.2 Overview](#)
- [35.2 Concepts](#)
  - [35.2.1 Overview of layering](#)
  - [35.2.2 Layer sets](#)
  - [35.2.3 Layers](#)
  - [35.2.4 Viewports](#)
- [35.3 Abstract types](#)
  - [35.3.1 X3DLayerNode](#)
  - [35.3.2 X3DViewportNode](#)
- [35.4 Node reference](#)
  - [35.4.1 Layer](#)
  - [35.4.2 LayerSet](#)
  - [35.4.3 Viewport](#)
- [35.5 Support levels](#)
- [Table 35.1 — Topics](#)

- [Table 35.2 — Layering component support levels](#)

## 35.2 Concepts

### 35.2.1 Overview of layering

A scene is embodied by the basic concept of layering. A scene is defined to consist of a sequence of layers and the order in which they are to be rendered.

### 35.2.2 Layer sets

A layer set is defined to be an ordered list of [X3DLayerNode](#) nodes that form a scene. The layers are assigned ordinals according to their position in the list in the [LayerSet](#) node. The rendering order is specified by the order field. Thus, the layer first specified in the order field will be the first layer rendered and will appear to be below any other layers. The layer last specified in the order field will be the last layer rendered and will correspondingly appear to be on top of all other layers.

The LayerSet node may make access to some of its content public by using the EXPORT statement to identify public names.

Only one LayerSet node is allowed and shall be a root node.

### 35.2.3 Layers

Each subscene is specified by a single [X3DLayerNode](#) node that contains its definition. The [X3DLayerNode](#) nodes may contain any child nodes allowed in grouping nodes. Hence, [X3DLayerNode](#) nodes may be used to create special effects such as heads up displays or non-transforming control elements. Each [X3DLayerNode](#) node contains its own binding stacks and thus has its own viewpoints and navigation.

### 35.2.4 Viewports

The output to a surface can be constrained further by using an [X3DViewportNode](#) node. These nodes are special grouping nodes that each define a set of clipping bounds within the extent of a surface within which the children nodes of the [X3DViewportNode](#) will appear. This provides support for the typical front/side/back/oblique views used by CAD systems.

## 35.3 Abstract types

### 35.3.1 X3DLayerNode

```
X3DLayerNode : X3DNode, X3DPickableObject {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFString [in,out] objectType "ALL" ["ALL","NONE","TERRAIN",...]
  SFBool [in,out] pickable TRUE
  SFNode [in,out] viewport NULL [X3DViewportNode]
  SFBool [in,out] visible TRUE
}
```

The *X3DLayerNode* abstract node type is the base node type for layer nodes.

The *pickable* field determines if pick traversal is to be performed for this layer. An *X3DLayerNode* node specified with *pickable* set to `FALSE` will not participate in picking operations.

The *viewport* field constrains the output of the layer to a sub-region of the render surface.

The *visible* field specifies whether or not the content within a node is visually displayed. The value of this field has no effect on animation behaviors, collision behaviors, event passing, or other non-visual characteristics.

### 35.3.2 X3DViewportNode

```
X3DViewportNode : X3DGroupingNode {
  MFNode [in]  addChildren      [X3DChildNode]
  MFNode [in]  removeChildren  [X3DChildNode]
  MFNode [in,out] children    [] [X3DChildNode]
  SFBool [in,out] bboxDisplay  FALSE
  SFNode [in,out] metadata    NULL [X3DMetadataObject]
  SFBool [in,out] visible     TRUE
  SFVec3f []  bboxCenter      0 0 0 (-∞,∞)
  SFVec3f []  bboxSize       -1 -1 -1 (0,∞) or -1 -1 -1
}
```

The *X3DViewportNode* abstract node type is the base node type for viewport nodes. Nodes of this type specify a boundary to which all content affected by the node is to be clipped. The boundary is specified in units appropriate for the surface on which the content is to be rendered.

More details on the children, addChildren, and removeChildren fields can be found in [10.2 Concepts](#).

The *bboxCenter* and *bboxSize* fields specify a bounding box that encloses the children. This is a hint that may be used for optimization purposes. The results are undefined if the specified bounding box is smaller than the actual bounding box of the children at any time. The default *bboxSize* value, (-1, -1, -1), implies that the bounding box is not specified and, if needed, shall be calculated by the browser. More details on the *bboxCenter* and *bboxSize* fields can be found in [10.2.2 Bounding boxes](#).

## 35.4 Node Reference

### 35.4.1 Layer

```
Layer : X3DLayerNode {
  MFNode [in]  addChildren      [X3DChildNode]
  MFNode [in]  removeChildren  [X3DChildNode]
  MFNode [in,out] children    [] [X3DChildNode]
  SFNode [in,out] metadata    NULL [X3DMetadataObject]
  MFString [in,out] objectTvp "ALL" ["ALL", "NONE", "TERRAIN", ...]
  SFBool [in,out] pickable    TRUE
  SFNode [in,out] viewport    NULL [X3DViewportNode]
  SFBool [in,out] visible     TRUE
}
```

The Layer node specifies a *children* field that contains a list of nodes that define the contents of the layer.

[10.2.1 Grouping and children node types](#) provides a description of the *children*, *addChildren*, and *removeChildren* fields.

## 35.4.2 LayerSet

```

LayerSet : X3DNode {
  SFInt32 [in,out] activeLayer 0 f(0,∞)
  MFNode [in,out] layers [] [X3DLayerNode]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFInt32 [in,out] order [0] f(0,∞)
}

```

The LayerSet node specifies a list of layers and a rendering order.

The *activeLayer* field specifies the layer in which navigation takes place.

The list defined by *layers* contains the constituent parts of the scene. Each layer is assigned an ordinal number depending on its position in the list. Ordinals start with the numeral 1 representing the first item in the list.

The list defined by *order* specifies the order in which the layers are rendered. The number specified correspond to the ordinals of the layers. *Order* may contain repetitions of the ordinals in which case the layer is rendered again. If *order* contains numbers that are not ordinals assigned to layers, such numbers are ignored. Layers included in *layers* that are not listed in *order* are not rendered.

Object picking according to the *pickable* field of a Layer node occurs even if that Layer is not visible.

Nodes that are not part of a layer are considered to be the first nodes in layer 0.

## 35.4.3 Viewport

```

Viewport : X3DViewportNode, X3DBoundedObject {
  MFNode [in] addChildren [X3DChildNode]
  MFNode [in] removeChildren [X3DChildNode]
  MFNode [in,out] children [] [X3DChildNode]
  MFFloat [in,out] clipBoundary 0 1 0 1 [0,1]
  SFBool [in out] bboxDisplay FALSE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFBool [in out] visible TRUE
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 (0,∞) or -1 -1 -1
}

```

The Viewport node is a grouping node that specifies a set of rectangular clip boundaries against which the children nodes are clipped as they are rendered.

The *clipBoundary* field is specified in fractions of the normal render surface in the sequence left/right/bottom/top. When the children are rendered, the output will only appear in the specified subset of the render surface.

## 35.5 Support levels

The Layering component provides four levels of support as specified in [Table 35.2](#). Level 1 provides the support for scenes and layers.

**Table 35.2 — Layering component support levels**

Level	Prerequisites	Nodes	Support

1	Core 1; Grouping 1		
		<i>X3DLayerNode</i>	n/a
		<i>X3DViewportNode</i>	n/a
		Layer	All fields fully supported.
		LayerSet	All fields fully supported.
		Viewport	All fields fully supported.





## Extensible 3D (X3D) Part 1: Architecture and base components

### Bibliography



This annex contains the informative references in this part of ISO/IEC 19775. These are references to unofficial standards or documents. All official standards are referenced in [2 Normative references](#).

Identifier	Reference
BLINN	James F. Blinn, <a href="#">Models of light reflection for computer synthesized pictures</a> . Proc. 4th annual conference on computer graphics and interactive techniques: 192. <a href="http://dx.doi.org/10.1145/563858.563893">http://dx.doi.org/10.1145/563858.563893</a> .
CATROM	Catmull, E. and Rom, R., <i>A class of local interpolating splines</i> in Computer-Aided Geometric Design" by Barnhill, R.E and Reisenfeld, R. F., New York Press 1974.
Cg	<i>nVidia Cg Shading Language Specification</i> <a href="https://developer.download.nvidia.com/cg/Cg_1.5/1.5.0/0019/Cg_Specification.pdf">https://developer.download.nvidia.com/cg/Cg_1.5/1.5.0/0019/Cg_Specification.pdf</a>
COM	<i>Component Object Model (General)</i> , Microsoft Developer Network Library Component Development <a href="http://msdn.microsoft.com/library">http://msdn.microsoft.com/library</a>
DDS	DDS File Reference, Microsoft Software Developer Network, 2004. <a href="http://msdn.microsoft.com/en-us/library/windows/desktop/bb943992%28v=vs.85%29.aspx">http://msdn.microsoft.com/en-us/library/windows/desktop/bb943992%28v=vs.85%29.aspx</a>
DICOM	The DICOM Standard, Digital Imaging and Communications in Medicine, Rosslyn, VA, 2003. <a href="http://medical.nema.org">http://medical.nema.org</a>
EBERT	D.Ebert and P Rheingans, <i>Volume Illustration: Non Photorealistic Rendering of Volume Model</i> , Proceedings of IEEE Visualization '00, (San Francisco, California 2000), 195-202.
ENGEL	K. Engel, et. al., <i>Real-Time Volume Graphics</i> , A. K. Peters. <a href="http://www.real-time-volume-graphics.org">http://www.real-time-volume-graphics.org</a>
FOLEY	Foley, van Dam, Feiner and Hughes, <i>Computer Graphics Principles and Practice, 2nd Edition</i> , Addison Wesley, Reading, MA, 1990. <a href="http://www.aw-bc.com">http://www.aw-bc.com</a>
FX	<i>Effect Reference</i> , Microsoft Software Developer Network, 2004. <a href="http://msdn.microsoft.com/en-us/library/windows/desktop/bb219839%28v=vs.85%29.aspx">http://msdn.microsoft.com/en-us/library/windows/desktop/bb219839%28v=vs.85%29.aspx</a>
GOOCH1	Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen. 1998. A non-photorealistic lighting model for automatic technical illustration. In Proceedings of the 25th annual conference on Computer graphics and interactive techniques (SIGGRAPH '98). ACM, New York, NY, USA, 447-452. <a href="http://doi.acm.org/10.1145/280814.280950">http://doi.acm.org/10.1145/280814.280950</a>
GOOCH2	Bruce Gooch and Amy Gooch, <i>Non-photorealistic rendering</i> , A K Peters, Ltd., Natick, MA, 2001.
GIF	"GIF™" — <i>Graphics Interchange Format™</i> — A standard defining a mechanism for the storage and transmission of raster-based graphics information, Version 89a, CompuServe. <a href="http://www.w3.org/Graphics/GIF/spec-gif89a.txt">http://www.w3.org/Graphics/GIF/spec-gif89a.txt</a>



<b>GLSL</b>	<a href="#">The OpenGL Shading Language Language Version 1.10 Document Revision 59</a> , Silicon Graphics, Inc. 2004
<b>HENYEY</b>	L. Henyey and J. Greenstein, <i>Diffuse radiation in the galaxy</i> , Astrophysics Journal, Vol 93, 1941.
<b>HLSL</b>	<i>Microsoft High Level Shading Language Specification for DirectX 9.0</i> <a href="http://msdn.microsoft.com/en-us/library/windows/desktop/bb509638%28v=vs.85%29.aspx">http://msdn.microsoft.com/en-us/library/windows/desktop/bb509638%28v=vs.85%29.aspx</a>
<b>JAPI</b>	<i>The Java™ Application Programming Interface, Volume 1 Core Packages</i> by James Gosling, Frank Yellin and The Java Team, Addison Wesley, Reading Massachusetts, 1996, ISBN 0-201-63453-8. <i>The Java™ Application Programming Interface, Volume 2 Window Toolkit and Applets</i> by James Gosling, Frank Yellin and The Java Team, Addison Wesley, Reading Massachusetts, 1996, ISBN 0-201-63459-7.
<b>LMIP</b>	Sato et. al., <i>Local Maximum Intensity Projection: A new rendering method for vascular visualisation</i> , Journal of Computer Assisted Tomography, Vol 22, No 6, 1998, 2005 <a href="http://citeseer.ist.psu.edu/31456.html">http://citeseer.ist.psu.edu/31456.html</a>
<b>NRRD</b>	<i>Definition of NRRD File Format.</i> <a href="http://teem.sourceforge.net/nrrd/format.html">http://teem.sourceforge.net/nrrd/format.html</a>
<b>NURBS</b>	Piegl, Les and Tiller, Wayne; <i>The NURBS Book, 2nd Edition</i> , Springer-Verlag (Berlin), 1997, ISBN: 3-540-61545-8.
<b>OPENGL</b>	<a href="#">The OpenGL Graphics System: A Specification (Version 2.0 - October 22, 2004)</a> , Silicon Graphics, Inc., 2004.
<b>PERL</b>	<i>Programming Perl, 4th Edition</i> by Tom Christiansen, Brian D. Foy, Larry Wall, and John Orwant, O'Reilly Media, Sebastapol, CA, 2012. <a href="http://www.oreilly.com">http://www.oreilly.com</a>
<b>PHONG</b>	B. T. Phong, <i>Illumination for computer generated pictures</i> , Communications of ACM 18 (1975), no. 6, 311–317.
<b>SHOE</b>	Shoemake, Ken, <i>Animating Rotations with Quaternion Calculus</i> , ACM SIGGRAPH 1987, Course Notes 10.
<b>SNDA</b>	<i>Fundamentals of Computer Music</i> , Dodge & Jerse, Shirmer Books, New York, 1985, pp 20-21.
<b>SNDB</b>	<i>Spatial Audio Work in the Multimedia Computing Group</i> , Graphics, Visualization, and Usability Center, Georgia Institute of Technology, Atlanta, GA. <a href="http://apple2.org.za/gswv/a2zine/GS.WorldView/Resources/MISC/Hightech.Sound/Spatial.Audio.Work.html">http://apple2.org.za/gswv/a2zine/GS.WorldView/Resources/MISC/Hightech.Sound/Spatial.Audio.Work.html</a>
<b>SNY87</b>	<i>MAP Projections - A Working Manual</i> by J. P. Snyder. . U.S. Geological Survey Professional Paper 1395. U.S. Government Printing Office, Washington, DC, 1987.
<b>UDP</b>	<a href="#">IETF RFC 768</a> , <i>User Datagram Protocol, Internet standards track protocol.</i>
<b>URI</b>	<a href="#">IETF RFC 1630</a> , <i>Universal Resource Identifiers in WWW.</i>
<b>VOL</b>	Brooks, Paul, <i>Volume data format</i> , 2000. <a href="http://paulbourke.net/dataformats/volumetric">http://paulbourke.net/dataformats/volumetric</a>
<b>WAV</b>	<i>Waveform Audio File Format, Multimedia Programming Interface and Data Specification v1.0</i> , Issued by IBM & Microsoft, 1991. <a href="http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/Docs/RIFFNEW.pdf">http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/Docs/RIFFNEW.pdf</a>



## Extensible 3D (X3D) Part 1: Architecture and base components

### 15 Text component



#### 15.1 Introduction

##### 15.1.1 Name

The name of this component is "Text". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 15.1.2 Overview

This clause describes the Text component of this part of ISO/IEC 19775. [Table 15.1](#) provides links to the major topics in this clause.

**Table 15.1 — Topics**

- [15.1 Introduction](#)
  - [15.1.1 Name](#)
  - [15.1.2 Overview](#)
- [15.2 Concepts](#)
  - [15.2.1 Text semantics](#)
    - [15.2.1.1 Overview](#)
    - [15.2.1.2 Appearance](#)
  - [15.2.2 Text formatting](#)
    - [15.2.2.1 Introduction](#)
    - [15.2.2.2 Font family and style](#)
    - [15.2.2.3 Direction and justification](#)
    - [15.2.2.4 Language](#)
- [15.3 Abstract types](#)
  - [15.3.1 X3DFontStyleNode](#)
- [15.4 Node reference](#)
  - [15.4.1 FontStyle](#)
  - [15.4.2 Text](#)
- [15.5 Support levels](#)

- [Figure 15.1 — Key for Tables 15.5 and 15.6](#)
- [Figure 15.2 — Text size and spacing fields](#)
- [Figure 15.3 — \*lineBounds\* and \*textBounds\* measurements](#)
- [Table 15.1 — Topics](#)
- [Table 15.2 — Major Alignment, horizontal = TRUE](#)
- [Table 15.3 — Major Alignment, horizontal = FALSE](#)
- [Table 15.4 — Minor Alignment, horizontal = TRUE](#)
- [Table 15.5 — Minor Alignment, horizontal = FALSE](#)
- [Table 15.6 — horizontal = TRUE](#)
- [Table 15.7 — horizontal = FALSE](#)
- [Table 15.8 — Text component support levels](#)

## 15.2 Concepts

### 15.2.1 Text semantics

#### 15.2.1.1 Overview

Text is processed as geometry in X3D. There are special considerations when specifying text as well as when displaying text. This subclause describes the manner in which text values are specified in X3D using the [Text](#) node. [15.2.2 Text formatting](#) describes text formatting.

#### 15.2.1.2 Appearance

Textures are applied to text as follows. The texture origin is at the origin of the first string, as determined by the justification. The texture is scaled equally in both S and T dimensions, with the font height representing 1 unit. S increases to the right, and T increases up.

[12 Shape component](#) specifies how [Appearance](#), material and textures interact with lighting. [17 Lighting component](#) specifies the X3D lighting equations.

### 15.2.2 Text formatting

#### 15.2.2.1 Introduction

There is a long history of text layout and formatting. This standard specifies techniques to be used in X3D that provide support for a variety of languages and layout schemes. Additional layout functionality is specified in [36 Layout component](#).

#### 15.2.2.2 Font family and style

Font attributes are defined with the *family* and *style* fields. The browser shall map the specified font attributes to an appropriate available font as described below.

The *family* field contains a case-sensitive MFString value that specifies a sequence of

font family names in preference order. The browser shall search the MFString value for the first font family name matching a supported font family. If none of the string values matches a supported font family, the default font family "SERIF" shall be used. All browsers shall support at least "SERIF" (the default) for a serif font such as Times Roman; "SANS" for a sans-serif font such as Helvetica; and "TYPEWRITER" for a fixed-pitch font such as Courier. An empty *family* value is identical to ["SERIF"]. Any font family may be specified as shown in the following example of the specification of a font family:

```
[ "Lucida Sans Typewriter", "Lucida Sans", "Helvetica", "SANS" ]
```

In this example, the browser would first look for the font family "Lucida Sans Typewriter" on the system on which the browser is operating. If that is not available, the browser looks for "Lucida Sans". If that is not available, the browser looks for "Helvetica". If that is not available, the browser looks for any sans-serif font. If there are not sans-serif fonts installed, the browser will use any serif font (the default). It is the responsibility of the author that a suitable list of font families be specified so that the desired appearance is achieved in most operating environments. However, the author should always be willing to accept that the requested font families may not be available resulting in the use of a browser-selected "SERIF" font being used.

The *style* field specifies a case-sensitive SFString value that may be "PLAIN" (the default) for default plain type; "BOLD" for boldface type; "ITALIC" for italic type; or "BOLDITALIC" for bold and italic type. An empty *style* value ("" ) is identical to "PLAIN". In the case where the requested style is not available, the available style that is closest to the requested style shall be used. For example, some font families specify a Demibold style rather than Bold. In this case, specifying "BOLD" will result in the browser using Demibold as the nearest substitute.

### 15.2.2.3 Direction and justification

The *horizontal*, *leftToRight*, and *topToBottom* fields indicate the direction of the text. The *horizontal* field indicates whether the text advances horizontally in its major direction (*horizontal* = TRUE, the default) or vertically in its major direction (*horizontal* = FALSE). The *leftToRight* and *topToBottom* fields indicate direction of text advance in the major (characters within a single string) and minor (successive strings) axes of layout. Which field is used for the major direction and which is used for the minor direction is determined by the *horizontal* field. Note that the direction specification overrides any modes inherent in a particular language.

For horizontal text (*horizontal* = TRUE), characters on each line of text advance in the positive X direction if *leftToRight* is TRUE or in the negative X direction if *leftToRight* is FALSE. Characters are advanced according to their natural advance width. Each line of characters is advanced in the negative Y direction if *topToBottom* is TRUE or in the positive Y direction if *topToBottom* is FALSE. Lines are advanced by the amount of *size* × *spacing*.

For vertical text (*horizontal* = FALSE), characters on each line of text advance in the negative Y direction if *topToBottom* is TRUE or in the positive Y direction if *topToBottom* is FALSE. Characters are advanced according to their natural advance height. Each line of characters is advanced in the positive X direction if *leftToRight* is TRUE or in the negative X direction if *leftToRight* is FALSE. Lines are advanced by the amount of *size* × *spacing*.

The *justify* field determines alignment of the above text layout relative to the origin of the object coordinate system. The *justify* field is an MFString which can contain 2 values. The first value specifies alignment along the major axis and the second value specifies alignment along the minor axis, as determined by the *horizontal* field. An empty *justify* value ("") is equivalent to the default value. If the second string, minor alignment, is not specified, minor alignment defaults to the value "FIRST". Thus, *justify* values of "", "BEGIN", and ["BEGIN" "FIRST"] are equivalent.

The major alignment is along the X-axis when *horizontal* is `TRUE` and along the Y-axis when *horizontal* is `FALSE`. The minor alignment is along the Y-axis when *horizontal* is `TRUE` and along the X-axis when *horizontal* is `FALSE`. The possible values for each enumerant of the *justify* field are "FIRST", "BEGIN", "MIDDLE", and "END". For major alignment, each line of text is positioned individually according to the major alignment enumerant. For minor alignment, the block of text representing all lines together is positioned according to the minor alignment enumerant. [Tables 15.2-15.5](#) describe the behaviour in terms of which portion of the text is at the origin.

**Table 15.2 — Major Alignment, *horizontal* = `TRUE`**

<i>justify</i> Enumerant	<i>leftToRight</i> = <code>TRUE</code>	<i>leftToRight</i> = <code>FALSE</code>
FIRST	Left edge of each line	Right edge of each line
BEGIN	Left edge of each line	Right edge of each line
MIDDLE	Centred about X-axis	Centred about X-axis
END	Right edge of each line	Left edge of each line

**Table 15.3 — Major Alignment, *horizontal* = `FALSE`**

<i>justify</i> Enumerant	<i>topToBottom</i> = <code>TRUE</code>	<i>topToBottom</i> = <code>FALSE</code>
FIRST	Top edge of each line	Bottom edge of each line
BEGIN	Top edge of each line	Bottom edge of each line
MIDDLE	Centred about Y-axis	Centre about Y-axis
END	Bottom edge of each line	Top edge of each line

**Table 15.4 — Minor Alignment, *horizontal* = `TRUE`**

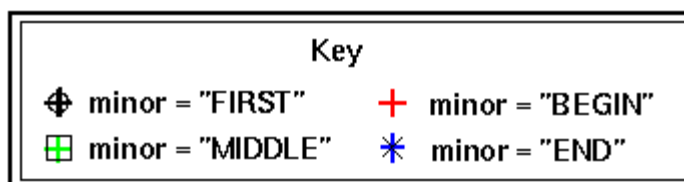
<i>justify</i> Enumerant	<i>topToBottom</i> = <code>TRUE</code>	<i>topToBottom</i> = <code>FALSE</code>
FIRST	Baseline of first line	Baseline of first line
BEGIN	Top edge of first line	Bottom edge of first line
MIDDLE	Centred about Y-axis	Centred about Y-axis

END	Bottom edge of last line	Top edge of last line
-----	--------------------------	-----------------------

**Table 15.5 — Minor Alignment, *horizontal* = FALSE**

<i>justify</i> Enumerant	<i>leftToRight</i> = TRUE	<i>leftToRight</i> = FALSE
FIRST	Left edge of first line	Right edge of first line
BEGIN	Left edge of first line	Right edge of first line
MIDDLE	Centred about X-axis	Centred about X-axis
END	Right edge of last line	Left edge of last line

The default minor alignment is "FIRST". This is a special case of minor alignment when *horizontal* is TRUE. Text starts at the baseline at the Y-axis. In all other cases, "FIRST" is identical to "BEGIN". In [Tables 15.6 and 15.7](#), each colour-coded cross-hair indicates where the X-axis and Y-axis shall be in relation to the text. [Figure 15.1](#) describes the symbols used in [Tables 15.6 and Table 15.7](#).



**Figure 15.1 — Key for Tables 15.6 and 15.7**

**Table 15.6 — *horizontal* = TRUE**

		major = "BEGIN" or "FIRST"		major = "MIDDLE"		major = "END"	
		leftToRight		leftToRight		leftToRight	
		TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
topToBottom	TRUE						
	FALSE						

Note: The "FIRST" minor axis marker ⊕ is offset from the "BEGIN" minor axis marker + in cases that they are coincident for presentation purposes only.

**Table 15.7 — *horizontal* = FALSE**

		major = "BEGIN" or "FIRST"		major = "MIDDLE"		major = "END"	
		leftToRight		leftToRight		leftToRight	
		TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
topToBottom	TRUE	⊕+ R s t * e o o a m d d e a y t ! e e x x t t	* t s R ⊕+ o o e d m a d a e d y ! t e e x x t t	s o m t R e e t ⊕+ a d * e e t e x x t t	s o m e R t o a e R ⊕+ d a * y ! t e x x t t	s o m e t o d t R R t d e e a a x y ⊕+ d t ! * t t	s o m t e o d t R d t R a e e y x a ⊕+ t d ⊕+ t t
	FALSE	t x e t ! y d e a a m d e o o ⊕+ R s t * t t	t x e ! t y a e d d m a o o e * t s R ⊕+ t t	t x e ! d e t y ⊕+ a e * e e R e m o s	t x e d ! y e t d ⊕+ a d * d e e R o t	⊕+ d t ! * a x y e e a R t d e t m o s	* ! t d ⊕+ y x a a e e d t R o t e m o s

Note: In every case, the "FIRST" minor axis marker ⊕ is coincident with the "BEGIN" minor axis marker + (and is offset for presentation purposes only).

### 15.2.2.4 Language

The *language* field specifies the context of the language for the text string in the form of a language and a country in which that language is used. Both the language and the country are specified using the language tags defined in [2.\[RFC3066\]](#) which may specify only a country (using the three-character codes defined in [ISO 3166](#)) or both a language (using the two-character codes specified in [ISO 639](#)) and a country (using the three-character codes specified in [ISO 3166](#)) utilizing a sub-tag structure as specified in [2.\[RFC3066\]](#). The language tags contain between one and eight characters. Note that the characters used in the language tag are in the Basic Latin alphabet that maps to single-byte characters in the UTF-8 encoding.

See [2 Normative references](#), for more information on RFC 3066 ([2.\[RFC3066\]](#)), [ISO/IEC 10646](#), [ISO/IEC 639](#), and [ISO 3166](#).

## 15.3 Abstract types

### 15.3.1 X3DTextStyleNode

```
X3DTextStyleNode : X3DNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This abstract node type is the base node type for all font style nodes.



## 15.4 Node reference

### 15.4.1 FontStyle

```
FontStyle : X3DFontStyleNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFString [] family "SERIF"
  SFBool [] horizontal TRUE
  MFString [] justify "BEGIN" ["BEGIN","END","FIRST","MIDDLE",""]
  SFString [] language ""
  SFBool [] leftToRight TRUE
  SFFloat [] size 1.0 (0,∞)
  SFFloat [] spacing 1.0 (0,∞)
  SFString [] style "PLAIN" ["PLAIN","BOLD","ITALIC","BOLDITALIC",""]
  SFBool [] topToBottom TRUE
}
```

The *FontStyle* node defines the size, family, and style used for [Text](#) nodes (see [15.2.2 Text formatting](#)), as well as the direction of the text strings and any language-specific rendering techniques used for non-English text. See [Text](#) for a description of the [Text](#) node.

The *size* field specifies the nominal height, in the local coordinate system of the [Text](#) node, of glyphs rendered and determines the spacing of adjacent lines of text. Values of the *size* field shall be greater than zero.

The *spacing* field determines the line spacing between adjacent lines of text. The distance between the baseline of each line of text is (*spacing* × *size*) in the appropriate direction (depending on other fields described below). The effects of the *size* and *spacing* field are depicted in [Figure 15.2](#) (*spacing* greater than 1.0). Values of the *spacing* field shall be non-negative.

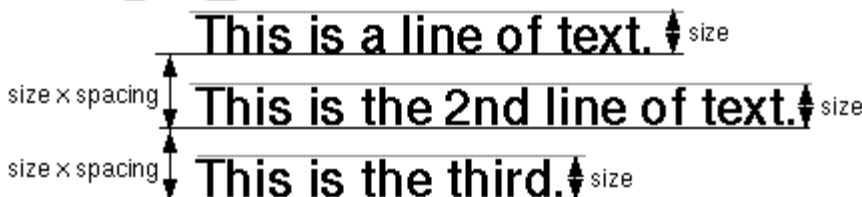


Figure 15.2 — Text *size* and *spacing* fields

### 15.4.2 Text

```
Text : X3DGeometryNode {
  SFNode [in,out] fontStyle NULL [X3DFontStyleNode]
  MFFloat [in,out] length [] (0,∞)
  SFFloat [in,out] maxExtent 0.0 (0,∞)
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFString [in,out] string []
  MFVec2f [out] lineBounds
  SFVec3f [out] origin
  SFVec2f [out] textBounds
  SFBool [] solid FALSE
}
```

The [Text](#) node specifies a two-sided **(by default)**, flat text string object positioned in the  $Z=0$  plane of the local coordinate system based on values defined in the *fontStyle* field (see [15.4.1 FontStyle](#)). [Text](#) nodes may contain multiple text strings specified using the UTF-8 encoding as specified by [ISO 10646](#). The text strings are stored in the order in which the text mode characters are to be produced as defined by the parameters in the [FontStyle](#) node.



The text strings are contained in the *string* field. The *fontStyle* field contains one FontStyle node that specifies the font size, font family and style, direction of the text strings, and any specific language rendering techniques used for the text. If no FontStyle node is specified by the *fontStyle* field, the default values of the FontStyle node are used.

The *maxExtent* field limits and compresses all of the text strings if the length of the maximum string is longer than the maximum extent, as measured in the local coordinate system. If the text string with the maximum length is shorter than the *maxExtent*, then there is no compressing. The maximum extent is measured horizontally for horizontal text (FontStyle node: *horizontal*=TRUE) and vertically for vertical text (FontStyle node: *horizontal*=FALSE). The *maxExtent* field shall be greater than or equal to zero.

The *length* field contains an MFFloat value that specifies the length of each text string in the local coordinate system. The length of each line of type is measured horizontally for horizontal text (FontStyle node: *horizontal*=TRUE) and vertically for vertical text (FontStyle node: *horizontal*=FALSE). The *length* and *maxExtent* fields thus refer to local coordinate units along the dimension of type flow (major axis). If the string is too short, it is stretched (either by scaling the text or by adding space between the characters). If the string is too long, it is compressed (either by scaling the text or by subtracting space between the characters). If a length value is missing (for example, if there are four strings but only three length values), the missing values are considered to be 0. The *length* field shall be greater than or equal to zero.

Specifying a value of 0 for both the *maxExtent* and *length* fields indicates that the string may be any length.

When the default values of *length* and *maxExtent* are used, the Text node shall generate events called *origin*, *lineBounds* and *textBounds* to provide applications with spatial data regarding the size and position of the rendered string(s) with the font being used. These events are also generated when the default values of *length* and *maxExtent* are used and the text is redrawn (e.g., the string field is changed programmatically or the FontStyle node is replaced).

The field *origin* is a single 3D position that specifies the origin of the text local coordinate system in units of the coordinate system in which the Text node is embedded. The value of the *origin* field represents the upper left corner of the *textBounds*. The field *lineBounds* is a set of 2D vectors where each vector contains the size of the 2D bounding box for each line of rendered text in local text x and y units. The *textBounds* event is a single 2D vector that contains the size in x and y dimensions of the Text node's 2D bounding box (all strings) as rendered. An example for each value of the *topToBottom* of the FontStyle node is depicted in [Figure 15.3](#). Through the *origin* event, authors can locate relative measures of *lineBounds* and *textBounds* regardless of the FontStyle's major or minor axis.

NOTE In horizontal font styles, the x dimension of the *lineBounds* and *textBounds* fields is equivalent to a specified *length* or *maxExtent* (the major axis). However, in vertical font styles, the x dimension of the *lineBounds* and *textBounds* fields is along the minor axis.

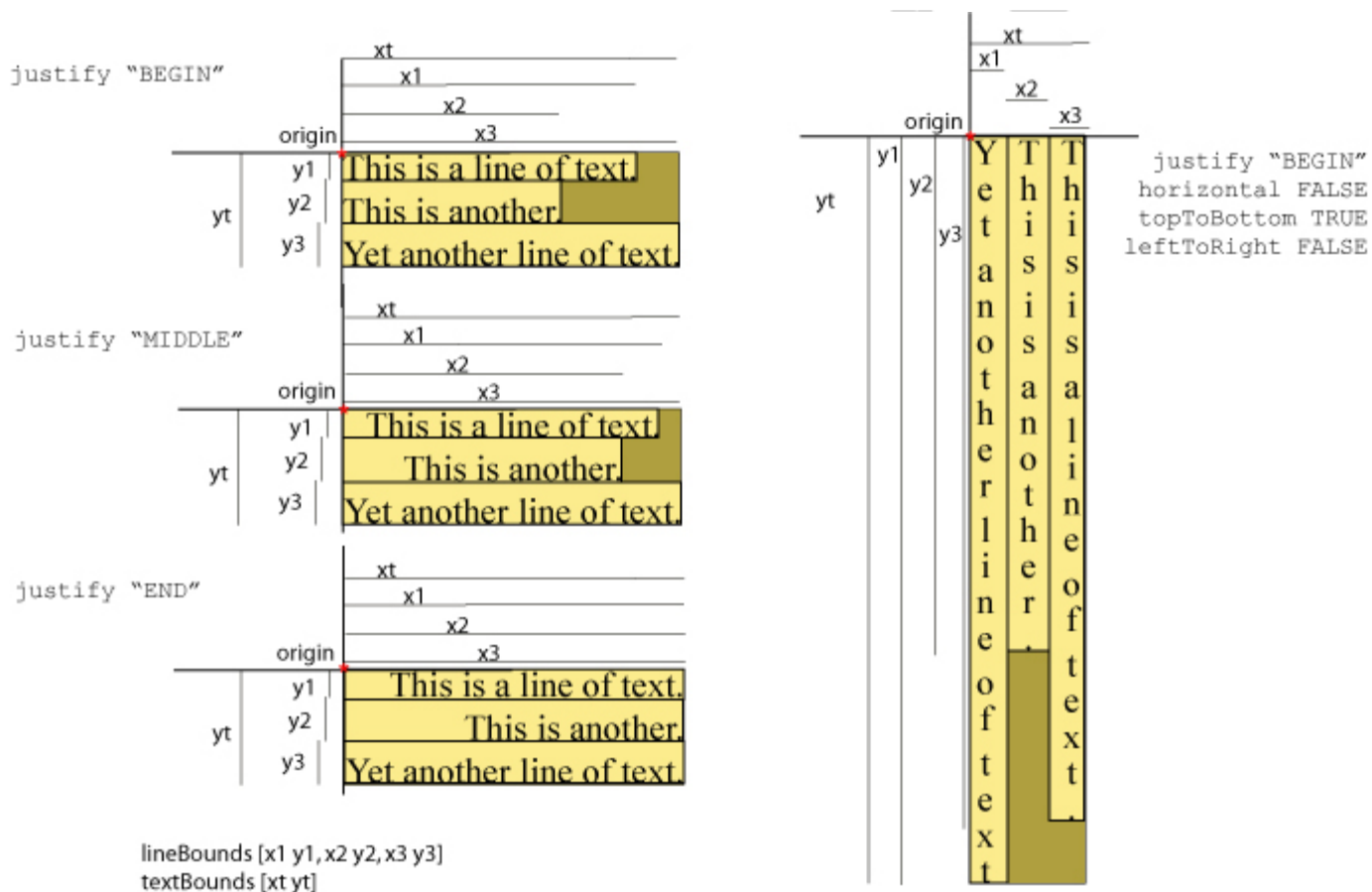


Figure 15.3 — lineBounds and textBounds measurements

[11.2.3 Common geometry fields](#) provides a complete description of the *solid* field.

## 15.5 Support levels

The Text component provides 1 level of support as specified in [Table 15.8](#).

Table 15.8 — Text component support levels

Level	Prerequisites	Nodes/Features	Support
1	Core 1 Grouping 1 Shape 1 Rendering 1		
		<i>X3DFontStyleNode</i> (abstract)	n/a
		FontStyle	All fields fully supported.
		Text	All fields fully supported.





## Extensible 3D (X3D) Part 1: Architecture and base components

### 36 Layout component

#### 36.1 Introduction

##### 36.1.1 Name

The name of this component is "Layout". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 36.1.2 Overview

This subclause describes the Layout component of this part of ISO/IEC 19775. This includes how to precisely position content in a scene in relation to the rendered results. [Table 36.1](#) provides links to the major topics in this subclause.

**Table 36.1 — Topics**

- [36.1 Introduction](#)
  - [36.1.1 Name](#)
  - [36.1.2 Overview](#)
- [36.2 Concepts](#)
  - [36.2.1 Overview](#)
  - [36.2.2 Pixel-specific addressing](#)
  - [36.2.3 Viewports](#)
- [36.3 Abstract types](#)
  - [36.3.1 X3DLayoutNode](#)
- [36.4 Node reference](#)
  - [36.4.1 Layout](#)
  - [36.4.2 LayoutGroup](#)
  - [36.4.3 LayoutLayer](#)
  - [36.4.4 ScreenFontStyle](#)
  - [36.4.5 ScreenGroup](#)
- [36.4 Support levels](#)
- [Table 36.1 — Topics](#)
- [Table 36.2 — Layout component support levels](#)

## 36.2 Concepts

### 36.2.1 Overview

This component provides a set of nodes that allow users to better integrate 2D content with 3D content. In X3D, authors have historically generated a Heads-Up Display (HUD) by placing content in a group that moves along with the user's viewpoint. This approach is limited in that the author has limited control over where the HUD geometry is rendered relative to the display viewport.

EXAMPLE There is no way to ensure that the content will be aligned with a particular edge of the display viewport.

This component provides several nodes that enable the integration of 2D content into the 3D scene. It allows for constructing a hierarchy of rectangular regions that are well suited to contain 2D content, but can also contain 3D content. These 2D regions are not effected by the user navigation or the bound [X3DViewpointNode](#). They are aligned relative to the main scene viewport, or the 2D region that act as its parent.

This component also contains a new [X3DTextStyleNode](#) node that can render text so that it appears identical to typical 2D applications, with the eye soothing technique of anti-aliasing.

### 36.2.2 Pixel-specific addressing

This component also provides utilities that allowing content authors the ability to scale and locate 2D regions and content using pixel-specific addressing. Therefore, some of the nodes and options in this component are dependent on the concept of pixel-based display devices. It is recognized that some implementations do not use such devices. Therefore, those pixel-specific nodes and options are not applicable to those implementations. The pixel-specific nodes and options are contained in a support level designated for pixel-specific concepts.

A node is specified that can exist anywhere in the scene hierarchy. This node forces a scale so that one unit is one pixel.

### 36.2.3 Viewports

The output to a surface can be constrained further by using an [X3DViewportNode](#) node. This node is a special grouping node that defines a set of clipping bounds within the extent of a surface within which the children nodes of the viewport will appear. This provides support for the typical front/side/back/oblique views used by CAD systems.

## 36.3 Abstract types

### 36.3.1 X3DLayoutNode

```
X3DLayoutNode : X3DChildNode {  
  SFNode [in,out] metadata NULL [X3DMetadataObject]
```

}

This is the base node type for layout nodes.

## 36.4 Node Reference

### 36.4.1 Layout

```
Layout : X3DLayoutNode {
  MFString [in,out] align    ["CENTER","CENTER"] ["LEFT"|"CENTER"|"RIGHT"&
    "BOTTOM"|"CENTER"|"TOP"]
  SFNode [in,out] metadata  NULL [X3DMetadataObject]
  MFFloat [in,out] offset   [0,0] (-∞ ∞)
  MFString [in,out] offsetUnits ["WORLD","WORLD"] ["WORLD","FRACTION","PIXEL"]
  MFString [in,out] scaleMode ["NONE","NONE"] ["NONE","FRACTION","STRETCH","PIXEL"]
  MFFloat [in,out] size     [1,1] (0 ∞)
  MFString [in,out] sizeUnits ["WORLD","WORLD"] ["WORLD","FRACTION","PIXEL"]
}
```

The Layout node is used in the layout field of the [LayoutLayer](#) and [LayoutGroup](#) nodes. The Layout node provides all the parameters that are required to define the size and location of a 2D rectangular region that is associated with the containing node. Also, it contains a field that defines how the content of the containing node shall be scaled.

The fields of interest in the Layout node are MFString and MFFloat fields. All have two elements. The first value corresponds to the horizontal direction and the second field corresponds to the vertical direction. If a field has a length of one, that value applies to both the horizontal and vertical directions. If the *align* field has only one value, that value shall be "CENTER".

The width and height of the layout rectangle is defined by two values in the *size* field. The *sizeUnits* field specifies how to interpret the size values. If the value of the *sizeUnits* field is "FRACTION", the size of the corresponding dimension is interpreted as a fraction of the corresponding parent's dimension.

**EXAMPLE** If the *size* value is ( 0.25, 0.5 ) and the value of *sizeUnits* (["FRACTION", "FRACTION"]), the width of the region is one quarter of the width of the parent and the height of the region is one half of the height of the parent.

A *sizeUnits* value of "WORLD" specifies that the corresponding *size* value is interpreted using the current world units of the parent node. Since the LayoutLayer node does not have a parent, a value of "WORLD" is equivalent to a value of "FRACTION". Lastly, a *sizeUnits* value of "PIXEL" specifies that the corresponding size value is in pixel units.

**NOTE** Implementations that do not support the concept of a pixel are not required to support the "PIXEL" option.

The values of the *align*, *offset*, and *offsetUnits* fields are used to determine the location of the layout region. First, the *align* field values align the sized rectangle to an edge or center of the parent rectangle. Then, the offset is applied using the units specified in the *offsetUnits* field. The first value of the *align* field corresponds to the horizontal alignment. The value "LEFT" specifies that the left side of this rectangle shall be aligned with the left side of the parent rectangle. The value "RIGHT" specifies that the right side of this rectangle shall be aligned with the right side of the parent rectangle. The value "CENTER" specifies that this rectangle shall be horizontally centred in its parent. Similarly, the second *align* field value aligns the vertical position of the rectangle to either the "TOP", "BOTTOM" or "CENTER" of the parent rectangle.

After the alignment is applied, the values of the *offset* field are used to translate the location of this rectangle after the initial alignment. The value of the offset field is interpreted using the value of the *offsetUnits* field, using the same options and logic as the *sizeUnits* field, described above.

The *scaleMode* field specifies how the scale of the parent is modified. The *scale* field has two values, the first specifies the horizontal scale and the second value specifies the vertical scale. A *scaleMode* field value of "NONE" specifies that the corresponding scale value is not modified. Instead, the scale is inherited from its parent. Since a *LayoutLayer* node does not have a parent, the value of "NONE" reverts to "FRACTION". A *scaleMode* value of "FRACTION" specifies a scale in the corresponding direction so that one unit is equal to the dimension (width or height) of this rectangle. A value of "PIXEL" specifies a scale in the corresponding direction such that one unit is equal to one pixel.

NOTE Implementations that do not support the concept of a pixel are not required to support this "PIXEL" option.

A *scaleMode* value of "STRETCH" specifies a scale in the corresponding direction such that the resulting scale in the horizontal direction is equal to the scale in the vertical direction, thus producing a uniform scale. If one of the dimensions has a *scaleMode* value of "STRETCH", and the other dimension has a value other than "STRETCH", the scale for the dimension that is not "STRETCH" shall be computed first and the dimension corresponding to the value of "STRETCH" can then be computed to achieve a uniform scale. If both components of the *scaleMode* field are "STRETCH", the scale component corresponding to the larger dimension of the rectangular region is set so that one unit is equal to the dimension of the rectangle, and the other scale component is set so that the resulting scale in the horizontal and vertical directions are the same.

## 36.4.2 LayoutGroup

```
LayoutGroup : X3DGroupingNode {
  MFNode [in]  addChildren    [X3DChildNode]
  MFNode [in]  removeChildren [X3DChildNode]
  MFNode [in,out] children    [] [X3DChildNode]
  SFFBool [in,out] bboxDisplay FALSE
  SFNode [in,out] layout      NULL [X3DLayoutNode]
  SFNode [in,out] metadata    NULL [X3DMetadataObject]
  SFNode [in,out] viewport    NULL [X3DViewportNode]
  SFFBool [in,out] visible    TRUE
  SFVec3f []   bboxCenter    0 0 0 (-∞,∞)
  SFVec3f []   bboxSize      0 0 0 (-∞,∞)
}
```

The *LayoutGroup* is a grouping node whose children are related by a common layout within a parent layout. Thus, a *LayoutGroup* can only be a child of a [LayoutLayer](#) node or another *LayoutGroup* node.

The *layout* field contains an [X3DLayoutNode](#) node that specifies the information required to locate and size the layout region of the *LayoutGroup* node relative to its parent's layout region and to scale the contents of the *LayoutGroup*. The content of the *LayoutGroup* is clipped by the specified viewport.

[10.2.1 Grouping and children node types](#) specifies the *children*, *addChildren*, and *removeChildren* fields.

The origin of the node is always in the center of its layout region. Thus, children (with the exception of *LayoutGroup*) are specified in a coordinate system whose origin is located at the center of the rectangle and can be transformed from that location.



The `LayoutGroup` node does not directly have any pixel dependent concepts. However, the `LayoutGroup` node does contain a `Layout` node that does have pixel-specific options.

### 36.4.3 LayoutLayer

```
LayoutLayer : X3DLayerNode {
  MFNode [in]  addChildren    [X3DChildNode]
  MFNode [in]  removeChildren [X3DChildNode]
  MFNode [in,out] children    [] [X3DChildNode]
  SFNode [in,out] layout      NULL [X3DLayoutNode]
  SFNode [in,out] metadata    NULL [X3DMetadataObject]
  MFString [in,out] objectType "ALL" ["ALL","NONE","TERRAIN",...]
  SFBool [in,out] isPickable  pickable TRUE
  SFNode [in,out] viewport    NULL [X3DViewportNode]
  SFBool [in,out] visible     TRUE
}
```

The `LayoutLayer` node specifies a *children* field that contains a list of nodes that define the subscene.

[10.2.1 Grouping and children node types](#) specifies the *children*, *addChildren*, and *removeChildren* fields.

An [OrthoViewpoint](#) node is automatically established as the default node on the binding stack. Although not restricted to require this, the `LayoutLayer` node is typically used as the last rendered node in a [LayerSet](#) ordering.

The *layout* field contains an instance of [X3DLayoutNode](#) that contains the information required to locate and size the `LayoutLayer` node's rectangular region relative to the main viewport, and to scale the content of the `LayoutLayer`. The content of the `LayoutLayer` is clipped by the defined rectangular region.

### 36.4.4 ScreenFontStyle

```
ScreenFontStyle : X3DFontStyleNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFString [] family "SERIF"
  SFBool [] horizontal TRUE
  MFString [] justify "BEGIN" ["BEGIN","END","FIRST","MIDDLE",""]
  SFString [] language ""
  SFBool [] leftToRight TRUE
  SFFloat [] pointSize 12.0 [0,∞)
  SFFloat [] spacing 1.0 [0,∞)
  SFString [] style "PLAIN" ["PLAIN","BOLD","ITALIC","BOLDITALIC",""]
  SFBool [] topToBottom TRUE
}
```

The `ScreenFontStyle` node specifies fonts styles in terms of the characteristics of a particular surface upon which the text is to be rendered.

The fields in the `ScreenFontStyle` node are the same as those in the [FontStyle](#) node with a single exception: the *size* field of the `FontStyle` node is replaced with a *pointSize* field. The *pointSize* field specifies the size of text in points. Thus, the distance between the baseline of each line of text is (*spacing* × *pointSize*) in the appropriate direction.

Each glyph of the text should be rendered as a quadrilateral with texture applied. The texture for each character shall be generated using the specified font and font attributes. The texture shall have an alpha component whose alpha value shall be derived from the anti-aliasing feature of the glyph. Rendering should occur with bi-linear filtering turned off for best results.

Otherwise, the attributes are as specified in [15.4.1 FontStyle](#).



## 36.4.5 ScreenGroup

```

ScreenGroup : X3DGroupingNode {
  MFNode [in]  addChildren    [X3DChildNode]
  MFNode [in]  removeChildren [X3DChildNode]
  MFNode [in,out] children    [] [X3DChildNode]
  SFBool [in,out] bboxDisplay FALSE
  SFNode [in,out] metadata    NULL [X3DMetadataObject]
  SFBool [in,out] visible     TRUE
  SFVec3f []  bboxCenter     0 0 0 (-∞,∞)
  SFVec3f []  bboxSize       -1 -1 -1 (0,∞) or -1 -1 -1
}

```

The ScreenGroup node is a node derived from [X3DGroupingNode](#) with one additional functional feature: it modifies the scale in such a way that one unit is equal to one pixel in both the horizontal and vertical directions.

If the ScreenGroup node is a child of a [Billboard](#) node that is screen-aligned (*i.e.*, has an *axisOfRotation* value of (0,0,0)), the children of the ScreenGroup shall be both screen-aligned and scaled so that one unit is equal to one pixel. This allows users to place screen-aligned and screen-scaled content into the 3D scene. It will maintain its location in the 3D scene but can be occluded by other geometry that lies in front. Additionally, it can occlude other geometry that lies in back.

[10.2.1 Grouping and children node types](#) specifies the *children*, *addChildren*, and *removeChildren* fields.

The *bboxCenter* and *bboxSize* fields specify a bounding box that encloses the children. This is a hint that may be used for optimization purposes. The results are undefined if the specified bounding box is smaller than the actual bounding box of the children at any time. The default *bboxSize* value, (-1, -1, -1), implies that the bounding box is not specified and, if needed, shall be calculated by the browser. More details on the *bboxCenter* and *bboxSize* fields can be found in [10.2.2 Bounding boxes](#).

## 36.5 Support levels

The Layout component provides two levels of support as specified in [Table 36.2](#). Level 1 provides the basic support for layout. Level 2 provides for pixel-specific addressing.

**Table 36.2 — Layout component support levels**

Level	Prerequisites	Nodes	Support
<b>1</b>	Core 1 Grouping 1 Layering 1		
		<i>X3DLayoutNode</i>	n/a
		Layout	All fields fully supported except "PIXEL" values <b>not optionally</b> supported.
		LayoutGroup	All fields fully supported.

		LayoutLayer	All fields fully supported.
<b>2</b>	Core 1 Grouping 1 Layering 1 Text 1		
		All Level 1 nodes	All fields fully supported.
		ScreenFontStyle	All fields fully supported.
		ScreenGroup	All fields fully supported.





## Extensible 3D (X3D) Part 1: Architecture and base components

### Component index

#### General

This index lists the components in alphabetical order by component title. The "Component" column lists the component title. **The entries in the "Component" column are also hyperlinked and includes links** to the component specification. The "Name" column lists the component name used in the COMPONENT statement. The "Clause" column specifies the clause that contains the specification of the component.

Component	Name	Clause
<a href="#">Annotation</a>	CADGeometry	42
<a href="#">CAD geometry</a>	CADGeometry	32
<a href="#">Core</a>	Core	7
<a href="#">Cube map environmental texturing</a>	CubeMapTexturing	34
<a href="#">Distributed interactive simulation</a>	DIS	28
<a href="#">Environmental effects</a>	EnvironmentalEffects	24
<a href="#">Environmental sensor</a>	EnvironmentalSensor	22
<a href="#">Event utilities</a>	EventUtilities	30
<a href="#">Followers</a>	Followers	39
<a href="#">Geometry2D</a>	Geometry2D	14
<a href="#">Geometry3D</a>	Geometry3D	13
<a href="#">Geospatial</a>	Geospatial	25
<a href="#">Grouping</a>	Grouping	10
<a href="#">Humanoid animation (H-Anim)</a>	H-Anim	26

<a href="#">Interpolation</a>	Interpolation	19
<a href="#">Key device sensor</a>	KeyDeviceSensor	21
<a href="#">Layering</a>	Layering	35
<a href="#">Layout</a>	Layout	36
<a href="#">Lighting</a>	Lighting	17
<a href="#">Navigation</a>	Navigation	23
<a href="#">Networking</a>	Networking	9
<a href="#">NURBS</a>	NURBS	27
<a href="#">Particle systems</a>	ParticleSystems	40
<a href="#">Picking sensor</a>	PickingSensor	38
<a href="#">Pointing device sensor</a>	PointDeviceSensor	20
<a href="#">Programmable shaders</a>	Shaders	31
<a href="#">Projective texture mapping</a>	ProjectiveTextureMapping	43
<a href="#">Rendering</a>	Rendering	11
<a href="#">Rigid body physics</a>	RigidBodyPhysics	37
<a href="#">Scripting</a>	Scripting	29
<a href="#">Shape</a>	Shape	12
<a href="#">Sound</a>	Sound	16
<a href="#">Text</a>	Text	15
<a href="#">Texturing</a>	Texturing	18
<a href="#">Texturing3D</a>	Texturing3D	33
<a href="#">Time</a>	Time	8
<a href="#">Volume rendering</a>	VolumeRendering	41





## Extensible 3D (X3D) Part 1: Architecture and base components

### 16 Sound component



#### 16.1 Introduction

##### 16.1.1 Name

The name of this component is "Sound". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 16.1.2 Overview

This clause describes the Sound component of this part of ISO/IEC 19775. This includes how sound is delivered to an X3D world as well as how sounds are accessed. [Table 16.1](#) provides links to the major topics in this clause.

Table 16.1 — Topics

- [16.1 Introduction](#)
  - [16.1.1 Name](#)
  - [16.1.2 Overview](#)
- [16.2 Concepts](#)
  - [16.2.1 Sound priority](#)
  - [16.2.2 Sound attenuation and spatialization](#)
  - [16.2.3 Sound propagation](#)
  - [16.2.4 Sound effects processing](#)
- [16.3 Abstract types](#)
  - [16.3.1 X3DSoundAnalysisNode](#)
  - [16.3.2 X3DSoundChannelNode](#)
  - [16.3.3 X3DSoundDestinationNode](#)
  - [16.3.4 X3DSoundProcessingNode](#)
  - [16.3.5 X3DSoundNode](#)
  - [16.3.6 X3DSoundSourceNode](#)
- [16.4 Node reference](#)
  - [16.4.1 Analyser](#)
  - [16.4.2 AudioBufferSource](#)
  - [16.4.3 AudioClip](#)
  - [16.4.4 AudioDestination](#)
  - [16.4.5 BiquadFilter](#)
  - [16.4.6 ChannelMerger](#)
  - [16.4.7 ChannelSplitter](#)
  - [16.4.8 Convolver](#)
  - [16.4.9 Delay](#)
  - [16.4.10 DynamicsCompressor](#)
  - [16.4.11 ListenerPoint](#)
  - [16.4.12 MicrophoneSource](#)
  - [16.4.13 OscillatorSource](#)
  - [16.4.14 PeriodicWave](#)
  - [16.4.15 SpatialSound](#)
  - [16.4.16 Sound](#)

- [16.4.17 StreamAudioDestination](#)
- [16.4.18 StreamAudioSource](#)
- [16.4.19 WaveShaper](#)
- [16.5 Support levels](#)
- [Figure 16.1 — Stereo panning](#)
- [Figure 16.2 — Sound node geometry](#)
- [Figure 16.3 — SpatialSound Panning Gain Relationships for viewer \(or ListenerPoint\)](#)
- [Table 16.1 — Topics](#)
- [Table 16.2 — Sound component support levels](#)

## 16.2 Concepts

### 16.2.1 Sound priority

If the browser does not have the resources to play all of the currently active sounds, it is recommended that the browser sort the active sounds into an ordered list using the following sort keys in the order specified:

- a. decreasing *priority*;
- b. for sounds with *priority* > 0.5, increasing (*now*-*startTime*);
- c. decreasing *intensity* at viewer location (*intensity* × "intensity attenuation");

where *priority* is the *priority* field of the Sound node, *now* represents the current time, *startTime* is the *startTime* field of the audio source node specified in the *source* field, and "intensity attenuation" refers to the intensity multiplier derived from the linear decibel attenuation ramp between inner and outer ellipsoids.

It is important that sort key 2 be used for the high priority (event and cue) sounds so that new cues are heard even when the browser is "full" of currently active high priority sounds. Sort key 2 should not be used for normal priority sounds, so selection among them is based on sort key 3 (intensity at the location of the viewer).

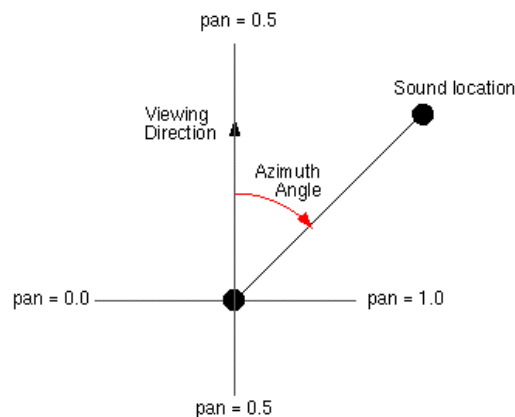
The browser shall play as many sounds from the beginning of this sorted list as it can given available resources and allowable latency between rendering. On most systems, the resources available for MIDI streams are different from those for playing sampled sounds, thus it may be beneficial to maintain a separate list to handle MIDI data.

### 16.2.2 Sound attenuation and spatialization

In order to create a linear decrease in loudness as the viewer moves from the inner to the outer ellipsoid of the sound, the attenuation must be based on a linear decibel ramp. To make the falloff consistent across browsers, the decibel ramp is to vary from 0 dB at the minimum ellipsoid to -20 dB at the outer ellipsoid. Sound nodes with an outer ellipsoid that is ten times larger than the minimum will display the inverse square intensity drop-off that approximates sound attenuation in an anechoic environment.

Browsers may support spatial localization of sounds whose *spatialize* field is `TRUE` as well as their underlying sound libraries will allow. Browsers shall at least support stereo panning of non-MIDI sounds based on the angle between the viewer and the source. This angle is obtained by projecting the *Sound location* (in global space) onto the XZ plane of the viewer. Determine the angle between the Z-axis and the vector from the viewer to the transformed *location*, and assign a pan value in the range [0.0, 1.0] as depicted in [Figure 16.1](#). Given this pan value, left and right channel levels can be obtained using the following equations:

$$\begin{aligned} \text{leftPanFactor} &= 1 - \text{pan}^2 \\ \text{rightPanFactor} &= 1 - (1 - \text{pan})^2 \end{aligned}$$



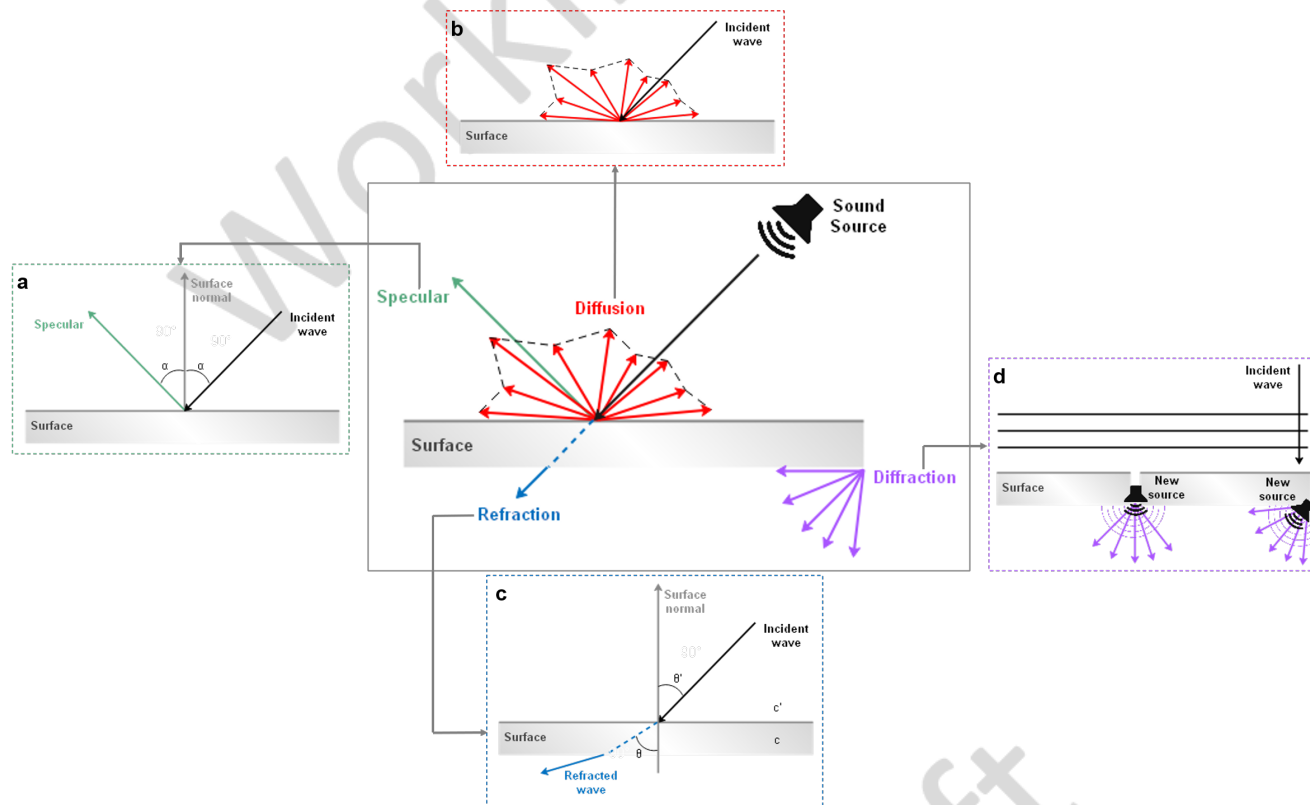
**Figure 16.1 — Stereo panning**

Using this technique, the loudness of the sound is modified by the *intensity* field value, then distance attenuation to obtain the unspatialized audio output. The values in the unspatialized audio output are then scaled by *leftPanFactor* and *rightPanFactor* to determine the final left and right output signals. The use of more sophisticated localization techniques is encouraged, but not required (see [\[SNDB1\]](#)).

These planar gain-reduction relationships pertain to relative direction of current viewer and also any [ListenerPoint](#) nodes.

### 16.2.3 Sound propagation

Sound-propagation techniques can be used to simulate sound waves as they travel from each source to scene listening points by taking into account the expected interactions with various objects in the scene. In other words, spatial sound rendering includes the estimation of physical effects involved in sound propagation such as surface reflection (specular, diffusion) and wave phenomena (refraction, diffraction) within a 3D scene. [Figure 16.2](#) provides an overview of the physical models of sound propagation that are considered.



**Figure 16.2 — Sound Propagation Phenomena**

- *Specular and diffuse reflection*: during the propagation of a sound wave in an enclosed space, the wave hits objects or room boundaries and its free propagation is disturbed. Moreover, during this process, at least a

portion of the incident wave is thrown back, a phenomenon known as reflection. If the wavelength of the sound wave is small enough with respect to the dimensions of the reflecting object and large compared with possible irregularities of the reflecting surface, a specular reflection occurs. This phenomenon is illustrated in [Figure 16.2](#) (inset a), in which the angle of reflection is equal to the angle of incidence. In contrast, if the sound wavelength is comparable to the corrugation dimensions of an irregular reflection surface, the incident sound wave is scattered in many directions. In this case, the phenomenon is called diffuse reflection and is illustrated in [Figure 16.2](#) (inset b).

- *Refraction*: it is the change in the propagation direction of waves when they obliquely cross the boundary between two mediums where their speed changes, as shown in [Figure 16.2](#) (inset c). For transmission of a plane sound wave from air into another medium, the refraction index in following equation (Snell's Law) is used, for calculating the geometric conditions.  

$$n = c'/c = \sin\theta'/\sin\theta$$
 where  $c'$  and  $c$  the sound speed in the two media,  $\theta$  the angle of incidence and  $\theta'$  the angle of refraction.
- *Diffraction*: the fact that a listener can hear sounds around corners and around barriers involves a diffraction model of sound. It is the spread of waves around corners, behind obstacles or around the edges of an opening as illustrated in [Figure 16.2](#) (inset d). The amount of diffraction increases with wavelength, meaning that sound waves with lower frequencies, and thus with greater wavelengths than obstacles or openings dimensions, is spread over larger regions behind the openings or around the obstacles.

(TODO: consider improvement or removal.) Diffraction sources are not explicitly represented in this component, and often can be handled by computational engines. Complex geometric openings may also be modeled by an audio chain including [ListenerPoint](#) and [SpatialSound](#) to emulate sophisticated diffraction propagation paths.

If a simplified geometry alternative from [Collision proxy](#) field is available, it is used preferentially by collision-detection algorithms for sound propagation, rather than descendant children of the [Collision](#) node. Such geometric simplifications can often reduce computational costs significantly without reduction in perceived audio fidelity of 3D scene acoustics. (TODO: consider need for *acousticProxy* field, or if *Shape/Appearance/AcousticProperties* is sufficient.)

## 16.2.4 Sound effects processing

Sound streams can be manipulated by a variety of sound effects. Audio graphs are a powerful mechanism for modeling the diversity of real-world and electronic modifications to sound that can occur. Close integration of sound rendering and effects with 3D models and aggregate scenes provides powerful capabilities for increased realism.

Historically a wide variety of computational libraries for sound generation and propagation have been available, often with significant differences and limitations. Sound propagation and effects processing in this component are based on design patterns found in W3C Web Audio API [\[W3C-WebAudio\]](#). Design goals of that specification include supporting "the capabilities found in modern game audio engines as well as some of the mixing, processing, and filtering tasks that are found in modern desktop audio production applications." These capabilities are broad, implemented in a variety of libraries, and deployed in multiple Web browsers. The primary interfaces of W3C Web Audio API [\[W3C-WebAudio\]](#) necessary for creating audio graphs have corresponding X3D node support in this component.

TODO continued design, implementation and evaluation work for this component is needed to ensure that full coverage of W3C Audio API capabilities is achieved.

Descriptions follow for a number of fields that are common to multiple nodes related to sound processing.

The *channelCount* field is the number of channels used when up-mixing and down-mixing connections to any inputs of a node. The default value is typically 2 except for specific nodes where its value is specially determined This attribute has no effect for nodes with no inputs.

The *channelCountMode* field is used to determine the *computedNumberOfChannels* that controls how inputs to a node are to be mixed.

- "max": use *computedNumberOfChannels* (value for *channelCount* is ignored)
- "clamped-max": use *computedNumberOfChannels* clamped to maximum value given by *channelCount*
- "explicit": Up-mix by filling channels until they run out then zero out remaining channels. Down-mix by filling as many channels as possible, then dropping remaining channels. .

The *channelInterpretation* field determines how individual channels are treated when up-mixing and down-mixing connections to any inputs to the node. The default value is "speakers". This attribute has no effect for nodes with no inputs. Allowed values include the following:

- "speakers": use up-mix equations or down-mix equations. In cases where the number of channels do not match "discrete"



any of these basic speaker layouts, revert to

- "discrete": *computedNumberOfChannels* is the exact value as specified by the *channelCount*.

The *gain* field is amplification applied to an input signal. [TODO linear factor or decibels?](#)

The *numberOfInputs* field is the number of inputs feeding into a node.

The *numberOfOutputs* field is the number of outputs coming out of a node.

## 16.3 Abstract types

TODO: do most or all interfaces include a *gain* field?

### 16.3.1 X3DSoundAnalysisNode

```
X3DSoundAnalysisNode : X3DNode {
  SFString [in,out] description ""
  SFInt32 [in,out] channelCount 0 [0,∞)
  SFString [in,out] channelCountMode "max" ["max", "clamped-max", "explicit"]
  SFString [in,out] channelInterpretation "speakers" ["speakers", "discrete"]
  SFInt32 [in,out] numberOfInputs 0 [0,∞)
  SFInt32 [in,out] numberOfOutputs 0 [0,∞)
  # Mechanisms for parent-child input-output graph design remain under review
}
```

This is the base node type for nodes which receive real-time generated data, without any change from the input to output sound information.

TODO: if enabled FALSE, does signal pass through unmodified or is it blocked? Perhaps an additional boolean is needed for pass-through state? Modeling the 'connect' attribute and defining defaults is necessary for each case.

### 16.3.2 X3DSoundChannelNode

```
X3DSoundChannelNode : X3DTimeDependentNode {
  SFString [in,out] description ""
  SFBool [in,out] enabled TRUE
  SFBool [in,out] loop FALSE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTime [in,out] pauseTime 0 (-∞,∞)
  SFTime [in,out] resumeTime 0 (-∞,∞)
  SFTime [in,out] startTime 0 (-∞,∞)
  SFTime [in,out] stopTime 0 (-∞,∞)
  SFTime [out] elapsedTime
  SFBool [out] isActive
  SFBool [out] isPaused
  SFInt32 [in,out] channelCount 0 [0,∞)
  SFString [in,out] channelCountMode "max" ["max", "clamped-max", "explicit"]
  SFString [in,out] channelInterpretation "speakers" ["speakers", "discrete"]
  SFInt32 [in,out] numberOfInputs 0 [0,∞)
  SFInt32 [in,out] numberOfOutputs 0 [0,∞)
  # Mechanisms for parent-child input-output graph design remain under review
}
```

This is the base node type for nodes that handle of channels in an audio stream, allowing them to be split or merged.

(Section moved here and adapted from [AudioClip](#).)

The *description*, *enabled*, *loop*, *pauseTime*, *resumeTime*, *startTime*, and *stopTime* inputOutput fields and the *elapsedTime*, *isActive*, and *isPaused* outputOnly fields, and their effects on nodes implementing this abstract node type, are discussed in detail in [X3DTimeDependentNode](#) and [8.2.4 Time-dependent nodes](#).

TODO: if enabled FALSE, does signal pass through unmodified or is it blocked? Perhaps an additional boolean is needed for pass-through state? Modeling the 'connect' attribute and defining defaults is necessary for each case.

### 16.3.3 X3DSoundDestinationNode

```
X3DSoundDestinationNode : X3DTimeDependentNode {
  SFString [in,out] description ""
  SFBool [in,out] enabled TRUE
  SFBool [in,out] loop FALSE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTime [in,out] pauseTime 0 (-∞,∞)
  SFTime [in,out] resumeTime 0 (-∞,∞)
  SFTime [in,out] startTime 0 (-∞,∞)
  SFTime [in,out] stopTime 0 (-∞,∞)
  SFTime [out] elapsedTime
  SFBool [out] isActive
  SFBool [out] isPaused
  SFInt32 [in,out] channelCount 0 [0,∞)
  SFString [in,out] channelCountMode "max" ["max", "clamped-max", "explicit"]
  SFString [in,out] channelInterpretation "speakers" ["speakers", "discrete"]
  SFInt32 [in,out] numberOfInputs 0 [0,∞)
  SFInt32 [in,out] numberOfOutputs 0 [0,∞)
  # Mechanisms for parent-child input-output graph design remain under review
}
```

This is the base node type for all sound destination nodes, which represent the final destination of an audio signal

and are what the user can ultimately hear. Such nodes are often considered as audio output devices which are connected to speakers. All rendered audio that is intended to be heard gets routed to these terminal nodes.

(Section moved here and adapted from [AudioClip](#).)

The *description*, *enabled*, *loop*, *pauseTime*, *resumeTime*, *startTime*, and *stopTime* inputOutput fields and the *elapsedTime*, *isActive*, and *isPaused* outputOnly fields, and their effects on nodes implementing this abstract node type, are discussed in detail in [X3DTimeDependentNode](#) and [8.2.4 Time-dependent nodes](#).

TODO: if enabled FALSE, does signal pass through unmodified or is it blocked? Perhaps an additional boolean is needed for pass-through state? Modeling the 'connect' attribute and defining defaults is necessary for each case.

### 16.3.4 X3DSoundProcessingNode

```
X3DSoundProcessingNode : X3DTimeDependentNode {
  SFString [in,out] description ""
  SFBool [in,out] enabled TRUE
  SFBool [in,out] loop FALSE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTime [in,out] pauseTime 0 (-∞,∞)
  SFTime [in,out] resumeTime 0 (-∞,∞)
  SFTime [in,out] startTime 0 (-∞,∞)
  SFTime [in,out] stopTime 0 (-∞,∞)
  SFTime [out] elapsedTime
  SFBool [out] isActive
  SFBool [out] isPaused
  # Mechanisms for parent-child input-output graph design remain under review
}
```

This is the base node type for all sound processing nodes, which are used to enhance audio with filtering, delaying, changing gain, etc.

(Section moved here and adapted from [AudioClip](#).)

The *description*, *enabled*, *loop*, *pauseTime*, *resumeTime*, *startTime*, and *stopTime* inputOutput fields and the *elapsedTime*, *isActive*, and *isPaused* outputOnly fields, and their effects on nodes implementing this abstract node type, are discussed in detail in [X3DTimeDependentNode](#) and [8.2.4 Time-dependent nodes](#).

TODO: if enabled FALSE, does signal pass through unmodified or is it blocked? Perhaps an additional boolean is needed for pass-through state? Modeling the 'connect' attribute and defining defaults is necessary for each case.

### 16.3.5 X3DSoundNode

```
X3DSoundNode : X3DChildNode { ""
  SFString [in,out] description ""
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This abstract node type is the base for all sound nodes.

### 16.3.6 X3DSoundSourceNode

```
X3DSoundSourceNode : X3DTimeDependentNode {
  SFString [in,out] description ""
  SFBool [in,out] loop FALSE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTime [in,out] pauseTime 0 (-∞,∞)
  SFTime [in,out] pitch 1.0 (0,∞)
  SFTime [in,out] resumeTime 0 (-∞,∞)
  SFTime [in,out] startTime 0 (-∞,∞)
  SFTime [in,out] stopTime 0 (-∞,∞)
  SFTime [out] duration_changed
  SFTime [out] elapsedTime
  SFBool [out] isActive
  SFBool [out] isPaused
}
```

This abstract node type is used to derive node types that can emit audio data.

(Section moved here and adapted from [AudioClip](#).)

The *description*, *loop*, *pauseTime*, *resumeTime*, *startTime*, and *stopTime* inputOutput fields and the *elapsedTime*, *isActive*, and *isPaused* outputOnly fields, and their effects on nodes implementing this abstract node type, are discussed in detail in [X3DTimeDependentNode](#) and [8.2.4 Time-dependent nodes](#).

The *pitch* field specifies a multiplier for the rate at which sampled sound is played. Values for the *pitch* field shall be greater than zero. Changing the *pitch* field affects both the pitch and playback speed of a sound. A *set\_pitch* event to an active [AudioClip](#) node is ignored and no *pitch\_changed* field is generated. If *pitch* is set to 2.0, the sound shall be played one octave higher than normal and played twice as fast. For a sampled sound, the *pitch* field alters the sampling rate at which the sound is played. The proper implementation of pitch control for MIDI (or other note sequence sound clips) is to multiply the tempo of the playback by the *pitch* value and adjust the MIDI Coarse Tune and Fine Tune controls to achieve the proper pitch change.

A *duration\_changed* event is sent whenever there is a new value for the "normal" duration of the clip. Typically, this will only occur when the current *url* in use changes and the sound data has been loaded, indicating that the clip is

playing a different sound source. The duration is the length of time in seconds for one cycle of the audio for a *pitch* set to 1.0. Changing the *pitch* field will not trigger a *duration\_changed* event. A duration value of "-1" implies that the sound data has not yet loaded or the value is unavailable for some reason. A *duration\_changed* event shall be generated if the **AudioClip** node is loaded when the X3D file is read or the **AudioClip** node is added to the scene graph.

The *isActive* field may be used by other nodes to determine if the **clip** node is currently active.

## 16.4 Node reference

### 16.4.1 Analyser

```

Analyser : X3DSoundAnalysisNode {
  SFString [in,out] description
  SFInt32 [in,out] fftSize 2048 [0,∞)
  SFInt32 [in,out] frequencyBinCount 1024 [0,∞)
  SFFloat [in,out] minDecibels -100 (-∞,∞)
  SFFloat [in,out] maxDecibels -30 (-∞,∞)
  SFFloat [in,out] smoothingTimeConstant 0.8 [0,∞)

  SFInt32 [in,out] channelCount 0 [0,∞)
  SFString [in,out] channelCountMode "max" ["max", "clamped-max", "explicit"]
  SFString [in,out] channelInterpretation "speakers" ["speakers", "discrete"]
  SFInt32 [in,out] numberOfInputs 0 [0,∞)
  SFInt32 [in,out] numberOfOutputs 0 [0,∞)
} # Mechanisms for parent-child input-output graph design remain under review

```

The Analyser node provides real-time frequency and time-domain analysis information, without any change to the input.

The *fftSize* field is an unsigned long value representing the size of the FFT ([Fast Fourier Transform](#)) to be used to determine the frequency domain.

The *frequencyBinCount* field is an unsigned long value half that of the FFT size. This generally equates to the number of data values you will have to play with for the visualization.

The *minDecibels* field is a value representing the minimum power value in the scaling range for the FFT analysis data, for conversion to unsigned byte values.

The *maxDecibels* field is a value representing the maximum power value in the scaling range for the FFT analysis data, for conversion to unsigned byte values.

The *smoothingTimeConstant* field is a value representing the averaging constant with the last analysis frame.

TODO determine if `accessType` is `outputOnly` for derived information

### 16.4.2 AudioBufferSource

```

AudioBufferSource : X3DSoundSourceNode {
  MFFloat [in,out] buffer NULL [-1,1]
  SFString [in,out] description ""
  SFFloat [in,out] detune 0 [0,∞)
  SFFloat [in,out] duration 0 [0,∞)
  SFBool [in,out] loop FALSE
  SFFloat [in,out] loopStart 0 [0,∞)
  SFFloat [in,out] loopEnd 0 [0,∞)
  SFInt32 [in,out] numberOfChannels 0 [0,∞)
  SFFloat [in,out] playbackRate 0 [-∞,∞)
  SFFloat [in,out] sampleRate 0 [0,∞)
  SFInt32 [out] length 0 [0,∞)

  SFInt32 [in,out] channelCount 0 [0,∞)
  SFString [in,out] channelCountMode "max" ["max", "clamped-max", "explicit"]
  SFString [in,out] channelInterpretation "speakers" ["speakers", "discrete"]
  SFInt32 [in,out] numberOfInputs 0 [0,∞)
  SFInt32 [in,out] numberOfOutputs 0 [0,∞)
} # Mechanisms for parent-child input-output graph design remain under review

```

The AudioBufferSource node represents a memory-resident audio asset. Its format is non-interleaved 32-bit floating-point linear PCM values with a normal range of  $[-1, 1]$ , but values are not limited to this range. It can contain one or more channels. Typically, it would be expected that the length of the PCM data would be fairly short (usually somewhat less than a minute). For longer sounds, such as music soundtracks, streaming such as [StreamAudioSource](#) should be used.

The *buffer* field is a data block holding the audio sample data.

The *detune* field

The *duration* field indicates the duration of the PCM audio data in seconds, computed from the *length* field divided by *sampleRate* field.

The *length* field is the length of the PCM audio data in sample-frames.

The *numberOfChannels* field is the discrete number of audio channels for this buffer.

The *playbackRate* field is the speed at which to render the audio stream.

The *sampleRate* field is the sample-rate used for the PCM audio data in samples per second.

### 16.4.3 AudioClip

```

AudioClip : X3DSoundSourceNode, X3DUrlObject {
  SFString [in,out] description ""
  SFBool [in,out] load TRUE
  SFBool [in,out] loop FALSE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTime [in,out] pauseTime 0 (-∞,∞)
  SFFloat [in,out] pitch 1.0 (0,∞)
  SFTime [in,out] refresh 0.0 (0,∞)
  SFTime [in,out] resumeTime 0 (-∞,∞)
  SFTime [in,out] startTime 0 (-∞,∞)
  SFTime [in,out] stopTime 0 (-∞,∞)
  MFString [in,out] url [] [URI]
  SFTime [out] duration_changed
  SFTime [out] elapsedTime
  SFBool [out] isActive
  SFBool [out] isPaused
}

```

An AudioClip node specifies audio data that can be referenced by [Sound](#) nodes.

The *description* field specifies a textual description of the audio source. A browser is not required to display the *description* field but may choose to do so in addition to playing the sound.

The *url* field specifies the URL from which the sound is loaded. Browsers shall support at least the *wavfile* format in uncompressed PCM format (see [\[WAV\]](#)). It is recommended that browsers also support the MIDI file type 1 sound format (see [2.\[MIDI\]](#)) and the MP3 compressed format (see [2.\[11172-11\]](#)). MIDI files are presumed to use the General MIDI patch set. [9.2.1 URLs](#) contains details on the *url* field.

(Sections moved to parent interface [X3DSoundSourceNode](#) and related interfaces in this component.)

The *loop*, *pauseTime*, *resumeTime*, *startTime*, and *stopTime* inputOutput fields and the *elapsedTime*, *isActive*, and *isPaused* outputOnly fields, and their effects on the AudioClip node, are discussed in detail in [8 Time component](#).

The *pitch* field specifies a multiplier for the rate at which sampled sound is played. Values for the *pitch* field shall be greater than zero. Changing the *pitch* field affects both the pitch and playback speed of a sound. A *set\_pitch* event to an active AudioClip is ignored and no *pitch\_changed* field is generated. If *pitch* is set to 2.0, the sound shall be played one octave higher than normal and played twice as fast. For a sampled sound, the *pitch* field alters the sampling rate at which the sound is played. The proper implementation of pitch control for MIDI (or other note sequence sound clips) is to multiply the tempo of the playback by the *pitch* value and adjust the MIDI Coarse Tune and Fine Tune controls to achieve the proper pitch change.

A *duration\_changed* event is sent whenever there is a new value for the "normal" duration of the clip. Typically, this will only occur when the current *url* in use changes and the sound data has been loaded, indicating that the clip is playing a different sound source. The duration is the length of time in seconds for one cycle of the audio for a *pitch* set to 1.0. Changing the *pitch* field will not trigger a *duration\_changed* event. A duration value of "-1" implies that the sound data has not yet loaded or the value is unavailable for some reason. A *duration\_changed* event shall be generated if the AudioClip node is loaded when the X3D file is read or the AudioClip node is added to the scene graph.

The "cycle" of an AudioClip is the length of time in seconds for one playing of the audio at the specified *pitch*.

The *isActive* field may be used by other nodes to determine if the clip is currently active. If an AudioClip is active, it shall be playing the sound corresponding to the sound time (*i.e.*, in the sound's local time system with sample 0 at time 0):

$$t = (\text{now} - \text{startTime}) \bmod (\text{duration} / \text{pitch})$$

### 16.4.4 AudioDestination

```

AudioDestination : X3DSoundDestinationNode {
  SFString [in,out] description ""
  SFInt32 [in,out] maxChannelCount 2 [0,∞)

  SFInt32 [in,out] channelCount 0 [0,∞)
  SFString [in,out] channelCountMode "max" ["max", "clamped-max", "explicit"]
  SFString [in,out] channelInterpretation "speakers" ["speakers", "discrete"]
  SFInt32 [in,out] numberOfInputs 0 [0,∞)
  SFInt32 [in,out] numberOfOutputs 0 [0,∞)
  # Mechanisms for parent-child input-output graph design remain under review
}

```

AudioDestination represents the final audio destination and is what user ultimately hears, typically from the speakers of user device. An AudioDestinationNode representing the audio hardware end-point (the normal case) can

potentially output more than 2 channels of audio if the audio hardware is multi-channel.

The *maxChannelCount* field is the maximum number of channels that the destination is capable of supporting.

### 16.4.5 BiquadFilter

```

BiquadFilter : X3DSoundProcessingNode {
  SFString [in,out] description ""
  SFFloat [in,out] detune 0 [0,∞)
  SFInt32 [in,out] frequency 350 [0,∞)
  SFFloat [in,out] Q 1 [0,∞)
  SFFloat [in,out] gain 0 [0,∞)
  SFString [in,out] type "lowpass" ["lowpass", "highpass", "bandpass", "lowshelf",
    "highshelf", "peaking", "notch", "allpass"]

  SFInt32 [in,out] channelCount 0 [0,∞)
  SFString [in,out] channelCountMode "max" ["max", "clamped-max", "explicit"]
  SFString [in,out] channelInterpretation "speakers" ["speakers", "discrete"]
  SFInt32 [in,out] numberOfInputs 0 [0,∞)
  SFInt32 [in,out] numberOfOutputs 0 [0,∞)
  # Mechanisms for parent-child input-output graph design remain under review
}
    
```

BiquadFilter represents different kinds of filters, tone control devices, and graphic equalizers. Low-order filters are the building blocks of basic tone controls (bass, mid, treble), graphic equalizers, and more advanced filters. Multiple BiquadFilterNode filters can be combined to form more complex filters. The filter parameters such as frequency can be changed over time for filter sweeps, etc.

The *detune* field is a detune value, in cents, for the frequency..

The *frequency* field is the frequency at which the BiquadFilterNode will operate, in Hz.

The *gain* field is the amplitude gain of the filter. Its value is in dB units. The gain is only used for lowshelf, highshelf, and peaking filters.

The *Q* field is Quality Factor (Q) of the filter.

The *type* field is the type of this BiquadFilterNode. Note that the meaning of the different properties (*frequency*, *detune* and *Q*) differs depending on the type of the filter used.

Enumeration	Description
"lowpass"	<p>A <a href="#">lowpass filter</a> allows frequencies below the cutoff frequency to pass through and attenuates frequencies above the cutoff. It implements a standard second-order resonant lowpass filter with 12dB/octave rolloff.</p> <p>frequency The cutoff frequency</p> <p><b>Q</b> Controls how peaked the response will be at the cutoff frequency. A large value makes the response more peaked.</p> <p>gain Not used in this filter type</p>
"highpass"	<p>A <a href="#">highpass filter</a> is the opposite of a lowpass filter. Frequencies above the cutoff frequency are passed through, but frequencies below the cutoff are attenuated. It implements a standard second-order resonant highpass filter with 12dB/octave rolloff.</p> <p>frequency The cutoff frequency below which the frequencies are attenuated</p> <p><b>Q</b> Controls how peaked the response will be at the cutoff frequency. A large value makes the response more peaked.</p> <p>gain Not used in this filter type</p>
	<p>A <a href="#">bandpass filter</a> allows a range of frequencies to pass through and attenuates the frequencies below and above this frequency range. It implements a second-order bandpass filter.</p>

<p>"bandpass"</p>	<p>frequency The center of the frequency band</p> <p><a href="#">Q</a> Controls the width of the band. The width becomes narrower as the Q value increases.</p> <p>gain Not used in this filter type</p>
<p>"lowshelf"</p>	<p>The lowshelf filter allows all frequencies through, but adds a boost (or attenuation) to the lower frequencies. It implements a second-order lowshelf filter.</p> <p>frequency The upper limit of the frequencies where the boost (or attenuation) is applied.</p> <p><a href="#">Q</a> Not used in this filter type.</p> <p>gain The boost, in dB, to be applied. If the value is negative, the frequencies are attenuated.</p>
<p>"highshelf"</p>	<p>The highshelf filter is the opposite of the lowshelf filter and allows all frequencies through, but adds a boost to the higher frequencies. It implements a second-order highshelf filter</p> <p>frequency The lower limit of the frequencies where the boost (or attenuation) is applied.</p> <p><a href="#">Q</a> Not used in this filter type.</p> <p>gain The boost, in dB, to be applied. If the value is negative, the frequencies are attenuated.</p>
<p>"peaking"</p>	<p>The peaking filter allows all frequencies through, but adds a boost (or attenuation) to a range of frequencies.</p> <p>frequency The center frequency of where the boost is applied.</p> <p><a href="#">Q</a> Controls the width of the band of frequencies that are boosted. A large value implies a narrow width.</p> <p>gain The boost, in dB, to be applied. If the value is negative, the frequencies are attenuated.</p>
<p>"notch"</p>	<p>The notch filter (also known as a <a href="#">band-stop or band-rejection filter</a>) is the opposite of a bandpass filter. It allows all frequencies through, except for a set of frequencies.</p> <p>frequency The center frequency of where the notch is applied.</p> <p><a href="#">Q</a> Controls the width of the band of frequencies that are attenuated. A large value implies a narrow width.</p> <p>gain Not used in this filter type.</p>

"allpass"	<p>An <a href="#">allpass filter</a> allows all frequencies through, but changes the phase relationship between the various frequencies. It implements a second-order allpass filter</p> <p>frequency</p> <p>The frequency where the center of the phase transition occurs. Viewed another way, this is the frequency with maximal <a href="#">group delay</a>.</p> <p><b>Q</b></p> <p>Controls how sharp the phase transition is at the center frequency. A larger value implies a sharper transition and a larger group delay.</p> <p>gain</p> <p>Not used in this filter type.</p>
-----------	---

## 16.4.6 ChannelMerger

```

ChannelMerger : X3DSoundChannelNode {
  SFString [in,out] description ""
  SFBool [in,out] enabled TRUE
  SFBool [in,out] loop FALSE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTime [in,out] pauseTime 0 (-∞,∞)
  SFTime [in,out] resumeTime 0 (-∞,∞)
  SFTime [in,out] startTime 0 (-∞,∞)
  SFTime [in,out] stopTime 0 (-∞,∞)
  SFTime [out] elapsedTime
  SFBool [out] isActive
  SFBool [out] isPaused

  SFInt32 [in,out] channelCount 0 [0,∞)
  SFString [in,out] channelCountMode "max" ["max", "clamped-max", "explicit"]
  SFString [in,out] channelInterpretation "speakers" ["speakers", "discrete"]
  SFInt32 [in,out] numberOfInputs 0 [0,∞)
  SFInt32 [in,out] numberOfOutputs 0 [0,∞)
  # Mechanisms for parent-child input-output graph design remain under review
}

```

ChannelMerger unites different monophonic input channels into a single output channel.

## 16.4.7 ChannelSplitter

```

ChannelSplitter : X3DSoundChannelNode {
  SFString [in,out] description ""
  SFBool [in,out] enabled TRUE
  SFBool [in,out] loop FALSE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTime [in,out] pauseTime 0 (-∞,∞)
  SFTime [in,out] resumeTime 0 (-∞,∞)
  SFTime [in,out] startTime 0 (-∞,∞)
  SFTime [in,out] stopTime 0 (-∞,∞)
  SFTime [out] elapsedTime
  SFBool [out] isActive
  SFBool [out] isPaused

  SFInt32 [in,out] channelCount 0 [0,∞)
  SFString [in,out] channelCountMode "max" ["max", "clamped-max", "explicit"]
  SFString [in,out] channelInterpretation "speakers" ["speakers", "discrete"]
  SFInt32 [in,out] numberOfInputs 0 [0,∞)
  SFInt32 [in,out] numberOfOutputs 0 [0,∞)
  # Mechanisms for parent-child input-output graph design remain under review
}

```

ChannelSplitter separates the different channels of an audio source into a set of monophonic output channels.

## 16.4.8 Convolver

```

Convolver : X3DSoundProcessingNode {
  SFString [in,out] description ""
  MFFloat [in,out] buffer NULL [-1,1]
  SFBool [in,out] normalize FALSE

  SFInt32 [in,out] channelCount 0 [0,∞)
  SFString [in,out] channelCountMode "max" ["max", "clamped-max", "explicit"]
  SFString [in,out] channelInterpretation "speakers" ["speakers", "discrete"]
  SFInt32 [in,out] numberOfInputs 0 [0,∞)
  SFInt32 [in,out] numberOfOutputs 0 [0,∞)
  # Mechanisms for parent-child input-output graph design remain under review
}

```

Convolver performs a linear convolution on a given AudioBuffer, often used to achieve a reverberation effect. Potential modifications include chorus effects, reverberation, and telephone-like speech.

The idea for producing room effects is to play back a reference sound in a room, record it, and then (metaphorically) take the difference between the original sound and the recorded one. The result of this is an impulse response that captures the effect that the room has on a sound. These impulse responses are painstakingly recorded in very specific studio settings, and doing this on your own requires serious dedication. There are sites that host many of these pre-recorded impulse response files (stored as audio files). The Web Audio API provides an easy way to apply these impulse responses to your sounds using the ConvolverNode.

The *buffer* field represents a memory-resident audio asset (for one-shot sounds and other short audio clips). Its format is non-interleaved 32-bit linear floating-point PCM values with a normal range of  $[-1, 1]$ , but values are not limited to this range. It can contain one or more channels. Typically, it would be expected that the length of the PCM data would be fairly short (usually somewhat less than a minute). For longer sounds, such as music soundtracks, streaming should be used with the `<audio>` HTML element and `AudioClip`.

The *normalize* field is a boolean that controls whether the impulse response from the buffer is scaled by an equal-power normalization when the buffer attribute is set, or not.

### 16.4.9 Delay

```
Delay : X3DSoundProcessingNode {
  SFString [in,out] description ""
  SFInt32 [in,out] delayTime 0 [0,∞)

  SFInt32 [in,out] channelCount 0 [0,∞)
  SFString [in,out] channelCountMode "max" ["max", "clamped-max", "explicit"]
  SFString [in,out] channelInterpretation "speakers" ["speakers", "discrete"]
  SFInt32 [in,out] numberOfInputs 0 [0,∞)
  SFInt32 [in,out] numberOfOutputs 0 [0,∞)
  # Mechanisms for parent-child input-output graph design remain under review
}
```

Delay causes a time delay between the arrival of input data and subsequent propagation to the output.

The *delayTime* field represents the amount of delay (in seconds) to apply.

### 16.4.10 DynamicsCompressor

```
DynamicsCompressor : X3DSoundProcessingNode {
  SFString [in,out] description ""
  SFFloat [in,out] attack 0.003 [0,∞)
  SFInt32 [in,out] knee 30 [0,∞)
  SFInt32 [in,out] ratio 12 [0,∞)
  SFFloat [in,out] reduction 0 [0,∞)
  SFInt32 [in,out] release 0.25 (-∞,∞)
  SFFloat [in,out] threshold -24 [0,∞)

  SFInt32 [in,out] channelCount 0 [0,∞)
  SFString [in,out] channelCountMode "max" ["max", "clamped-max", "explicit"]
  SFString [in,out] channelInterpretation "speakers" ["speakers", "discrete"]
  SFInt32 [in,out] numberOfInputs 0 [0,∞)
  SFInt32 [in,out] numberOfOutputs 0 [0,∞)
  # Mechanisms for parent-child input-output graph design remain under review
}
```

DynamicsCompressor implements a dynamics compression effect, lowering the volume of the loudest parts of the signal and raises the volume of the softest parts.

The *attack* field is the amount of time (in seconds) to reduce the gain by 10dB.

The *knee* field contains a decibel value representing the range above the threshold where the curve smoothly transitions to the compressed portion.

The *ratio* field represents the amount of change, in dB, needed in the input for a 1 dB change in the output.

The *reduction* field represents the amount of gain reduction currently applied by the compressor to the signal.

The *release* field represents the amount of time (in seconds) to increase the gain by 10dB.

The *threshold* field represents the decibel value above which the compression will start taking effect.

### 16.4.11 ListenerPoint

```
ListenerPoint : X3DAudioListenerNode {
  SFBool [in] set_bind ""
  SFString [in,out] description ""
  SFBool [in,out] enabled TRUE
  SFInt32 [in,out] gain 1 [0,∞)
  SFFloat [in,out] interauralDistance 0 [0, infinity)
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFRotation [in,out] orientation 0 0 1 0 [-1,1],(-∞,∞)
  SFVec3f [in,out] position 0 0 10 (-∞,∞)
  SFBool [in,out] trackCurrentView FALSE
  SFTime [out] bindTime
  SFBool [out] isBound
  # Mechanisms for parent-child input-output graph design remain under review
}
```

ListenerPoint represents the position and orientation of the person listening to the audio scene. It provides single or multiple sound channels as output. Multiple ListenerPoint nodes can be active for sound processing, but only one can be bound as the active listening point for the user.

The *interauralDistance* field is used for binaural recording.

If TRUE the *trackCurrentView* field matches *position* and *orientation* to the user's current view.



## 16.4.12 MicrophoneSource

```
MicrophoneSource : X3DSoundSourceNode {
  SFString [in,out] description ""
  SFBool [in,out] enabled TRUE
  SFBool [in,out] isActive FALSE
  SFString [in,out] mediaDeviceID ""
  # Mechanisms for parent-child input-output graph design remain under review
}
```

MicrophoneSource captures input from a physical microphone.

The *mediaDeviceID* field is a unique identifier for the represented device.

TODO: reconcile whether all the many fields of X3DSoundSourceNode are appropriate.

## 16.4.13 OscillatorSource

```
Oscillator : X3DSoundSourceNode {
  SFString [in,out] description ""
  SFBool [in,out] loop FALSE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTime [in,out] pauseTime 0 (-∞,∞)
  SFFloat [in,out] pitch 1.0 (0,∞)
  SFTime [in,out] resumeTime 0 (-∞,∞)
  SFTime [in,out] startTime 0 (-∞,∞)
  SFTime [in,out] stopTime 0 (-∞,∞)
  SFTime [out] durationChanged
  SFTime [out] elapsedTime
  SFBool [out] isActive
  SFBool [out] isPaused

  SFFloat [in,out] detune 0 [0,∞)
  SFInt32 [in,out] frequency 0 [0,∞)
  SFNode [in,out] periodicWave NULL [PeriodicWave]
  SFString [in,out] type "sine" ["sine", "square", "sawtooth", "triangle", "custom"]
  # Mechanisms for parent-child input-output graph design remain under review
}
```

The Oscillator node represents an audio source generating a periodic waveform, providing a constant tone.

The *detune* field is an a-rate AudioParam representing detuning of oscillation in cents (though the AudioParam returned is read-only, the value it represents is not).

The *frequency* field is an a-rate AudioParam representing the frequency of oscillation in hertz (though the AudioParam returned is read-only, the value it represents is not). The default value is 440 Hz (a standard middle-A note).

The *periodicWave* field is an PeriodicWave used when type="custom" is indicated.

The *type* field is a string which specifies the shape of waveform to play; this can be one of a number of standard values, or custom to use a PeriodicWave to describe a custom waveform. Different types of waves produce different sounds. Standard values are "sine", "square", "sawtooth", "triangle" and "custom". Allowed values are

- "sine": a sine wave
- "square": a square wave of duty period 0.5
- "sawtooth": a sawtooth wave
- "triangle": a triangle wave
- "custom": a custom periodic wave

## 16.4.14 PeriodicWave

```
PeriodicWave : X3DSoundProcessingNode {
  SFString [in,out] description ""
  SFInt32 [in,out] frequency 0 [0,∞)
  SFString [in,out] type "square"
  SFFloat [in,out] detune 0 [0,∞)
}
```

PeriodicWave defines a periodic waveform that can be used to shape the output of an Oscillator.

TODO confirm and describe attributes

## 16.4.15 SpatialSound

```
SpatialSound : X3DSoundNode {
  SFFloat [in,out] coneInnerAngle 6.2832 [0,2π]
  SFFloat [in,out] coneOuterAngle 6.2832 [0,2π]
  SFFloat [in,out] coneOuterGain 0 (-∞,∞)
  SFString [in,out] description ""
  SFVec3f [in,out] direction 0 0 1 (-∞,∞)
  SFString [in,out] distanceModel "INVERSE" ["LINEAR" "INVERSE" "EXPONENTIAL"]
  SFFloat [in,out] intensity 1 [0,1]
  SFVec3f [in,out] location 0 0 0 (-∞,∞)
  SFFloat [in,out] maxDistance 10000 [0,∞)
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFBool [in,out] enableHRTF FALSE
}
```

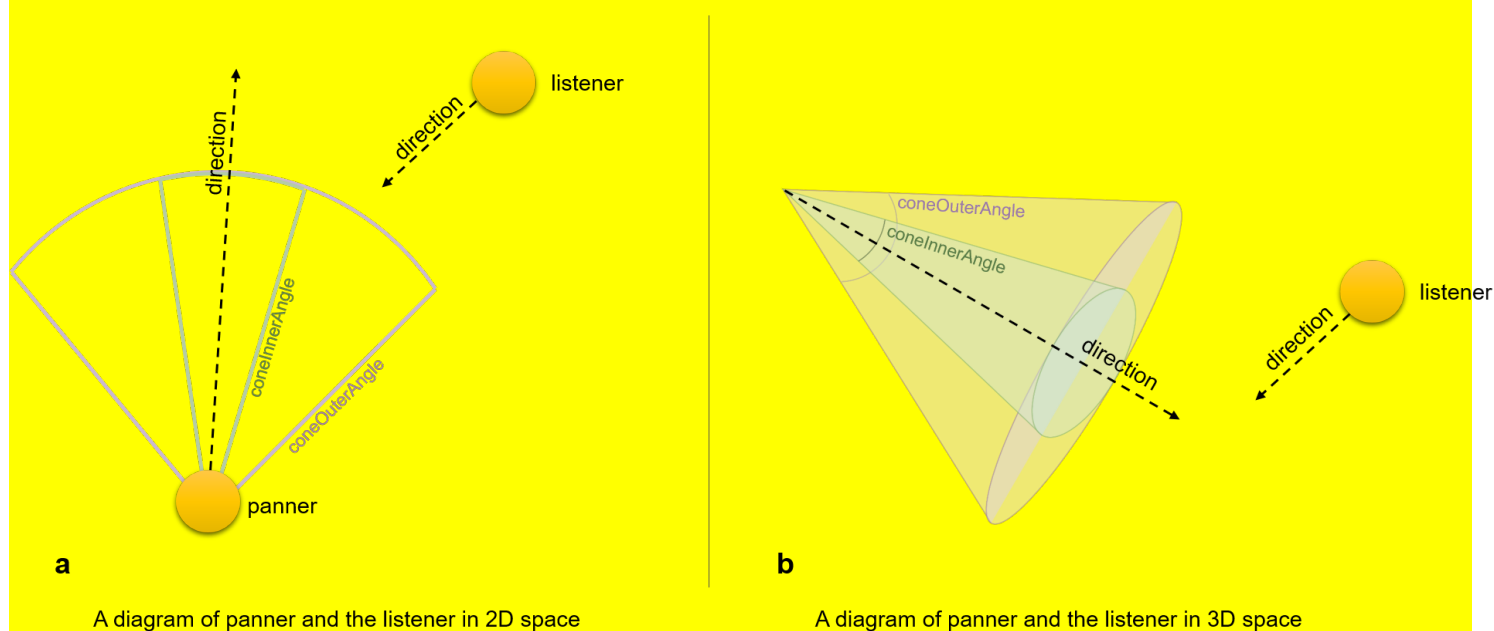
```

SFFloat [in,out] referenceDistance 1 [0,∞)
SFFloat [in,out] rolloffFactor 1 [0,∞)
SFFloat [in,out] priority 0 [0,1]
SFNode [in,out] source NULL [X3DSoundSourceNode] # and other types
SFBool [] spatialize TRUE
    }
    }

```

SpatialSound represents a processing node which positions, emits and spatializes an audio stream in three-dimensional space.

The *coneInnerAngle* is centered along direction and defines the inner conical volume, inside of which no source gain reduction occurs. The *coneOuterAngle* is centered along direction and defines an outer conical volume, within which the sound gain decreases linearly from full gain to *coneOuterGain*. Outside of *coneOuterAngle*, gain equals *coneOuterGain*. The value of *coneOuterAngle* is greater than or equal to *coneInnerAngle*. Corresponding gain reductions for 2D and 3D spatial panning between this source and a viewer (or [ListenerPoint](#)) are shown in [Figure 16.3](#).



**Figure 16.3 — SpatialSound Panning Gain Relationships for viewer (or ListenerPoint)**

The *direction*, *intensity*, *location*, *priority*, *source* and *spatialize* fields match field definitions for Sound node.

The *referenceDistance* field is reference distance for reducing volume as source moves further from the listener.

The *rolloffFactor* field indicates how quickly volume is reduced as source moves further from listener.

The *distanceModel* field specifies which algorithm to use for sound attenuation, corresponding to distance between an audio source and a listener, as it moves away from the listener.

a. LINEAR gain model determined by

$$1 - \text{rolloffFactor} * (\text{distance} - \text{referenceDistance}) / (\text{maxDistance} - \text{referenceDistance})$$

b. INVERSE gain model determined by

$$\text{refDistance} / (\text{referenceDistance} + \text{rolloffFactor} * (\text{Math.max}(\text{distance}, \text{referenceDistance}) - \text{referenceDistance}))$$

c. EXPONENTIAL gain model determined by

$$\text{pow}((\text{Math.max}(\text{distance}, \text{referenceDistance}) / \text{referenceDistance}, -\text{rolloffFactor})$$

The *enableHRTF* field specifies whether to enable Head Related Transfer Function (HRTF) auralization, if available.

The *maxDistance* field is the maximum distance where sound is renderable between source and listener, after which no reduction in sound volume occurs.

Spatial sound has a conceptual role in the Web3D environments, due to highly realism scenes that can provide. Since Web Audio API is the most popular sound engine, we propose to get the necessary steps required to make X3D fully compatible with this library. In fact, we propose the enrichment of X3D with spatial sound features, using the structure and the functionality of [Web Audio API](#).

Particularly, the Web Audio API involves handling audio operations inside an audio context and has been designed to allow modular routing. Also, the approach of Web Audio API is based on the concept of audio context, which

represents the direction of audio stream flows between sound nodes.

TODO describe "cone" fields, likely need explanatory diagram.

## 16.4.16 Sound

```

Sound : X3DSoundNode {
  SFVec3f [in,out] direction 0 0 1 (-∞,∞)
  SFFloat [in,out] intensity 1 [0,1]
  SFVec3f [in,out] location 0 0 0 (-∞,∞)
  SFFloat [in,out] maxBack 10 [0,∞)
  SFFloat [in,out] maxFront 10 [0,∞)
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFFloat [in,out] minBack 1 [0,∞)
  SFFloat [in,out] minFront 1 [0,∞)
  SFFloat [in,out] priority 0 [0,1]
  SFNode [in,out] source NULL [X3DSoundSourceNode]
  SFBool [] spatialize TRUE
}

```

The Sound node specifies the spatial presentation of a sound in a X3D scene. The sound is located at a point in the local coordinate system and emits sound in an elliptical pattern (defined by two ellipsoids). The ellipsoids are oriented in a direction specified by the *direction* field. The shape of the ellipsoids may be modified to provide more or less directional focus from the location of the sound.

The *source* field specifies the sound source for the Sound node. If the *source* field is not specified, the Sound node will not emit audio. The *source* field shall specify either an [AudioClip](#) node or a [MovieTexture](#) node. If a [MovieTexture](#) node is specified as the sound source, the [MovieTexture](#) shall refer to a movie format that supports sound (EXAMPLE MPEG-1Systems, see [ISO/IEC 11172-1](#)).

The *intensity* field adjusts the loudness (decibels) of the sound emitted by the Sound node. The *intensity* field has a value that ranges from 0.0 to 1.0 and specifies a factor which shall be used to scale the normalized sample data of the sound source during playback. A Sound node with an intensity of 1.0 shall emit audio at its maximum loudness (before attenuation), and a Sound node with an intensity of 0.0 shall emit no audio. Between these values, the loudness should increase linearly from a -20 dB change approaching an *intensity* of 0.0 to a 0 dB change at an *intensity* of 1.0.

NOTE This is different from the traditional definition of intensity with respect to sound; see [\[SNDA\]](#).

The *priority* field provides a hint for the browser to choose which sounds to play when there are more active Sound nodes than can be played at once due to either limited system resources or system load. [16.2 Concepts](#) describes a recommended algorithm for determining which sounds to play under such circumstances. The *priority* field ranges from 0.0 to 1.0, with 1.0 being the highest priority and 0.0 the lowest priority.

The *location* field determines the location of the sound emitter in the local coordinate system. A Sound node's output is audible only if it is part of the traversed scene. Sound nodes that are descended from [LOD](#), [Switch](#), or any grouping or prototype node that disables traversal (*i.e.*, drawing) of its children are not audible unless they are traversed. If a Sound node is disabled by a [Switch](#) or [LOD](#) node, and later it becomes part of the traversal again, the sound shall resume where it would have been had it been playing continuously.

The Sound node has an inner ellipsoid that defines a volume of space in which the maximum level of the sound is audible. Within this ellipsoid, the normalized sample data is scaled by the *intensity* field and there is no attenuation. The inner ellipsoid is defined by extending the *direction* vector through the *location*. The *minBack* and *minFront* fields specify distances behind and in front of the *location* along the *direction* vector respectively. The inner ellipsoid has one of its foci at *location* (the second focus is implicit) and intersects the *direction* vector at *minBack* and *minFront*.

The Sound node has an outer ellipsoid that defines a volume of space that bounds the audibility of the sound. No sound can be heard outside of this outer ellipsoid. The outer ellipsoid is defined by extending the *direction* vector through the *location*. The *maxBack* and *maxFront* fields specify distances behind and in front of the *location* along the *direction* vector respectively. The outer ellipsoid has one of its foci at *location* (the second focus is implicit) and intersects the *direction* vector at *maxBack* and *maxFront*.

The *minFront*, *maxFront*, *minBack*, and *maxBack* fields are defined in local coordinates, and shall be greater than or equal to zero. The *minBack* field shall be less than or equal to *maxBack*, and *minFront* shall be less than or equal to *maxFront*. The ellipsoid parameters are specified in the local coordinate system but the ellipsoids' geometry is affected by ancestors' transformations.

Between the two ellipsoids, there shall be a linear attenuation ramp in loudness, from 0 dB at the minimum ellipsoid to -20 dB at the maximum ellipsoid:

$$\text{attenuation} = -20 \times (d' / d'')$$

where  $d'$  is the distance along the location-to-viewer vector, measured from the transformed minimum ellipsoid

boundary to the viewer, and  $d''$  is the distance along the location-to-viewer vector from the transformed minimum ellipsoid boundary to the transformed maximum ellipsoid boundary (see [Figure 16.2](#)).

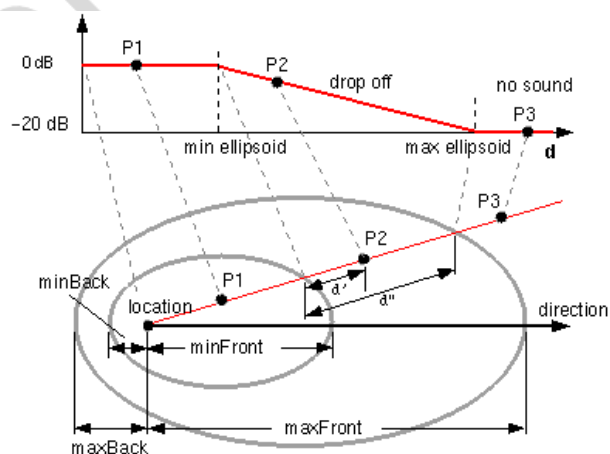


Figure 16.2 — Sound Node Geometry

The *spatialize* field specifies if the sound is perceived as being directionally located relative to the viewer. If the *spatialize* field is `TRUE` and the viewer is located between the transformed inner and outer ellipsoids, the viewer's *direction* and the relative location of the Sound node should be taken into account during playback. Details outlining the minimum required spatialization functionality can be found in [16.2.2 Sound attenuation and spatialization](#). If the *spatialize* field is `FALSE`, directional effects are ignored, but the ellipsoid dimensions and *intensity* will still affect the loudness of the sound. If the sound source is multi-channel (`EXAMPLE` stereo), the source shall retain its channel separation during playback.

### 16.4.17 StreamAudioDestination

```
StreamAudioDestination : X3DSoundDestinationNode {
  SFString [in,out] description ""
  MFFloat [in,out] stream NULL [-1,1]

  SFInt32 [in,out] channelCount 0 [0,∞)
  SFString [in,out] channelCountMode "max" ["max", "clamped-max", "explicit"]
  SFString [in,out] channelInterpretation "speakers" ["speakers", "discrete"]
  SFInt32 [in,out] numberOfInputs 0 [0,∞)
  SFInt32 [in,out] numberOfOutputs 0 [0,∞)
  # Mechanisms for parent-child input-output graph design remain under review
}
```

StreamAudioDestination is an audio destination representing a MediaStream with a single MediaStreamTrack whose kind is "audio".

TODO confirm and describe attributes

### 16.4.198 StreamAudioSource

```
StreamAudioSource : X3DSoundSourceNode {
  SFString [in,out] description ""
  MFFloat [in,out] mediaStream NULL [-1,1]

  SFInt32 [in,out] channelCount 0 [0,∞)
  SFString [in,out] channelCountMode "max" ["max", "clamped-max", "explicit"]
  SFString [in,out] channelInterpretation "speakers" ["speakers", "discrete"]
  SFInt32 [in,out] numberOfInputs 0 [0,∞)
  SFInt32 [in,out] numberOfOutputs 0 [0,∞)
  # Mechanisms for parent-child input-output graph design remain under review
}
```

StreamAudioSource operates as an audio source whose media is received from a MediaStream obtained using the WebRTC or Media Capture and Streams APIs. This media source might originate from a microphone or sound-processing channel provided by a remote peer on a WebRTC call.

TODO confirm and describe attributes

### 16.4.19 WaveShaper

```
WaveShaper : X3DSoundProcessingNode {
  SFString [in,out] description ""
  MFFloat [in,out] curve [] [-1,-1]
  SFString [in,out] oversample "none" ["none", "2x", "4x"]

  SFInt32 [in,out] channelCount 0 [0,∞)
  SFString [in,out] channelCountMode "max" ["max", "clamped-max", "explicit"]
  SFString [in,out] channelInterpretation "speakers" ["speakers", "discrete"]
  SFInt32 [in,out] numberOfInputs 0 [0,∞)
  SFInt32 [in,out] numberOfOutputs 0 [0,∞)
  # Mechanisms for parent-child input-output graph design remain under review
}
```

WaveShaper represents a nonlinear distorter that applies a wave-shaping distortion curve to the signal. Non-linear waveshaping distortion is commonly used for both subtle non-linear warming, or more obvious distortion effects. Arbitrary non-linear shaping curves may be specified.

The *curve* field is an Array of floats numbers describing the distortion to apply.

The *oversample* field specifies what type of oversampling (if any) should be used when applying the shaping curve. Allowed values follow. Note that for some applications, avoiding oversampling can produce a precise shaping curve.

- "none": the curve is applied directly to the input samples with no oversampling.
- "2x": oversample two times to improve the quality of the processing by avoiding some aliasing.
- "4x": oversample four times for highest quality of the processing.

## 16.5 Support levels

The Sound component provides one level of support as specified in [Table 16.2](#).

**Table 16.2 — Sound component support levels**

Level	Prerequisites	Nodes/Features	Support
1	Core 1 Time 1		
		<i>X3DSoundSourceNode</i> (abstract)	n/a
		<i>X3DSoundNode</i> (abstract)	n/a
		AudioClip	All fields fully supported.
		Sound	All fields fully supported.
2	Core 1 Time 1		
		All level 1 Sound nodes	All fields fully supported.
		<i>X3DSoundAnalysisNode</i> , <i>X3DSoundChannelNode</i> , <i>X3DSoundDestinationNode</i> , <i>X3DSoundProcessingNode</i>	All fields fully supported.
		Analyser, AudioBufferSource, AudioBufferSource, AudioDestination, BiquadFilter, ChannelMerger, ChannelSplitter, Convolver, Delay, DynamicsCompressor, ListenerPoint, MicrophoneSource, OscillatorSource, PeriodicWave, SpatialSound, StreamAudioDestination, StreamAudioSource, WaveShaper	All fields fully supported.





## Extensible 3D (X3D) Part 1: Architecture and base components

### 37 Rigid body physics

#### 37.1 Introduction

##### 37.1.1 Name

The name of this component is "RigidBodyPhysics". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 37.1.2 Overview

This clause describes how to model rigid bodies and their interactions through the application of basic physics principles to effect motion. [Table 37.1](#) provides links to the major topics in this clause.

**Table 37.1 — Topics**

- [37.1 Introduction](#)
  - [37.1.1 Name](#)
  - [37.1.2 Overview](#)
- [37.2 Concepts](#)
  - [37.2.1 Overview](#)
  - [37.2.2 Bodies](#)
    - [37.2.2.1 Event model evaluation](#)
    - [37.2.2.2 Transformation hierarchy](#)
  - [37.2.3 Joints](#)
    - [37.2.3.1 What a joint describes](#)
    - [37.2.3.2 Range of motion limits](#)
  - [37.2.4 Coordinate systems](#)
    - [37.2.4.1 Initial coordinate system](#)
    - [37.2.4.2 Breaking joint](#)
    - [37.2.4.3 Collision contact description](#)
- [37.3 Abstract types](#)
  - [37.3.1 X3DNBodyCollidableNode](#)

- [37.3.2 X3DNBodyCollisionSpaceNode](#)
- [37.3.3 X3DRigidJointNode](#)
- [37.4 Node reference](#)
  - [37.4.1 BallJoint](#)
  - [37.4.2 CollidableOffset](#)
  - [37.4.3 CollidableShape](#)
  - [37.4.4 CollisionCollection](#)
  - [37.4.5 CollisionSensor](#)
  - [37.4.6 CollisionSpace](#)
  - [37.4.7 Contact](#)
  - [37.4.8 DoubleAxisHingeJoint](#)
  - [37.4.9 MotorJoint](#)
  - [37.4.10 RigidBody](#)
  - [37.4.11 RigidBodyCollection](#)
  - [37.4.12 SingleAxisHingeJoint](#)
  - [37.4.13 SliderJoint](#)
  - [37.4.14 UniversalJoint](#)
- [37.5 Support levels](#)
- [Table 37.1 — Topics](#)
- [Table 37.2 — appliedParameters valid values](#)
- [Table 37.3 — Rigid body physics component support levels](#)

## 37.2 Concepts

### 37.2.1 Overview

This component provides the ability to influence the visual output of the scene graph in accordance to some of the laws of physics. Only the subset of the laws of physics known as rigid body physics is supported. Rigid body physics models deal with objects as solid, unchangeable sets of mass with a velocity. These bodies can be connected together with the use of various forms of joints, that allow one body's motion to effect another.

Rigid body physics evaluation requires the solving of many different factors in parallel, typically through the use of ordinary differential equations. Because these equations are heavily floating point based, their accuracy is highly dependent on both the implementation of the solver and the computing hardware. Due to this non-precise nature of the calculations, modelling rigid body physics requires a lot of care and attention to detail. Small changes can very quickly lead to numerical instability resulting in visual representations that may make the model look like it is exploding. Most of the node definitions in this component include factors that can be modified to trade off accuracy in visual output for the stability of the calculations. In many cases, the two are inversely proportional. That is, a more accurate simulation has a far greater chance of suffering numerical instability than a less accurate result. Intersections between bodies and the way that they interact per frame can have significant effects on the application visuals.



A consequence of this problem is that using physically accurate values for masses and sizes in the physics model is not likely to produce the best results, or even lead to a stable simulation. The physics modelling presented by this component is independent of the visual representation, allowing the user to create a stable physical model that has no relationship to the visual model that is driven by the physics.

## 37.2.2 Integration with X3D

### 37.2.2.1 Event model evaluation

Evaluating the physics model within the constraints of the X3D event model requires the ability to evaluate time in discrete time chunks. This is known as *discrete event simulation*.

Evaluation of the physics model is performed once per frame. Since the user needs to be able to modify the model on a frame-by-frame basis, this requires that the physics model is evaluated after all possible user input has been received for that frame. Thus, physics model evaluation is performed just after Step d in [4.4.8.3 Execution model](#). After evaluating the physics model, the results are used to further modify the existing scene graph immediately before rendering is performed.

Physics modelling libraries typically require fixed length time intervals between iterations. A real-time 3D graphics environment typically varies the frame rates based on:

- a. the current content in view,
- b. scripting, and
- c. other interactions.

An implementation of this specification shall be responsible for keeping the physics fixed time interval evaluations synchronized with the varying visual frame time intervals.

Some nodes offer output events that describe output of the physics model, such as the current separation between two bodies or the rate of separation between them. These values are exposed as a set of sensors that can be used to track the output of the physics model and report it at the start of the next frame, in accordance with the standard sensor node model.

### 37.2.2.2 Transformation hierarchy

The nodes defined in this component are not part of the transformation hierarchy. Instead, the nodes may be linked to parts of the scene graph that are part of the transformation hierarchy in order to affect their motion. They may also be linked as part of the n-body object collision-detection capabilities so that a coordinated system of graphics and physics may be modelled.

## 37.2.3 Bodies

A body represents a section of mass in the system that can be effected by the physics



model. A body is represented by the following properties:

- a. mass,
- b. density model,
- c. position and orientation,
- d. linear velocity,
- e. angular velocity, and
- f. various forces and torques applied.

A body is a standalone object within a collection. Bodies are influenced by joints that connect this body to another within the collection. Bodies are not required to be connected by a joint and may exist as a standalone entity. All bodies exist within the world space of their collection. There is no concept of a transformation hierarchy of bodies within bodies.

## 37.2.4 Joints

### 37.2.4.1 What a joint describes

A joint is used to connect two bodies together in a way that imposes a set of constraints on the movement of the two bodies relative to one another. Many different joint types are provided allowing the user to constrain the motions of the bodies according to the desired physical properties.

### 37.2.4.2 Range of motion limits

Each of the joints has a range of motion through which they can travel. This range of motion may be radial angles or linear distance. Typically these values are limited to a single rotation in any one axis.

EXAMPLE 1  $2\pi$  radians indicates full rotatability.

Each joint contains a set of fields that can be used to limit the range of motions to less than full ability. These fields are termed *stops*.

A stop is defined by its value and a number of parameters to control the effects output from the physics engine. Firstly, a stop may permit some amount of bouncing due to the action of the joint hitting it. These same values are also used to perform internal self-correction of objects that have interpenetrated due to the discrete time step intervals that the evaluation of the physics model uses.

EXAMPLE 2 In the real world, a lot of stops have a rubber cushion on the end to absorb the impacts and help return the joint to the central position.

## 37.2.5 Coordinate systems

### 37.2.5.1 Initial coordinate system

When the two bodies are initially placed in the scene, their initial positions define the resting coordinate frames for the two bodies on that joint. Output values from those

joints are then relative to this initial position.

The anchor position and axis values of joints are always specified in world coordinate positions, regardless of whether the two joining bodies have been offset or not.

Mass is defined in kilograms. It is important to note that rigid body physics models, due to inaccuracy in floating point calculations, cannot typically deal with real-world values. Values provided should be defined in relative proportions rather than absolute values. This will help the model stay stable over long calculation periods.

### 37.2.5.2 Breaking joints

Each joint node will have two output-only fields that indicate the calculated location of the relative positions within their own frame of reference. By comparing the difference between these two values, it is possible to determine if the joint has broken as a result of the input from the last frame. If the joint broke, the difference between the two values will be non-zero (although the author should also allow a small tolerance due to the inaccuracy of floating point calculations).

### 37.2.5.3 Collision contact description

When a collision is found between two objects, it is described with the following details:

- a unit vector describing the surface normal from body 1 to body 2 at the point of contact,
- a primary direction of motion for body 1 relative to body 2, and
- a second direction that is perpendicular to both the normal and the primary direction is implied for the purposes of providing various sets of coefficients.

The [CollisionCollection](#) node specifies a set of default coefficients to use for all contacts unless overridden by geometry-specific information. These coefficients are generally described using SFVec2f fields. The 2D vector describes the coefficients for the primary direction for the first value and secondary direction for the second value.

## 37.3 Abstract types

### 37.3.1 X3DNBodyCollidableNode

```
X3DNBodyCollidableNode : X3DChildNode, X3DBoundedObject {
  SFBool [in,out] bboxDisplay FALSE
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFRotation [in,out] rotation 0 0 1 0 [0,1]
  SFVec3f [in,out] translation 0 0 0 (-∞,∞)
  SFBool [in,out] visible TRUE
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}
```

The *X3DNBodyCollidableNode* abstract node type represents objects that act as the interface between the rigid body physics, collision geometry proxy, and renderable objects in the scene graph hierarchy.

The *enabled* field is used to specify whether a collidable object is eligible for collision-detection interactions.

The *translation* and *rotation* fields define an offset from, and rotation about, the body's center that the collidable node occupies. This can be used to place the collidable geometry in a different location relative to the actual rigid body that has the physics model being applied.

### 37.3.2 X3DNBodyCollisionSpaceNode

```
X3DNBodyCollisionSpaceNode : X3DNode, X3DBoundedObject {
  SFBool [in,out] bboxDisplay FALSE
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFBool [in,out] visible TRUE
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}
```

The *X3DNBodyCollisionSpaceNode* abstract node type represents objects that act as a self-contained spatial collection of objects that can interact through collision-detection routines. Different types of spaces may be defined depending on spatial organization or other optimization mechanisms.

The *enabled* field specifies whether the collision space is to be considered during collision processing.

### 37.3.3 X3DRigidJointNode

```
X3DRigidJointNode : X3DNode {
  SFNode [in,out] body1 NULL [RigidBody]
  SFNode [in,out] body2 NULL [RigidBody]
  MFString [in,out] forceOutput "NONE" ["ALL", "NONE", ...]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

The *X3DRigidJointNode* abstract node type is the base type for all joint types.

The *forceOutput* field is used to control which output fields are to be generated for the next frame. In physics models, the amount of data that can be generated per frame can be quite extensive, particularly in complex models with a large number of joints. A typical application will need only a few of them, if any at all. This field is used to control which of those outputs the author requires to be generated. The values of the array are to describe the names, exactly, of the output field(s) that are to be updated at the start of the next frame. Two special values are defined: "ALL" and "NONE". If "ALL" is specified anywhere in the array, all fields are to be updated. If "NONE" is specified, no updates are performed. If the list of values is empty, it shall be treated as if "NONE" were specified. Other values provided in addition to "NONE" shall be ignored.

Because computers are not guaranteed to be accurate in their mathematical calculations and because of the nature of the discrete time steps in the evaluation mechanisms, the behaviour of the system will not be 100% accurate.

EXAMPLE Objects may intersect that should not and joints may break that should not.

Every joint type will have a set of joint-specific fields that define a set of error correction conditions. This error correction conditions provide guidance as to how to automatically correct for internally calculated errors including such errors as object interpenetration. In addition, these error correction conditions can be used to control how quickly the errors should be corrected. Fast corrections may not always be desirable for the appropriate visual output required.

## 37.4 Node reference

### 37.4.1 BallJoint

```
BallJoint : X3DRigidJointNode {
  SFVec3f [in,out] anchorPoint 0 0 0
  SFNode [in,out] body1 NULL [RigidBody]
  SFNode [in,out] body2 NULL [RigidBody]
  MFString [in,out] forceOutput "NONE" ["ALL", "NONE", "...]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFVec3f [out] body1AnchorPoint
  SFVec3f [out] body2AnchorPoint
}
```

The `BallJoint` node represents an unconstrained joint between two bodies that pivot about a common anchor point.

`body1AnchorPoint` and `body2AnchorPoint` represent the output that describes where the `anchorPoint` is relative to the two bodies local coordinate reference frame. This can be used to detect if the joint has caused a separation if the two values are not the same for a given frame.

### 37.4.2 CollidableOffset

```
CollidableOffset : X3DNBodyCollidableNode {
  SFBool [in out] bboxDisplay FALSE
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFRotation [in,out] rotation 0 0 1 0 [0,1]
  SFVec3f [in,out] translation 0 0 0 (-∞,∞)
  SFBool [in out] visible TRUE
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
  SFNode [] collidable NULL [X3DNBodyCollidableNode]
}
```

The `CollidableOffset` node is used to reposition a piece of geometry relative to the center of the owning body while keeping it consistent within the geometry space.

The `collidable` field holds a reference to a single nested item of collidable scene graph. If there are multiple transformation paths to this reference, the results are undefined.

### 37.4.3 CollidableShape

```
CollidableShape : X3DNBodyCollidableNode {
  SFBool [in out] bboxDisplay FALSE
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFRotation [in,out] rotation 0 0 1 0 [0,1]
  SFVec3f [in,out] translation 0 0 0 (-∞,∞)
  SFBool [in out] visible TRUE
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
  SFNode [] shape NULL [Shape]
}
```

The `CollidableShape` node represents the glue between the collision-detection system, the rigid body model, and the renderable scene graph. Its job is to take a single piece of geometry wrapped in a [Shape](#) node and provide a way for the physics model body to move the geometry. In addition, it allows the collision detection system to determine the location of the geometry primitives that it uses for collision management. When placed under a part of the transformation hierarchy, it can be used to visually represent the movement of the object.

The *shape* field uses the geometry proxy for specifying which geometry best represents the collidable object.

NOTE Since the shape node is still writable, it is strongly recommended that the author not dynamically change the Shape's *geometry* field as it may have large performance impacts due to optimizations used by the collision system.

Not all geometry types are mappable to the collision node type.

EXAMPLE PointSet

If the containing shape node is given an explicit bounding box size, the geometry shall be approximated using that shape for the purposes of collision detection. If there is no bounding box, the results are implementation-dependent.

### 37.4.4 CollisionCollection

```
CollisionCollection : X3DChildNode {
  MFString [in,out] appliedParameters    "BOUNCE"
  SFFloat [in,out] bounce                0 [0,1]
  MFNode [in,out] collidables            NULL [X3DNBodyCollisionSpaceNode,
                                           X3DNBodyCollidableNode]
  SFBool [in,out] enabled                 TRUE
  SFVec2f [in,out] frictionCoefficients  0 0 [0,∞)
  SFNode [in,out] metadata                NULL [X3DMetadataObject]
  SFFloat [in,out] minBounceSpeed        0.1 [0,∞)
  SFVec2f [in,out] slipFactors            0 0 (-∞,∞)
  SFFloat [in,out] softnessConstantForceMix 0.0001 [0,1]
  SFFloat [in,out] softnessErrorCorrection 0.8 [0,1]
  SFVec2f [in,out] surfaceSpeed          0 0 (-∞,∞)
}
```

The CollisionCollection node holds a collection of objects in the *collidables* field that can be managed as a single entity for resolution of inter-object collisions with other groups of collidable objects. A group consists of both collidable objects as well as spaces that may be collided against each other. A set of parameters are provided that specify default values that will be assigned to all [Contact](#) nodes generated from the [CollisionSensor](#) node. A user may then override the individual Contact node by inserting a script between the output of the sensor and the input to the [RigidBodyCollection](#) node if it is desired to process the contact stream.

The *enabled* field is used to control whether the collision-detection system for this collection should be run at the end of this frame. A value of `TRUE` enables it while a value of `FALSE` disables it. A CollisionSensor node watching this collection does not report any outputs for this collection for this frame if it is not enabled.

The *bounce* field indicates how bouncy the surface contact is. A value of 0 indicates no bounce at all while a value of 1 indicates maximum bounce.

The *minBounceSpeed* field indicates the minimum speed, in speed base units, that an object shall have before an object will bounce. If the object is below this speed, it will not bounce, effectively having an equivalent value for the *bounce* field of zero.

The *surfaceSpeed* field defines the speed in the two friction directions in speed base units. This is used to indicate if the contact surface is moving independently of the motion of the bodies.

EXAMPLE a conveyor belt.

The *softnessConstantForceMix* value applies a constant force value to make the colliding surfaces appear to be somewhat soft.

The *softnessErrorCorrection* determines how much of the collision error should be fixed in a set of evaluations. The value is limited to the range of [0,1]. A value of 0 specifies no error correction while a value of 1 specifies that all errors should be corrected in a single step.

The *appliedParameters* indicates globally which parameters are to be applied to the collision outputs when passing information into the the rigid body physics system. These parameters specify a series of defaults that apply to all contacts generated. Individual contacts may override which values are applicable, if needed, by setting the field of the same name in the contact itself. The following are valid values:

- "BOUNCE": The bounce field value is used.
- "USER\_FRICTION": The system will normally calculate the friction direction vector that is perpendicular to the contact normal. This setting indicates that the user-supplied value in this contact should be used.
- "FRICTION\_COEFFICIENT-2": The *frictionCoefficients* field values are used.
- "ERROR\_REDUCTION": The *softnessErrorCorrection* field value in the contact evaluation should be used.
- "CONSTANT\_FORCE": The *softnessConstantForceMix* field value in the contact evaluation should be used.
- "SPEED-1": The *surfaceSpeed* field value first component is used.
- "SPEED-2": The *surfaceSpeed* field value second component is used.
- "SLIP-1": The *slipFactors* field value first component is used.
- "SLIP-2": The *slipFactors* field value second component is used.

## 37.4.5 CollisionSensor

```
CollisionSensor : X3DSensorNode {
  SFNode [in,out] collider    NULL [CollisionCollection]
  SFBool [in,out] enabled    TRUE
  SFNode [in,out] metadata    NULL [X3DMetadataObject]
  MFNode [out] intersections [X3DNBodyCollidableNode]
  MFNode [out] contacts      [Contact]
  SFBool [out] isActive
}
```

The CollisionSensor node is used to send collision-detection information into the scene graph for user processing. The collision-detection system does not require an instance of this class to be in the scene in order for it to run or affect the physics model. This class is used to report to the user contact information should the user require this information for other purposes.

The *collidables* field specifies the nodes and spaces that are to be included in collision-detection computations.

The *contacts* field is used to report contacts that were generated as a result of the scene graph changes last frame. This field generates instances of the [Contact](#) node.

**NOTE** While it is possible to route from this field to the *set\_contacts* field of the [RigidBodyCollection](#) node, it is strongly advised that this not be done. The collision system will have already taken these into account internally and processed them in the visual results from the last frame. Setting the values again to the RigidBodyCollection



node will result in undefined behaviour.

The *contacts* field is only available when using the RigidBodyPhysics support level 2 and above.

The CollisionSensor is active (*isActive* is `TRUE`) when contacts were located as a result of the movement of the watched objects from last frame.

The *intersections* field is used to report the colliding geometry that was detected in this last frame.

## 37.4.6 CollisionSpace

```
CollisionSpace : X3DNBodyCollisionSpaceNode {
  MFNode [in,out] collidables NULL [X3DNBodyCollisionSpaceNode, X3DNBodyCollidableNode]
  SFBool [in,out] bboxDisplay FALSE
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFBool [in,out] useGeometry FALSE
  SFBool [in,out] visible TRUE
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}
```

The CollisionSpace node holds a collection of objects in the *collidables* field that can be considered as a single entity for resolution of inter-object collisions with other groups of collidable objects. A group consists of both collidable objects as well as nested collections. This grouping allows creation of efficient collision detection scenarios by grouping functional sets of objects together. Spaces may be collided against each other to determine if the larger group of objects are anywhere near each other. If there is some intersection between two spaces, or between a collidable space and a collidable object, the system will traverse into the contained objects looking for finer resolution on exactly which objects collided together.

The *useGeometry* field indicates whether the collision-detection code should check for collisions down to the level of geometry or only make approximations using the bounds of the geometry. Using the geometry will be more accurate but slower. In most cases, just testing against the bounds of the object is sufficient.

## 37.4.7 Contact

```
Contact : X3DNode {
  MFString [in,out] appliedParameters "BOUNCE"
  SFNode [in,out] body1 NULL [RigidBody]
  SFNode [in,out] body2 NULL [RigidBody]
  SFFloat [in,out] bounce 0 [0,1]
  SFVec3f [in,out] contactNormal 0 1 0 (-∞,∞)
  SFFloat [in,out] depth 0 (-∞,∞)
  SFVec2f [in,out] frictionCoefficients 0 0 [0,∞)
  SFVec3f [in,out] frictionDirection 0 1 0 (-∞,∞)
  SFNode [in,out] geometry1 NULL [X3DNBodyCollidableNode]
  SFNode [in,out] geometry2 NULL [X3DNBodyCollidableNode]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFFloat [in,out] minbounceSpeed 0 [0,∞)
  SFVec3f [in,out] position 0 0 0 (-∞,∞)
  SFVec2f [in,out] slipCoefficients 0 0 (-∞,∞)
  SFFloat [in,out] softnessConstantForceMix 0.0001 [0,1]
  SFFloat [in,out] softnessErrorCorrection 0.8 [0,1]
  SFVec2f [in,out] surfaceSpeed 0 0 (-∞,∞)
}
```

The Contact node specifies information concerning a contact between collidable objects and/or spaces.

The *body1* and *body2* fields specify two top-level nodes that should be evaluated in the

physics model as a single set of interactions with respect to each other.

The *geometry1* and *geometry2* fields specify information about *body1* and *body2*.

The *position* field indicates the exact location of the contact that was made between the two objects.

The *contactNormal* field is a unit vector describing the normal between the two colliding bodies.

The *depth* field indicates how deep the current intersection is along the normal vector.

The *frictionDirection* field is used to control the vector that describes which way friction is to be applied to the contact location. If there is no friction, the direction should be set to 0, 0, 0.

The *bounce* field indicates how bouncy the surface contact is. A value of 0 indicates no bounce at all while a value of 1 indicates maximum bounce.

The *minBounceSpeed* field indicates the minimum speed, in speed base units, that an object shall have before an object will bounce. If the object is below this speed, it will not bounce, effectively having an equivalent value for the *bounce* field of zero.

The *surfaceSpeed* field defines the speed in the two friction directions in speed base units. This is used to indicate whether the contact surface is moving independently of the motion of the bodies.

**EXAMPLE** A conveyor belt mechanism may be stationary while its belt is moving. The object being placed on the conveyor belt will not be affected by the motion of the belt until it is in contact with it.

The *softnessConstantForceMix* value applies a constant force value to make the colliding surfaces appear to be somewhat soft.

The *softnessErrorCorrection* determines how much of the collision error should be fixed in a set of evaluations. The value is limited to the range of [0,1] where 0 specifies no error correction while a value of 1 specifies that all errors should be corrected in a single step.

The *appliedParameters* indicates globally which parameters are to be applied to the collision outputs when passing information into the the rigid body physics system. These parameters specify a series of defaults that apply to all contacts generated. Individual contacts may override which values are applicable, if needed, by setting the field of the same name in the contact itself. The valid values are specified in [Table 37.2](#):

**Table 37.2 — appliedParameters valid values**

Value	Meaning
"BOUNCE"	The bounce field value is used.
"USER_FRICTION"	The system will normally calculate the friction direction vector that is perpendicular to the contact normal. This setting indicates that the user-supplied value in this contact



	should be used.
"FRICTION_COEFFICIENT-2"	The <i>frictionCoefficients</i> field values are used.
"ERROR_REDUCTION"	The <i>softnessErrorCorrection</i> field value in the contact evaluation should be used.
"CONSTANT_FORCE"	The <i>softnessConstantForceMix</i> field value in the contact evaluation should be used.
"SPEED-1"	The <i>surfaceSpeed</i> field value first component is used.
"SPEED-2"	The <i>surfaceSpeed</i> field value second component is used.
"SLIP-1"	The <i>slipFactors</i> field value first component is used.
"SLIP-2"	The <i>slipFactors</i> field value second component is used.

## 37.4.8 DoubleAxisHingeJoint

```

DoubleAxisHingeJoint : X3DRigidJointNode {
  SFVec3f [in,out] anchorPoint      0 0 0
  SFVec3f [in,out] axis1           0 0 0
  SFVec3f [in,out] axis2           0 0 0
  SFNode [in,out] body1            NULL [RigidBody]
  SFNode [in,out] body2            NULL [RigidBody]
  SFFloat [in,out] desiredAngularVelocity1 0 (-∞,∞)
  SFFloat [in,out] desiredAngularVelocity2 0 (-∞,∞)
  MFString [in,out] forceOutput    "NONE" ["ALL", "NONE", ...]
  SFFloat [in,out] maxAngle1       π [-π,π]
  SFFloat [in,out] maxTorque1      0 (-∞,∞)
  SFFloat [in,out] maxTorque2      0 (-∞,∞)
  SFNode [in,out] metadata         NULL [X3DMetadataObject]
  SFFloat [in,out] minAngle1       -π [-π,π]
  SFFloat [in,out] stopBounce1     0 [0,1]
  SFFloat [in,out] stopConstantForceMix1 0.001 [0,∞)
  SFFloat [in,out] stopErrorCorrection1 0.8 [0,1]
  SFFloat [in,out] suspensionErrorCorrection 0.8 [0,1]
  SFFloat [in,out] suspensionForce 0 [0,∞)
  SFVec3f [out] body1AnchorPoint
  SFVec3f [out] body1Axis
  SFVec3f [out] body2AnchorPoint
  SFVec3f [out] body2Axis
  SFFloat [out] hinge1Angle
  SFFloat [out] hinge1AngleRate
  SFFloat [out] hinge2Angle
  SFFloat [out] hinge2AngleRate
}

```

The `DoubleAxisHingeJoint` node represents a joint that has two independent axes that are located around a common anchor point. Axis 1 is specified relative to the first body (specified by the *body1* field) and axis 2 is specified relative to the second body (specified by the *body2* field). Axis 1 can have limits and a motor, axis 2 can only have a motor.

The *minAngle1* and *maxAngle1* fields are used to control the maximum angles through which the hinge is allowed to travel. A hinge may not travel more than  $\pi$  radians (or the equivalent angle base units) in either direction from its initial position.

The *stopBounce1* field is used to set how bouncy the minimum and maximum angle stops are for axis 1. A value of zero means they are not bouncy while a value of 1 means maximum bounciness (full reflection of force arriving at the stop).

The *stopErrorCorrection1* and *suspensionErrorCorrection* fields describe how quickly the

system should resolve intersection errors due to floating point inaccuracies. This value ranges between 0 and 1. A value of 0 means no correction at all while a value of 1 indicates that all errors should be corrected in a single step.

The *stopConstantForceMix1* and *suspensionForce* fields can be used to apply damping to the calculations by violating the normal constraints by applying a small, constant force to those calculations. This allows joints and bodies to be a fraction springy, as well as helping to eliminate numerical instability. The larger the value, the more soft each of the constraints being evaluated. A value of zero indicates hard constraints so that everything is exactly honoured. By combining the *stopErrorCorrection1* and *stopConstantForceMix1* fields and/or the *suspensionErrorCorrection* and *suspensionForce* fields, various effects, such as spring-driven or spongy connections, can be emulated.

The *maxTorque1* field defines the maximum amount of torque that the motor can apply on axis 1 in order to achieve the *desiredAngularVelocity1* value. Similarly, *maxTorque2* controls the maximum amount of torque to achieve *desiredAngularVelocity2* on axis 2.

The *hingeXAngle* output fields report the current relative angle between the two bodies in angle base units and the *hingeXAngleRate* field describes the rate at which that angle is currently changing in **angular\_rate angular velocity** base units.

The body anchor point and body axis output fields report the current location of the anchor point relative to the corresponding body. This can be used to determine if the joint has broken.

### 37.4.9 MotorJoint

```
MotorJoint : X3DRigidJointNode { SFFloat [in,out] axis1Angle 0 [-π,π] SFFloat [in,out]
axis1Torque 0 (-∞,∞) SFFloat [in,out] axis2Angle 0 [-π,π] SFFloat [in,out] axis2Torque
0 (-∞,∞) SFFloat [in,out] axis3Angle 0 [-π,π] SFFloat [in,out] axis3Torque 0 (-∞,∞)
SFNode [in,out] body1 NULL [RigidBody] SFNode [in,out] body2 NULL [RigidBody]
SFInt32 [in,out] enabledAxes 1 [0,3] MFString [in,out] forceOutput "NONE"
["ALL","NONE",...] SFNode [in,out] metadata NULL [X3DMetadataObject] SFVec3f
[in,out] motor1Axis 0 0 0 SFVec3f [in,out] motor2Axis 0 0 0 SFVec3f [in,out]
motor3Axis 0 0 0 SFFloat [in,out] stop1Bounce 0 [0,1] SFFloat [in,out]
stop1ErrorCorrection 0.8 [0,1] SFFloat [in,out] stop2Bounce 0 [0,1] SFFloat [in,out]
stop2ErrorCorrection 0.8 [0,1] SFFloat [in,out] stop3Bounce 0 [0,1] SFFloat [in,out]
stop3ErrorCorrection 0.8 [0,1] SFFloat [out] motor1Angle SFFloat [out]
motor1AngleRate SFFloat [out] motor2Angle SFFloat [out] motor2AngleRate SFFloat
[out] motor3Angle SFFloat [out] motor3AngleRate SFBool [] autoCalc FALSE }
```

The *MotorJoint* node allows control of the relative angular velocities between the two bodies (specified by the *body1* and *body2* fields) associated with a joint. This can be especially useful with a [BallJoint](#) where there is no restriction on the angular degrees of freedom.

The *autoCalc* field is used to control whether the user shall manually provide the individual angle rotations each frame or if they are to be automatically calculated from the motor's implementation.

The *motorAxis* fields define the axis vector of the corresponding axis. If the value is 0,

0, 0, the corresponding axis is disabled and the motor does not apply a force or torque along that axis. The *motorAxis1* field is anchored to the global frame. The *motorAxis2* field is anchored to *body1*'s frame of reference, and the *motorAxis3* field is anchored to *body2*'s frame of reference.

The three axis angle fields provide angles (in angle base units) for this frame for the corresponding motor axis when in user-calculated mode.

When the *autoCalc* field is set to `FALSE`, the *enabledAxes* field indicates how many axes can currently be controlled and modified. If the value is zero, the motor is effectively disabled. If the value is 1, only axis1 is enabled, a value of 2 has axis 1 and axis 2 enabled and a value of 3 has all axes enabled.

The motor angle output fields provide the calculated angle in angle base units for this motor joint from the last frame. The motor angle rate output fields describe the rate, in **angular\_rate angular velocity** base units, that the motor is turning.

The stop bounce fields describe how much the joint should bounce the body back on the corresponding axis if the joint limit has been reached or exceeded. A value of zero indicates no bounce at all, and a value of one says that it should bounce with velocity equal and opposite to the collision velocity of the contact.

The stop error correction fields describe the amount of error correction to be performed in a time step when the joint reaches the limit on the corresponding axis. A value of zero means no error correction is to be performed and a value of one means all error should be corrected in a single step.

## 37.4.10 RigidBody

```
RigidBody : X3DNode {
  SFFloat [in,out] angularDampingFactor 0.001 [0,1]
  SFVec3f [in,out] angularVelocity 0 0 0 (-∞,∞)
  SFBool [in,out] autoDamp FALSE
  SFBool [in,out] autoDisable FALSE
  SFVec3f [in,out] centerOfMass 0 0 0 (-∞,∞)
  SFFloat [in,out] disableAngularSpeed 0 [0,∞)
  SFFloat [in,out] disableLinearSpeed 0 [0,∞)
  SFFloat [in,out] disableTime 0 [0,∞)
  SFTime [in,out] disableTime 0 [0,∞)
  SFBool [in,out] enabled TRUE
  SFVec3f [in,out] finiteRotationAxis 0 0 0 [-1,1]
  SFBool [in,out] fixed FALSE
  MFVec3f [in,out] forces []
  MFNode [in,out] geometry [] [X3DNBodyCollidableNode]
  SFMatrix3f [in,out] inertia 1 0 0
    0 1 0
    0 0 1
  SFFloat [in,out] linearDampingFactor 0.001 [0,1]
  SFVec3f [in,out] linearVelocity 0 0 0 (-∞,∞)
  SFFloat [in,out] mass 1 (0,∞)
  SFNode [in,out] massDensityModel NULL [Sphere, Box, Cone]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFRotation [in,out] orientation 0 0 1 0 [0,1]
  SFVec3f [in,out] position 0 0 0 (-∞,∞)
  MFVec3f [in,out] torques []
  SFBool [in,out] useFiniteRotation FALSE
  SFBool [in,out] useGlobalGravity TRUE
}
```

The *RigidBody* node describes a body and its properties that can be affected by the physics model. A body is modelled as a collection of shapes that describe mass distribution rather than renderable geometry. Bodies are connected together using Joints and are represented by geometry.

The *geometry* field is used to connect the body modelled by the physics engine

implementation to the real geometry of the scene through the use of collidable nodes. This allows the geometry to be connected directly to the physics model as well as collision detection. Collidable nodes have their location set to the same location as the body instance in which they are located. Their position and location are not relative to this object, unless otherwise defined.

The *massDensityModel* field is used to describe the geometry type and dimensions used to calculate the mass density in the physics model. This geometry has no renderable property, other than for defining the model of the mass density. It is not rendered, nor modified by the physics model.

The *finiteRotationAxis* field specifies a vector around which the object rotates.

The *useFiniteRotation* field is used to influence the way the body's rotation is calculated. In very fast rotating objects, such as a wheel of a car, an infinitely small time step can cause the modelling to explode. The default value is to use the faster infinite mode. Setting the field value to `TRUE` uses the finite calculation model. Using the finite model is more costly to compute but will be more accurate for high rotation speed bodies.

The *useGlobalGravity* field is used to indicate whether this particular body should be influenced by the containing [RigidBodyCollection](#)'s *gravity* setting. A value of `TRUE` indicates that the gravity is used, a value of `FALSE` indicates that it is not used. This only applies to this body instance. Contained sub-bodies shall not be affected by this setting.

The *inertia* field represents a 3x2 inertia tensor matrix. If the set values are less than six items, the results are implementation dependent. If the value set is greater than six values, only the first six values of the array are used.

The *fixed* field is used to indicate that this body does not move. Any calculations involving collisions with this body should take into account that this body does not move. This is useful for representing objects such as the ground, walls etc that can be collided with, have an effect on other objects, but are not capable of moving themselves.

The *mass* field indicates the mass of the body in mass base units. All bodies shall have a non-zero mass, with the default value of 1 mass base unit.

The damping factor fields allow the user to instruct the implementation to automatically damp the motion of the body over time. The value of the field is used to take a multiple of the value calculated in the last frame and apply it in opposition to the current motion for this frame. Damping is useful to provide an appearance of frictional forces and also to prevent the body from exploding due to numerical instability of the physics model calculations. Damping is proportional to the current velocity and/or rotation of the object. The application of damping is controlled through the use of the *autoDamp* field. When the value is `FALSE`, no damping is applied. When the value is `TRUE`, rotational and translational damping is calculated and applied.

**EXAMPLE** The body is calculated in the previous frame to have a velocity of (0 1 0). A damping factor of 0.01 is active. In this next simulation time step, a force of  $0.01 \times (0 \ 1 \ 0) \times -1$  is applied to the object.

The *torques* and *forces* fields define zero or more sets of torque and force values that are applied to the object every frame. These are continuously applied until reset to zero

by the user.

The velocity fields are used to provide a constant velocity value to the object every frame. If both forces and velocity are defined, the velocity is used only on the first frame that the node is active, and then the forces are applied. The velocity fields then report the changed values as a result of the application of the physics model in each frame. Setting a new value to the appropriate field will reset the body's velocity for the next frame. Caution should be used in doing this as the underlying physics models may assume some amount of caching between time step evaluations and instantaneous velocity changes may lead to numerical instability.

The *position* and *orientation* fields are used to set the initial conditions of this body's location in world space. After the initial conditions have been set, these fields are used to report the current information based on the most recent physics model evaluation. Setting new values will cause the objects to be moved to the new location and orientation for the start of the next evaluation cycle. Care should be used in manually changing the *position* and *orientation* as the underlying physics models may cache information between time step evaluations and sudden instantaneous changes may lead to numerical instability.

The disable fields define conditions for when the body ceases to be considered as part of the rigid body calculations and should be considered as at rest. Due to the numerical instability of physics models, even bodies initially declared to be at rest may gain some amount of movement, even when not effected by an external force. These values define tolerances for which the physics model should start to ignore this object in any calculation, thus resulting in them being actually at rest and not subject to these instability conditions. Once any one of these values is achieved, the body is considered as being at rest unless acted upon by an external force (e. g., collision or action of connected joint). By default, this automatic disabling is turned off. It may be enabled by setting the *autoDisable* field to `TRUE`.

The *enabled* field controls whether the information in this node is submitted to the physics engine for processing. If the *enabled* field is set `TRUE`, the node is submitted to the physics engine. If the *enabled* field is set `FALSE`, the node is not submitted to the physics engine for processing.

### 37.4.11 RigidBodyCollection

```
RigidBodyCollection : X3DChildNode {
  MFNode [in]  set_contacts          [Contact]
  SFBool [in,out] autoDisable        FALSE
  MFNode [in,out] bodies             [] [RigidBody]
  SFFloat [in,out] constantForceMix  0.0001 [0,∞)
  SFFloat [in,out] contactSurfaceThickness 0 [0,∞)
  SFFloat [in,out] disableAngularSpeed 0 [0,∞)
  SFFloat [in,out] disableLinearSpeed 0 [0,∞)
  SFFloat [in,out] disableTime       0 [0,∞)
  SFTime [in,out] disableTime        0 [0,∞)
  SFBool [in,out] enabled            TRUE
  SFFloat [in,out] errorCorrection    0.8 [0,1]
  SFVec3f [in,out] gravity            0 -9.8 0
  SFInt32 [in,out] iterations         10 [0,∞)
  MFNode [in,out] joints             [] [X3DRigidJointNode]
  SFFloat [in,out] maxCorrectionSpeed -1 [0,∞) or -1
  SFNode [in,out] metadata           NULL [X3DMetadataObject]
  SFBool [in,out] preferAccuracy     FALSE
  SFNode [] collider                 NULL [CollisionCollection]
}
```

The *RigidBodyCollection* node represents a system of bodies that will interact within a single physics model. The collection is not a renderable part of the scene graph nor are

its children as a typical model may need to represent the geometry for physics separately, and in less detail, than those needed for visuals.

The *bodies* field contains a collection of the top-level nodes that comprise a set of bodies that should be evaluated as a single set of interactions.

The *joints* field is used to register all the joints between the bodies contained in this collection. If a joint is connected between bodies in two different collections, the result is implementation-dependent. If a joint instance is registered with more than one collection, the results are implementation dependent. Joints not registered with any collection are not evaluated.

The *enabled* field is used to control whether the physics model for this collection should be run this frame.

The *contactSurfaceThickness* field represents how far bodies may interpenetrate after a collision. This allows simulation of softer bodies that may deform somewhat during collision. The default value is zero.

NOTE Since a value of 0 may cause jittering due to floating point inaccuracy, a typically small value of 0.001 length base units may be useful.

The *gravity* field indicates direction and strength (in acceleration base units) of the local gravity vector for this collection of bodies. The default gravity is standard earth gravity of 9.8 meters/second<sup>2</sup> downwards.

The *set\_contacts* input field is used to provide per-frame sets of information about contacts between bodies in this frame. These contacts are then used to modify the location of the bodies within the scene graph when the physics model is evaluated at the end of the frame. For efficiency, a user may reuse instances of the Contact node for each frame rather than allocating a new instance per frame. A browser implementation shall not make assumptions about the same object instance having the same values each frame.

The *preferAccuracy* field is used to provide a performance hint to the underlying evaluation about whether the user prefers to have very accurate models or fast models. Accuracy comes at a large penalty in both speed and memory usage, but may not be needed most of the time. The default setting is to optimize for speed rather than accuracy.

The *iterations* field is used to control how many iterations over the collections of joints and bodies are to be performed each time the model is evaluated. Rigid body physics is a process of iterative refinement in order to maintain reasonable performance. As the number of iterations grow, the more stable the final results are at the cost of increasing evaluation time. Since maintaining real-time performance is a trade off between accuracy and frame rate, this setting allows the user to control that trade off to a limited extent.

The *errorCorrection* field describes how quickly the system should resolve intersection errors due to floating point inaccuracies. This value ranges between 0 and 1. A value of 0 means no correction at all while a value of 1 indicates that all errors should be corrected in a single step.



The *constantForceMix* field can be used to apply damping to the calculations by violating the normal constraints by applying a small, constant force to those calculations. This allows joints and bodies to be a fraction springy, as well as helping to eliminate numerical instability. The larger the value, the more soft each of the constraints being evaluated. A value of zero indicates hard constraints so that everything is exactly honoured. By combining the *errorCorrection* and *constantForceMix* fields, various effects, such as spring-driven or spongy connections, can be emulated.

The *collider* field associates a collision collection with this rigid body collection allowing seamless updates and integration without the need to use the X3D event model.

The disable fields define conditions for when the body ceases to be considered as part of the rigid body calculations and should be considered as at rest. Due to the numerical instability of physics models, even bodies initially declared to be at rest may gain some amount of movement, even when not effected by an external forces. These values define tolerances for which the physics model should start to ignore this object in any calculation, thus resulting in them being actually at rest and not subject to these instability conditions. Once any one of these values is achieved, the body is considered as being at rest, unless acted upon by an external force (e. g., collision or action of connected joint). By default, this automatic disabling is turned off. It may be enabled by setting the *autoDisable* field to `TRUE`.

## 37.4.12 SingleAxisHingeJoint

```
SingleAxisHingeJoint : X3DRigidJointNode {
  SFVec3f [in,out] anchorPoint 0 0 0
  SFVec3f [in,out] axis 0 0 0
  SFNode [in,out] body1 NULL [RigidBody]
  SFNode [in,out] body2 NULL [RigidBody]
  MFString [in,out] forceOutput "NONE" ["ALL","NONE",...]
  SFFloat [in,out] maxAngle  $\pi$ 
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFFloat [in,out] minAngle  $-\pi$ 
  SFFloat [in,out] stopBounce 0 [0,1]
  SFFloat [in,out] stopErrorCorrection 0.8 [0,1]
  SFFloat [out] angle
  SFFloat [out] angleRate
  SFVec3f [out] body1AnchorPoint
  SFVec3f [out] body2AnchorPoint
}
```

This node represents a joint with a single axis about which to rotate. As the name suggests, this is a joint that works like a traditional door hinge. The axis of the hinge is defined to be along the unit vector described in the *axis* field and centered on the *anchorPoint* described in world coordinates. The objects on each side of the hinge are specified by the *body1* and *body2* fields.

The *minAngle* and *maxAngle* fields are used to control the maximum angles through which the hinge is allowed to travel. A hinge may not travel more than  $n$  radians (or the equivalent angle base units) in either direction from its initial position.

The *stopBounce* field describes how much the joint should bounce the body back if the joint limit has been reached or exceeded. A value of zero indicates no bounce at all, and a value of one says that it should bounce with velocity equal and opposite to the collision velocity of the contact.

The *stopErrorCorrection* field describes the amount of error correction to be performed in a time step when the joint reaches the limit. A value of zero means no error

correction is to be performed and a value of one means all error should be corrected in a single step.

The *angle* output field reports the current relative angle between the two bodies in angle base units and the *angleRate* field describes the rate at which that angle is currently changing in **angular\_rate** **angular velocity** base units.

The body anchor point output fields report the current location of the anchor point relative to the corresponding body. This can be used to determine if the joint has broken.

### 37.4.13 SliderJoint

```
SliderJoint : X3DRigidBodyNode {
  SFVec3f [in,out] axis          0 1 0
  SFNode [in,out] body1         NULL [RigidBody]
  SFNode [in,out] body2         NULL [RigidBody]
  MFString [in,out] forceOutput "NONE" ["ALL","NONE",...]
  SFFloat [in,out] maxSeparation 1 [0,∞)
  SFNode [in,out] metadata      NULL [X3DMetadataObject]
  SFFloat [in,out] minSeparation 0 [0,∞)
  SFFloat [in,out] sliderForce  0 [-∞,∞)
  SFFloat [in,out] stopBounce   0 [0,1]
  SFFloat [in,out] stopErrorCorrection 1 [0,1]
  SFFloat [out] separation
  SFFloat [out] separationRate
}
```

The *SliderJoint* node represents a joint where all movement between the bodies specified by the *body1* and *body2* fields is constrained to a single dimension along a user-defined axis.

The *axis* field indicates which axis along which the two bodies will act. The value should represent a normalized vector.

The *sliderForce* field value is used to apply a force (specified in force base units) along the axis of the slider in equal and opposite directions to the two bodies. A positive value applies a force such that the two bodies accelerate away from each other while a negative value applies a force such that the two bodies accelerate toward each other.

If *minSeparation* is greater than *maxSeparation*, the stops become ineffective as if the object has no stops at all.

The *separation* output field is used to indicate the final separation of the two bodies.

The *separationRate* output field is used to indicate the change in separation over time since the last update.

The *stopBounce* field describes how much the joint should bounce the body back if the joint limit has been reached or exceeded. A value of zero indicates no bounce at all, and a value of one indicates that it should bounce with velocity equal and opposite to the collision velocity of the contact.

The *stopErrorCorrection* field describes the amount of error correction to be performed in a time step when the joint reaches the limit. A value of zero means no error correction is to be performed and a value of one means all error should be corrected in a single step.

### 37.4.14 UniversalJoint



```

UniversalJoint : X3DRigidJointNode {
  SFVec3f [in,out] anchorPoint 0 0 0
  SFVec3f [in,out] axis1 0 0 0
  SFVec3f [in,out] axis2 0 0 0
  SFNode [in,out] body1 NULL [RigidBody]
  SFNode [in,out] body2 NULL [RigidBody]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFString [in,out] forceOutput "NONE" ["ALL","NONE",...]
  SFFloat [in,out] stopBounce1 0 [0,1]
  SFFloat [in,out] stop1ErrorCorrection 0.8 [0,1]
  SFFloat [in,out] stop2Bounce 0 [0,1]
  SFFloat [in,out] stop2ErrorCorrection 0.8 [0,1]
  SFVec3f [out] body1AnchorPoint
  SFVec3f [out] body1Axis
  SFVec3f [out] body2AnchorPoint
  SFVec3f [out] body2Axis
}
    
```

A universal joint is like a `BallJoint` that constrains an extra degree of rotational freedom. Given the axis specified by the `axis1` field on the body specified by the `body1` field, and the axis specified by the `axis2` field on `body2` that is perpendicular to `axis1`, the `UniversalJoint` node keeps the axes perpendicular to each other. Thus, rotation of the two bodies about the direction perpendicular to the two axes will be equal.

The vectors specified by the `axis1` and `axis2` fields shall be perpendicular. If not, the interactions are undefined.

The stop bounce fields describe how much the joint should bounce the body back on the corresponding axis if the joint limit has been reached or exceeded. A value of zero indicates no bounce at all, and a value of one indicates that it should bounce with velocity equal and opposite to the collision velocity of the contact.

The stop error correction fields describe the amount of error correction to be performed in a time step when the joint reaches the limit on the corresponding axis. A value of zero means no error correction is to be performed and a value of one means all error should be corrected in a single step.

The body anchor point and body axis output fields report the current location of the anchor point relative to the corresponding body. This can be used to determine if the joint has broken.

## 37.5 Support levels

The Rigid Body Physics component defines two levels of support as specified in [Table 37.3](#).

**Table 37.3 — Rigid body physics component support levels**

Level	Prerequisites	Nodes/Features	Support
1	Core 1 Grouping 1 Shape 1 Geometry3D 1		
		<code>X3DNBodyCollidableNode</code>	n/a
		<code>X3DNBodyCollisionSpaceNode</code>	n/a

		CollidableOffset	All fields fully supported.
		CollidableShape	All fields fully supported.
		CollisionCollection	All fields fully supported.
		CollisionSensor	All fields fully supported except <i>contacts_changed</i> .
		CollisionSpace	All fields fully supported.
<b>2</b>	Core 1 Grouping 1 Shape 1 Geometry3D 1		
		<i>X3DRigidJointNode</i>	n/a
		BallJoint	All fields fully supported.
		CollisionSensor	All fields fully supported.
		Contact	All fields fully supported.
		DoubleAxisHingeJoint	All fields fully supported.
		MotorJoint	All fields fully supported.
		RigidBody	All fields fully supported.
		RigidBodyCollection	All fields fully supported.
		SingleAxisHingeJoint	All fields fully supported.
		SliderJoint	All fields fully supported.
		UniversalJoint	All fields fully supported.





## Extensible 3D (X3D) Part 1: Architecture and bases

### Profile index



#### General

This index lists the profiles in alphabetical order.

Profile	Annex
<a href="#">CADInterchange</a>	H
<a href="#">Core</a>	A
<a href="#">Full</a>	F
<a href="#">Immersive</a>	E
<a href="#">Interactive</a>	C
<a href="#">Interchange</a>	B
<a href="#">MedicalInterchange</a>	L
<a href="#">MPEG-4 interactive</a>	D





## Extensible 3D (X3D) Part 1: Architecture and base components

### 17 Lighting component

---

#### 17.1 Introduction

##### 17.1.1 Name

The name of this component is "Lighting". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 17.1.2 Overview

This clause describes the Lighting component of this part of ISO/IEC 19775. This includes how light sources are defined and positioned as well as how lights effect the rendered image. [Table 17.1](#) provides links to the major topics in this clause.

**Table 17.1 — Topics**

- |   |
|---|
| <ul style="list-style-type: none"> <li>• <a href="#">17.1 Introduction</a> <ul style="list-style-type: none"> <li>◦ <a href="#">17.1.1 Name</a></li> <li>◦ <a href="#">17.1.2 Overview</a></li> </ul> </li> <li>• <a href="#">17.2 Concepts</a> <ul style="list-style-type: none"> <li>◦ <a href="#">17.2.1 Light source semantics</a> <ul style="list-style-type: none"> <li>▪ <a href="#">17.2.1.1 Overview</a></li> <li>▪ <a href="#">17.2.1.2 Scoping of lights</a></li> </ul> </li> <li>◦ <a href="#">17.2.2 Lighting model</a> <ul style="list-style-type: none"> <li>▪ <a href="#">17.2.2.1 Introduction</a></li> <li>▪ <a href="#">17.2.2.2 Lighting 'off'</a></li> <li>▪ <a href="#">17.2.2.3 Lighting 'on'</a></li> <li>▪ <a href="#">17.2.2.4 Lighting equations</a></li> <li>▪ <a href="#">17.2.2.5 References</a></li> </ul> </li> </ul> </li> </ul> |
| <ul style="list-style-type: none"> <li>▪ <a href="#">17.2.2.1 Introduction</a></li> <li>▪ <a href="#">17.2.2.2 Texture sampling</a></li> <li>▪ <a href="#">17.2.2.3 Common definitions for all lighting models</a></li> <li>▪ <a href="#">17.2.2.4 Unlit lighting model</a></li> </ul>  |

#### 17.2.2.5 Phong lighting model

- [17.2.2.6 Physical lighting model](#)
- [17.2.2.7 References](#)
- [17.2.2.8 Gouraud shading](#)

- [17.3 Abstract types](#)
  - [17.3.1 X3DLightNode](#)
- [17.4 Node reference](#)
  - [17.4.1 DirectionalLight](#)
  - [17.4.2 PointLight](#)
  - [17.4.3 SpotLight](#)
- [17.5 Support levels](#)
- [Figure 17.1 — SpotLight node](#)
- [Table 17.1 — Topics](#)
- [Table 17.2 — Unlit colour and alpha mapping](#)
- [Table 17.3 — Lit colour and alpha mapping](#)
- [Table 17.4 — Calculation of the spotlight factor](#)
- [Table 17.5 — Calculation of the fog interpolant](#)
- [Table 17.6 — Lighting component support levels](#)

## 17.2 Concepts

### 17.2.1 Light source semantics

#### 17.2.1.1 Overview

The following node types are light source nodes:

- [DirectionalLight](#)
- [PointLight](#)
- [SpotLight](#)

[PointLight](#) and [SpotLight](#) illuminate all objects in the world that fall within their volume of lighting influence regardless of location within the transformation hierarchy (by default, when their *global* field is `TRUE`). [PointLight](#) defines this volume of influence as a sphere centred at the light (defined by a radius). [SpotLight](#) defines the volume of influence as a solid angle defined by a radius and a cut-off angle. [DirectionalLight](#) nodes illuminate only the objects descended from the light's parent grouping node, including any descendent children of the parent grouping nodes (by default, when their *global* field is `FALSE`).

Shape nodes are illuminated by the sum of all of the lights in the world that affect them. This includes the contribution of both the direct and ambient illumination from light sources. Ambient illumination results from the scattering and reflection of light originally emitted directly by light sources. The amount of ambient light is associated with the individual lights in the scene. This is a gross approximation to how ambient reflection actually occurs in nature.

Any node used as a source of illumination is derived from [X3DLightNode](#). All light sources contain an *intensity*, a *color*, and an *ambientIntensity* field. The *intensity* field specifies the brightness of the direct emission from the light, and the *ambientIntensity* specifies the intensity of the ambient emission from the light. Light intensity may range from 0.0 (no light emission) to **1.0 (full intensity)infinity**. The *color* field specifies the spectral colour properties of both the direct and ambient light emission as an RGB value. The *on* field specifies whether the light is enabled or disabled. If the value is `FALSE`, the light is disabled and will not affect any nodes in the scene. If the value is `TRUE`, the light will affect other nodes according to the [17.2.1.2 Scoping of lights](#).

In the *physical lighting model* (see [17.2.2.6 Physical lighting model](#)) the *intensity* value should correspond to:

- luminous intensity in candela (lm/sr) in case of [PointLight](#) and [SpotLight](#).
- illuminance in lux (lm/m<sup>2</sup>) in case of [DirectionalLight](#).

In the physical lighting model, the *ambientIntensity* value is unused. Future specification versions may introduce a use for it, therefore we recommend leaving it at 0 (default) in case of physical rendering, to avoid future changes.

In case of the *unlit lighting model* all lights are ignored. See the [17.2.2.4 Unlit lighting model](#).

### 17.2.1.2 Scoping of lights

Each light type defines a *global* field that determines whether the light is global or scoped. Global lights illuminate all objects that fall within their volume of lighting influence. Scoped lights only illuminate objects that are in the same transformation hierarchy as the light; *i.e.*, only the children and descendants of its enclosing parent group are illuminated. This allows the creation of realistic effects such as lights that illuminate a single room.

## 17.2.2 Lighting model

### 17.2.2.1 Introduction

The X3D lighting model provides detailed equations that specify the colours to apply to each geometric object. For each object, the values of the [Material](#), *color*, and/or texture currently being applied to the object are combined with the lights illuminating the object and the currently bound [X3DFogObject](#) (if specified). These equations are designed to simulate the physical properties of light striking a surface.

If a programmable shader is defined for an [Appearance](#) node, the lighting model shall be disabled and replaced by the functionality implemented by the shader program. See [31 Programmable shaders component](#) for more information.

### 17.2.2.2 Lighting 'off'

A [Shape](#) node is unlit if either of the following is true:

- a. The shape's *appearance* field is `NULL` (default).
- b. The *material* field in the Appearance node is `NULL` (default).

NOTE Geometry nodes that represent lines or points do not support lighting.

If the shape is unlit, the colour ( $I_{rgb}$ ) and alpha ( $A$ , 1 – transparency) of the shape at each point on the shape's geometry is specified in [Table 17.2](#).

**Table 17.2 — Unlit colour and alpha mapping**

Texture type	Colour per-vertex or per-face	Colour NULL
No texture	$I_{rgb} = I_{C_{rgb}}$ $A = 1$	$I_{rgb} = (1, 1, 1)$ $A = 1$
Intensity (one-component)	$I_{rgb} = I_T \times I_{C_{rgb}}$ $A = 1$	$I_{rgb} = (I_T, I_T, I_T)$ $A = 1$
Intensity+Alpha (two-component)	$I_{rgb} = I_T \times I_{C_{rgb}}$ $A = A_T$	$I_{rgb} = (I_T, I_T, I_T)$ $A = A_T$
RGB (three-component)	$I_{rgb} = I_{T_{rgb}}$ $A = 1$	$I_{rgb} = I_{T_{rgb}}$ $A = 1$
RGBA (four-component)	$I_{rgb} = I_{T_{rgb}}$ $A = A_T$	$I_{rgb} = I_{T_{rgb}}$ $A = A_T$

where:

$A_T$  = normalized [0, 1] alpha value from 2 or 4 component texture image

$I_{C_{rgb}}$  = interpolated per-vertex colour, or per-face colour, from Color node

$I_T$  = normalized [0, 1] intensity from 1 or 2 component texture image

$I_{T_{rgb}}$  = colour from 3-4 component texture image

### 17.2.2.3 Lighting 'on'

If the [Shape](#) node is lit (*i.e.*, a [Material](#) and an [Appearance](#) node are specified for the Shape), the Material and Texture nodes determine the diffuse colour for the lighting equation as specified in [Table 17.3](#).

The Material's `diffuseColor` field modulates the color in the texture. Hence, a `diffuseColor` of white will result in the pure color of the texture, while a `diffuseColor` of black will result in a black diffuse factor regardless of the texture.

The Material's `transparency` field modulates the alpha in the texture. Hence, a `transparency` of 0 will result in an alpha equal to that of the texture. A `transparency` of 1 will result in an alpha of 0 regardless of the value in the texture.



**Table 17.3 — Lit colour and alpha mapping**

Texture type	Colour per-vertex or per-face	Color node NULL
No texture	$O_{Drgb} = I_{Crgb}$ $A = 1 - T_M$	$O_{Drgb} = I_{Drgb}$ $A = 1 - T_M$
Intensity texture (one-component)	$O_{Drgb} = I_T \times I_{Crgb}$ $A = 1 - T_M$	$O_{Drgb} = I_T \times I_{Drgb}$ $A = 1 - T_M$
Intensity+Alpha texture (two-component)	$O_{Drgb} = I_T \times I_{Crgb}$ $A = A_T$	$O_{Drgb} = I_T \times I_{Drgb}$ $A = A_T$
RGB texture (three-component)	$O_{Drgb} = I_{Trgb}$ $A = 1 - T_M$	$O_{Drgb} = I_{Trgb}$ $A = 1 - T_M$
RGBA texture (four-component)	$O_{Drgb} = I_{Trgb}$ $A = A_T$	$O_{Drgb} = I_{Trgb}$ $A = A_T$

where:

$I_{Drgb}$  = material *diffuseColor*

$O_{Drgb}$  = diffuse factor, used in lighting equations below

$T_M$  = material *transparency*

All other terms are as defined in [17.2.2.2 Lighting off](#).

### 17.2.2.4 Lighting equations

An ideal X3D implementation will evaluate the following lighting equation at each point on a lit surface. RGB intensities at each point on a geometry ( $I_{rgb}$ ) are given by:

$$I_{rgb} = I_{Frgb} \times (1 - f_\theta) + f_\theta \times (O_{Ergb} + \text{SUM}(on_i \times \text{attenuation}_i \times \text{spot}_i \times I_{Lrgb} \times (\text{ambient}_i + \text{diffuse}_i + \text{specular}_i)))$$

where:

$$\text{attenuation}_i = 1 / \max(e_1 + e_2 \times d_L + e_3 \times d_L^2, 1)$$

$$\text{ambient}_i = I_{ia} \times O_{Drgb} \times O_a$$

$$\text{diffuse}_i = I_i \times O_{Drgb} \times (\mathbf{N} \cdot \mathbf{L})$$

$$\text{specular}_i = I_i \times O_{Srgb} \times (\mathbf{N} \cdot ((\mathbf{L} + \mathbf{v}) / |\mathbf{L} + \mathbf{v}|))^{shininess} \times 128$$

and:

$\cdot$  = modified vector dot product:

~~if dot product  $< 0$ , then 0.0, otherwise, dot product  
 $e_4, e_2, e_3$  = light  $i$  *attenuation*  
 $d_v$  = distance from point on geometry to viewer's position, in coordinate system of current fog node  
 $d_L$  = distance from light to point on geometry, in light's coordinate system  
 $f_\theta$  = fog interpolant, see [Table 17.5](#) for calculation  
 $t_{Frgb}$  = currently bound fog's *color*  
 $t_{Lrgb}$  = light  $i$  *color*  
 $t_i$  = light  $i$  *intensity*  
 $t_{ia}$  = light  $i$  *ambientIntensity*  
 $L$  = ([PointLight/SpotLight](#)) normalized vector from point on geometry to light source  $i$  position  
 $L$  = ([DirectionalLight](#)) direction of light source  $i$   
 $N$  = normalized normal vector at this point on geometry (interpolated from vertex normals specified in a node derived from [X3DNormalNode](#) or calculated by browser)  
 $\Theta_a$  = [X3DMaterialNode](#) *ambientIntensity*  
 $\Theta_{Drgb}$  = diffuse colour, from a node derived from [X3DMaterialNode](#), a node derived from [X3DColorNode](#), and/or a texture node  
 $\Theta_{Ergb}$  = [X3DMaterialNode](#) *emissiveColor*  
 $\Theta_{Srgb}$  = [X3DMaterialNode](#) *specularColor*  
 $on_i$  = 1, if light source  $i$  affects this point on the geometry;  
 0, if light source  $i$  does not affect this geometry. The following conditions indicate that light source  $i$  does not affect this geometry:  
 -  
 - a. if the geometry is farther away than *radius* for [PointLight](#) or [SpotLight](#);  
 - b. if the geometry is outside the enclosing [X3DGroupingNode](#); and/or  
 - c. if the *on* field is `FALSE`.  
*shininess* = [X3DMaterialNode](#) *shininess*  
 $spotAngle$  =  $\arccosine(-L \cdot spotDir_i)$   
 $spot_{BW}$  = [SpotLight](#)  $i$  *beamWidth*  
 $spot_{CO}$  = [SpotLight](#)  $i$  *cutOffAngle*  
 $spot_i$  = spotlight factor, see [Table 17.4](#) for calculation  
 $spotDir_i$  = normalized [SpotLight](#)  $i$  *direction*  
 SUM: sum over all light sources  $i$   
 $v$  = normalized vector from point on geometry to viewer's position~~

**Table 17.4 — Calculation of the spotlight factor**

Condition (in order)	spot <sub>i</sub> =
light <sub>i</sub> is PointLight or DirectionalLight	1
spotAngle ≥ spot <sub>CO</sub>	0
spotAngle ≤ spot <sub>BW</sub>	1
spot <sub>BW</sub> < spotAngle < spot <sub>CO</sub>	(spotAngle - spot <sub>CO</sub> ) / (spot <sub>BW</sub> - spot <sub>CO</sub> )

~~Table 17.5 — Calculation of the fog interpolant~~

Condition	$f_0 =$
no fog	1
fogType "LINEAR", $d_v < \text{fogVisibility}$	$(\text{fogVisibility} - d_v) / \text{fogVisibility}$
fogType "LINEAR", $d_v \geq \text{fogVisibility}$	0
fogType "EXPONENTIAL", $d_v < \text{fogVisibility}$	$\exp(-d_v / (\text{fogVisibility} - d_v))$
fogType "EXPONENTIAL", $d_v \geq \text{fogVisibility}$	0

### ~~17.2.2.5 References~~

The X3D lighting equations are based on the simple illumination equations given in [\[FOLEY\]](#) and [\[OPENGL\]](#).

### 17.2.2.1 Introduction

The X3D lighting model provides detailed equations that specify the colours to apply to each geometric object. For each object, the values of the material, color, and/or texture currently being applied to the object are combined with the lights illuminating the object and the currently bound *X3DFogObject* (if specified). These equations are designed to simulate the physical properties of light striking a surface.

If a programmable shader is defined for an [Appearance](#) node, the lighting model shall be disabled and replaced by the functionality implemented by the shader program. See [31 Programmable shaders component](#) for more information.

*Backward compatibility note:* The lighting equations in X3D 4.0 are backward compatible with X3D 3.3. If you take a combination of X3D 3.3 material and light nodes, and simply use them in X3D 4.0 (leaving the new X3D 4.0 fields at their default values) then the rendering result will be equivalent. The only exception to this statement is differentiating between grayscale and RGB textures, which is discussed in section below in [17.2.2.2 Texture sampling](#).

### 17.2.2.2 Texture sampling

When sampling *any* texture, the grayscale texture is exactly equivalent to using an RGB texture with all 3 components (red, green, blue) equal.

When sampling *any* texture, the texture without an alpha channel is exactly equivalent to using a texture with an alpha channel filled with 1 (indicating opaque).

These rules make treatment of the textures simple, and consistent with other 3D authoring software. They are also consistent with how the graphic APIs and GPU

shaders query the textures.

The browsers are encouraged to optimize loading of the textures, to not load all textures as 4 channels (RGBA) to the GPU. Intensity texture can be loaded as just 1 channel, intensity + alpha is only 2 channels, RGB texture without alpha is 3 channels. Optimizing this loading is useful to keep GPU memory usage low, and to keep texture loading time smaller. However, this is just an optimization. Performing it is optional, and the rendering result should be the same as if all textures were loaded as full RGBA textures.

*Backward compatibility note:* In X3D version 3, the treatment of grayscale and RGB textures was not consistent. In some cases (using *ImageTexture* node, but not inside *MultiTexture*) grayscale texture resulted in a different rendering result than the equivalent RGB texture (with all red, green, blue components equal). As this was inconsistent (within X3D, and with other software and model formats), uneasy to implement (browsers needed to investigate the image header), needlessly limiting to authors, and the implementation was inconsistent across the existing browsers — in X3D 4.0 it was streamlined.

### 17.2.2.3 Common definitions for all lighting models

The declarations and definitions below are presented using a pseudo-code similar to the usual shading language code.

- The function declarations look like this: *functionName*(*typeOfParameter1* *parameter1*, *typeOfParameter2* *parameter2*, ...). Type names are underlined.
- The types can be X3D nodes, scalars (*float*), vectors with 2, 3 or 4 components (*vector2*, *vector3*, *vector4*), universal type (*vectorAny* which can represent any vector or scalar).
- In function definitions, we often extract vector components with syntax like: *thisVector.rgb* (converts *vector4* to *vector3*, discarding alpha channel) or *thisVector.a* (converts *vector4* to *float*, extracting alpha channel value).
- The symbol  $\times$  performs a component-wise multiplication of vectors.
- The symbol  $\cdot$  is a modified vector dot product that always returns value  $\geq 0$ , defined like this:

$$x \cdot y = \max(0.0, \text{dotProduct}(x, y))$$

The *mixTexture*(*vector4* *color*, *X3DTextureNode* *texture*) function, used in the equations below, takes care of mixing an RGBA color with the RGBA value sampled from the texture at the given shape point.

- If the *texture* is `NULL`, then this function just returns unmodified *color*.
- Otherwise, if the *texture* is not a *MultiTexture*, then

$$\text{mixTexture}(\text{color}, \text{texture}) = \text{color} \times \text{textureSample}(\text{texture})$$

The *textureSample*(*texture*) is a function sampling the texture (recovering a single

color from an array of pixels), with the correct texture coordinates and transformation.

In effect the *color* modulates the color from the texture.

- *color.rgb = white = (1, 1, 1)* will result in the pure color of the texture,
- *color.rgb = black = (0, 0, 0)* will result in a black output, regardless of the texture.

The alpha (opacity) values are multiplied too, hence:

- *color.a* equal 1 (transparency equal 0) will result in an alpha equal to that of the texture,
  - *color.a* equal 0 (transparency equal 1) will result in an alpha of 0 regardless of the value in the texture.
- Otherwise, if the *texture* is a *MultiTexture*, then the *mixTexture* modifies this color following the [MultiTexture](#) mode specification. This is only possible when the *MultiTexture* is provided in the *Appearance.texture* field. See [12.2.5 Coexistence of textures specified in material nodes with the "Appearance.texture" field](#) for details when multi-texturing is used.

The *lerp(float factor, vectorAny x, vectorAny y)* function performs a standard linear interpolation, applicable to scalars or vectors of any dimension:

$$\text{lerp}(\text{factor}, x, y) = x * (1 - \text{factor}) + y * \text{factor} = x + (y - x) * \text{factor}$$

The *occlusion(vector4 color)* function, used in the equations below for *Phong* and *physical* lighting model, is used to apply the *occlusionTexture* effect that can be specified in these nodes.

- If the *occlusionTexture* was not provided (left `NULL`) then this function just returns unmodified *color*.
- If the *occlusionTexture* was provided, then

$$\text{occlusion}(\text{color}) = \text{lerp}(\text{occlusionStrength}, \text{color}, \text{color} * \text{textureSample}(\text{occlusionTexture}).r)$$

In effect, the *occlusionTexture* multiplies the input *color* when *occlusionStrength* is 1.0. the *occlusionTexture* has no effect when *occlusionStrength* is 0.0. Values in-between of *occlusionStrength* allow to smoothly interpolate between these two states.

The *applyColorPerVertex(vector4 color)* is used to change the *color* in case geometry uses *Color* or *ColorRGBA* nodes. All the lighting models use this function, although it affects a different parameter: *emissiveParameter* in case of *unlit* model, *diffuseParameter* in case of *Phong* model, and *baseParameter* in case of *physical* model. The function returns:

- The interpolated per-vertex colour, or per-face colour, from the *Color* node, if the *Color* node is provided in the geometry *color* field.

Resulting *rgb* is derived from the values in the *Color* node.

Resulting *a* (alpha component) is taken from input *color.a* in this case.

- Otherwise, the interpolated per-vertex colour, or per-face colour, from the *ColorRGBA* node, if the *ColorRGBA* node is provided in the geometry *color* field.

Resulting *rgba* vector is derived from the values in the *ColorRGBA* node.

- Otherwise (if the geometry *color* field is empty) then it returns unmodified *color*.

The future X3D versions may introduce an option for *Color* and *ColorRGBA* to multiply the input color, instead of replacing it.

Moreover we define the following vectors:

- $\mathbf{N}$  = normalized normal vector at this point on geometry. This vector is interpolated from vertex normals specified in a node derived from [X3DNormalNode](#) or calculated by the browser. It is modified by the *normalTexture* providing normals in the tangent space (see [X3DOneSidedMaterialNode](#) definition).
- $\mathbf{v}$  = normalized vector from point on geometry to viewer's position.

The following definitions are specific to a light source *i*, which is indicated by a subscript in this text:

- $\mathbf{L}_i$  is, conceptually, *direction to the light i*. It is precisely defined like this:
  - $\mathbf{L}_i = (\text{PointLight/SpotLight})$  normalized vector from point on geometry to light source *i* position
  - $\mathbf{L}_i = (\text{DirectionalLight})$  negated and normalized direction of light source *i*
- $attenuation_i = 1 / \max(c_1 + c_2 \times lightDistance + c_3 \times lightDistance^2, 1)$

where

$c_1, c_2, c_3$  are the values from light *i* *attenuation* field.

*lightDistance* is the distance from light to point on geometry, in light's coordinate system.

- $on_i = 1$ , if light source *i* affects this point on the geometry or 0 if it doesn't.

The following conditions indicate that light source *i* does not affect this geometry:

1. if the geometry is farther away than *radius* for PointLight or SpotLight
2. if the geometry is outside the enclosing [X3DGroupingNode](#) in case of lights with *global = FALSE*
3. if the *lightSource.on* field is `FALSE`.

- $spot_i$  is the spotlight factor. It calculates intensity within the [SpotLight](#) cone. [Table](#)

[17.4](#) specifies how it is calculated. It relies on the following terms:

- $spotDirection_i$  = normalized SpotLight  $i$  *direction* field.
- $spotAngle = \arccosine(-\mathbf{L} \cdot spotDirection_i)$

**Table 17.4 — Calculation of the spotlight factor**

summary=""

Condition (in order)	spot <sub>i</sub> =
light <sub>i</sub> is <i>PointLight</i> or <i>DirectionalLight</i>	1
spotAngle ≥ SpotLight.cutOffAngle	0
spotAngle ≤ SpotLight.beamWidth	1
SpotLight.beamWidth < spotAngle < lightSource.cutOffAngle	(spotAngle - SpotLight.cutOffAngle) / (SpotLight.beamWidth - SpotLight.cutOffAngle)

The *applyFog(vector4 color)* is used to change the *color* using the fog. Is it used by all lighting models, as the last operation performed on the color. The definition is:

$applyFog(color) = lerp(fogInterpolant(fogDistance), fogColor, color)$

where:

- *fogInterpolant* is the fog interpolant, see [Table 17.5](#) for calculation.
- *fogDistance* is the distance from point on geometry to viewer's position, in coordinate system of current fog node.
- *fogColor* is the currently bound fog's *color*.

**Table 17.5 — Calculation of the *fogInterpolant(FogDistance)* function:**

Condition	fogInterpolant(fogDistance) =
no fog	1
fogType "LINEAR", <i>fogDistance</i> < fogVisibility	(fogVisibility - <i>fogDistance</i> ) / fogVisibility
fogType "LINEAR", <i>fogDistance</i> ≥ fogVisibility	0
fogType "EXPONENTIAL", <i>fogDistance</i> < fogVisibility	$\exp(-FogDistance / (fogVisibility - fogDistance))$

fogType "EXPONENTIAL", <i>fogDistance</i> $\geq$ fogVisibility	0
---	---

### 17.2.2.4 Unlit lighting model

A [Shape](#) node is unlit if either of the following is true:

1. The shape's *appearance* field is `NULL` (default).
2. The *material* field in the Appearance node is `NULL` (default).
3. The *material* field in the Appearance node contains a node of type [UnlitMaterial](#).

In the first two cases above, the rendering is exactly equivalent as if the *Appearance* node was provided (not `NULL`), and the *material* field inside contained an *UnlitMaterial* with all the fields at their default. Effectively, it means a *white untextured unlit* material is the default.

NOTE Geometry nodes that represent lines or points do not support lighting if the normal vectors for them are not provided.

If the shape is unlit, the RGBA color of the shape at each point on the shape's geometry is calculated using this equation:

$$\text{fragmentColor} = \text{applyFog}(\text{mixTexture}(\text{applyColorPerVertex}(\text{emissiveParameter}), \text{emissiveTextureParameter}))$$

where:

- *emissiveParameter.rgb* (RGB channels) are taken from *UnlitMaterial.emissiveColor*.
- *emissiveParameter.a* (alpha channel) is taken from  $1 - \text{UnlitMaterial.transparency}$ .
- *emissiveTextureParameter* is equal to:
  - *emissiveTexture* of the *UnlitMaterial* node, if it is not `NULL`.
  - Otherwise, *Appearance.texture*, if the *UnlitMaterial* has *emissiveTexture* equal `NULL`, but *Appearance.texture* is not `NULL`. See [12.2.5 Coexistence of textures specified in material nodes with the "Appearance.texture" field](#).
  - Otherwise (if both the *UnlitMaterial.emissiveTexture* and *Appearance.texture* are `NULL`) then *emissiveTextureParameter* is `NULL`. In other words, the *mixTexture(...)* function used above simply returns the unmodified *applyColorPerVertex(emissiveParameter)*.

### 17.2.2.5 Phong lighting model

The [Shape](#) node is lit with a *Phong lighting model* if the [Appearance](#) node is specified for the Shape, and the *material* field contains a [Material](#) node.

Note: This node is simply called *Material* for historical reasons. Conceptually, you should



think about it now as a *PhongMaterial*.

The rendered fragment (pixel) color is determined by these equations:

$$\text{fragmentColor} = \text{applyFog}(\text{emissiveParameter} + \text{occlusion}(\text{sumOverAllLights}(\text{lightContribution}_i)))$$

$$\text{lightContribution}_i = \text{on}_i \times \text{attenuation}_i \times \text{spot}_i \times (\text{ambient}_i + \text{diffuse}_i + \text{specular}_i)$$

An ideal X3D implementation will evaluate the following lighting equation at each point on a lit surface. This means that we advise using *Phong shading* and assume it when writing equations below. For implementations that perform *Gouraud shading* see [17.2.2.8 Gouraud shading](#) section.

The meaning of all the terms is explained below.

## Material parameters

First, the parameters whose value doesn't depend on the light source:

The material *diffuseParameter* is calculated as follows:

$$\text{diffuseParameter} = \text{mixTexture}(\text{applyColorPerVertex}(\text{diffuseParameter}), \text{diffuseTextureParameter})$$

where:

- *diffuseParameter.a* (RGB channels) is taken from *Material.diffuseColor*.
- *diffuseParameter.a* (alpha channel) is taken from  $1 - \text{Material.transparency}$ .
- *diffuseTextureParameter* is equal to:
  - *diffuseTexture* of the *Material* node, if it is not `NULL`.
  - Otherwise, *Appearance.texture*, if the *Material* has *diffuseTexture* equal `NULL`, but *Appearance.texture* is not `NULL`. See [12.2.5 Coexistence of textures specified in material nodes with the "Appearance.texture" field](#).
  - Otherwise (if both the *Material.diffuseTexture* and *Appearance.texture* are `NULL`) then *diffuseTextureParameter* is `NULL`. In other words, the *mixTexture(...)* function used above simply returns the unmodified *applyColorPerVertex(diffuseParameter)*.

The remaining parameters are defined below:

- $\text{ambientParameter} = \text{ambientIntensity} \times \text{diffuseColor} \times \text{textureSample}(\text{ambientTexture}).\text{rgb}$
- $\text{emissiveParameter} = \text{emissiveColor} \times \text{textureSample}(\text{emissiveTexture}).\text{rgb}$
- $\text{specularParameter} = \text{specularColor} \times \text{textureSample}(\text{specularTexture}).\text{rgb}$
- $\text{shininessParameter} = \text{shininess} \times \text{textureSample}(\text{shininessTexture}).\text{a} \times 128$

In the above equations, if the given texture is `NULL` then behave as if the `textureSample(texture)` returned a white opaque value (1, 1, 1, 1).

## Light parameters

Now we can define terms that depend on the light source:

- $ambient_i = lightSource.ambientIntensity \times ambientParameter$
- $diffuse_i = lightSource.intensity \times diffuseParameter \times (\mathbf{N} \cdot \mathbf{L})$
- $specular_i = lightSource.intensity \times specularParameter \times (\mathbf{N} \cdot ((\mathbf{L} + \mathbf{v}) / |\mathbf{L} + \mathbf{v}|))^{shininessParameter}$

### 17.2.2.6 Physical lighting model

The [Shape](#) node is lit with a *Physical lighting model* if the [Appearance](#) node are specified for the Shape, and the *material* field contains a [PhysicalMaterial](#) node. We perform in this case a *physically-based rendering*.

The rendered fragment (pixel) color is determined by these equations:

$$fragmentColor = applyFog(emissiveParameter + occlusion(sumOverAllLights(lightContribution_i)))$$

$$lightContribution_i = on_i \times attenuation_i \times spot_i \times physicalLightContribution_i$$

The input values used by the physical lighting equation are as follows:

$$baseParameter = mixTexture(applyColorPerVertex(baseParameter), baseTextureParameter)$$

where:

- *baseParameter.a* (RGB channels) is taken from *PhysicalMaterial.baseColor*.
- *baseParameter.a* (alpha channel) is taken from  $1 - PhysicalMaterial.transparency$ .
- *baseTextureParameter* is equal to:
  - *baseTexture* of the *PhysicalMaterial* node, if it is not `NULL`.
  - Otherwise, *Appearance.texture*, if the *PhysicalMaterial* has *baseTexture* equal `NULL`, but *Appearance.texture* is not `NULL`. See [12.2.5 Coexistence of textures specified in material nodes with the "Appearance.texture" field](#).
  - Otherwise (if both the *PhysicalMaterial.baseTexture* and *Appearance.texture* are `NULL`) then *baseTextureParameter* is `NULL`. In other words, the *mixTexture(...)* function used above simply returns the unmodified *applyColorPerVertex(baseParameter)*.

$$metallicParameter = PhysicalMaterial.metallic \times$$

$textureSample(PhysicalMaterial.metallicRoughnessTexture).b$

$roughnessParameter = PhysicalMaterial.roughness \times textureSample(PhysicalMaterial.metallicRoughnessTexture).g$

If the *metallicRoughnessTexture* is `NULL`, then *metallicParameter* and *roughnessParameter* are just equal to (respectively) *metallic* and *roughness* values given by the *PhysicalMaterial* node.

Using the *baseParameter*, *metallicParameter* and *roughnessParameter*, the physical lighting model performs the exact same computations as the recommended glTF lighting model, to calculate *physicalLightContribution<sub>i</sub>* value. See [glTF 2.0 specification section about lighting equations](#) and [glTF-Sample-Viewer implementation of it](#). Future revisions of this draft will contain the final recommended equations.

### 17.2.2.7 References

The *Phong* lighting equations are based on the simple illumination equations given in [\[FOLEY\]](#) and [\[OPENGL\]](#).

The *physical* lighting equations are based on the [glTF 2.0 specification](#).

### 17.2.2.8 Gouraud shading

An ideal X3D implementation will evaluate the lighting equation at each point on a lit surface. The lighting equations in previous sections assume that you use *Phong shading* (not to be confused with *Phong lighting model*). In case of *Phong shading*, lighting is calculated at each fragment (pixel of the screen).

However, some implementations perform *Gouraud shading*, either by default, or as an option for the user (for efficiency), or as a fallback for an older GPUs. In such case, the lighting equations given above cannot be reproduced precisely. In Gouraud shading, you calculate lighting per-vertex, which means that it doesn't make sense to use texture information to modify material parameters.

In case when *Gouraud shading* is used we recommend this algorithm:

- The vertex shader calculates lighting equations, ignoring any texture information (as if all the textures were `NULL`).
- The fragment shader multiplies the resulting interpolated color by the *main texture*. The *main texture* is:
  - *Material.diffuseTexture*, if *Phong Material* node is used.
  - *UnlitMaterial.emissiveTexture*, if *UnlitMaterial* node is used.
  - *PhysicalMaterial.baseTexture*, if *PhysicalMaterial* node is used.

If the given texture is `NULL`, then use the *Appearance.texture*. If the *Appearance.texture* is also `NULL`, no texture is used.

All channels (RGB and alpha) are affected by the *main texture*.

This method of determining the *main texture* is deliberately 100% consistent with:

1. The way how the *Appearance.texture* cooperates with textures inside the materials, following section [12.2.5 Coexistence of textures specified in material nodes with the "Appearance.texture" field](#). *Appearance.texture* may be used in place of *Material.diffuseTexture*, or *UnlitMaterial.emissiveTexture*, or *PhysicalMaterial.baseTexture*.
2. The determination which texture contains the alpha channel that is multiplied by material *transparency*. We take it from alpha channel of *Material.diffuseTexture*, or *UnlitMaterial.emissiveTexture*, or *PhysicalMaterial.baseTexture*.

This recommendation tries to preserve the intended look as much as possible in *Gouraud shading*.

Note: This recommendation actually doesn't change anything in case of *UnlitMaterial*. And this is correct, *Gouraud shading* actually doesn't change how the *UnlitMaterial* can be implemented.

Note: When using *PhysicalMaterial* on older hardware, some implementations may fall back to the *Phong* lighting model. If this is necessary, we recommend using *PhysicalMaterial.baseColor* as the *Phong* diffuse factor, and *PhysicalMaterial.baseTexture* as the texture to multiply the resulting color.

## 17.3 Abstract types

### 17.3.1 X3DLightNode

```
X3DLightNode : X3DChildNode {
  SFFloat [in,out] ambientIntensity 0 [0,1]
  SFColor [in,out] color 1 1 1 [0,1]
  SFBool [in,out] global FALSE
  SFFloat [in,out] intensity 1 [0,1]
  SFFloat [in,out] intensity 1 [0,∞]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFBool [in,out] on TRUE
}
```

The *X3DLightNode* abstract node type is the base type from which all node types that serve as light sources are derived. A description of the *ambientIntensity*, *color*, *intensity*, and *on* fields is in [17.2.1 Light source semantics](#). A description of the *global* field is in [17.2.1.2 Scoping of lights](#).

## 17.4 Node reference

### 17.4.1 DirectionalLight

```
DirectionalLight : X3DLightNode {
  SFFloat [in,out] ambientIntensity 0 [0,1]
  SFColor [in,out] color 1 1 1 [0,1]
  SFVec3f [in,out] direction 0 0 -1 (-∞,∞)
  SFBool [in,out] global FALSE
  SFFloat [in,out] intensity 1 [0,1]
  SFFloat [in,out] intensity 1 [0,∞]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFBool [in,out] on TRUE
}
```

The `DirectionalLight` node defines a directional light source that illuminates along rays parallel to a given 3-dimensional vector. A description of the *ambientIntensity*, *color*, *intensity*, and *on* fields is in [17.2.1 Light source semantics](#). A description of the *global* field is in [17.2.1.2 Scoping of lights](#).

The *direction* field specifies the direction vector of the illumination emanating from the light source in the local coordinate system. Light is emitted along parallel rays from an infinite distance away. A directional light source illuminates only the objects in its enclosing parent group. The light may illuminate everything within this coordinate system, including all children and descendants of its parent group. The accumulated transformations of the parent nodes affect the light.

`DirectionalLight` nodes do not attenuate with distance. A precise description of X3D's lighting equations is contained in [17.2.2 Lighting model](#).

## 17.4.2 PointLight

```
PointLight : X3DLightNode {
  SFFloat [in,out] ambientIntensity 0 [0,1]
  SFVec3f [in,out] attenuation 1 0 0 [0,∞)
  SFColor [in,out] color 1 1 1 [0,1]
  SFBool [in,out] global TRUE
  SFFloat [in,out] intensity 1 [0,1]
  SFFloat [in,out] intensity 1 [0,∞)
  SFVec3f [in,out] location 0 0 0 (-∞,∞)
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFBool [in,out] on TRUE
  SFFloat [in,out] radius 100 [0,∞)
}
```

The `PointLight` node specifies a point light source at a 3D location in the local coordinate system. A point light source emits light equally in all directions; that is, it is omnidirectional. `PointLight` nodes are specified in the local coordinate system and are affected by ancestor transformations. A description of the *global* field is in [17.2.1.2 Scoping of lights](#).

Subclause [17.2.1 Light source semantics](#), contains a detailed description of the *ambientIntensity*, *color*, and *intensity* fields.

A `PointLight` node illuminates geometry within *radius* length base units of its *location*. Both *radius* and *location* are affected by ancestors' transformations (scales affect *radius* and transformations affect *location*). The *radius* field shall be greater than or equal to zero.

`PointLight` node's illumination falls off with distance as specified by three *attenuation* coefficients. The attenuation factor is:

$$1/\max(\text{attenuation}[0] + \text{attenuation}[1] \times r + \text{attenuation}[2] \times r^2, 1)$$

where *r* is the distance from the light to the surface being illuminated. The default is no attenuation. An *attenuation* value of (0, 0, 0) is identical to (1, 0, 0). Attenuation values shall be greater than or equal to zero. A detailed description of X3D's lighting equations is contained in [17.2.2 Lighting model](#).

## 17.4.3 SpotLight

```
SpotLight : X3DLightNode {
  SFFloat [in,out] ambientIntensity 0 [0,1]
  SFVec3f [in,out] attenuation 1 0 0 [0,∞)
```

```

SFFloat [in,out] beamWidth      π/4  (0,π/2]
SFColor [in,out] color          1 1 1  [0,1]
SFFloat [in,out] cutOffAngle    π/2  (0,π/2]
SFVec3f [in,out] direction      0 0 -1  (-∞,∞)
SFBool [in,out] global          TRUE
SFFloat [in,out] intensity      1      [0,1]
SFFloat [in,out] intensity      1      [0,∞)
SFVec3f [in,out] location        0 0 0  (-∞,∞)
SFNode [in,out] metadata        NULL  [X3DMetadataObject]
SFBool [in,out] on              TRUE
SFFloat [in,out] radius          100  [0,∞)
}

```

The `SpotLight` node defines a light source that emits light from a specific point along a specific direction vector and constrained within a solid angle. Spotlights may illuminate geometry nodes that respond to light sources and intersect the solid angle defined by the `SpotLight`. Spotlight nodes are specified in the local coordinate system and are affected by ancestors' transformations. A description of the *global* field is in [17.2.1.2 Scoping of lights](#).

A detailed description of *ambientIntensity*, *color*, *intensity*, and the lighting equations of X3D is provided in [17.2.1 Light source semantics](#). More information on lighting concepts can be found in [17.2.2 Lighting model](#), including a detailed description of the X3D lighting equations.

The *location* field specifies a translation offset of the centre point of the light source from the light's local coordinate system origin. This point is the apex of the solid angle which bounds light emission from the given light source. The *direction* field specifies the direction vector of the light's central axis defined in the local coordinate system.

The *on* field specifies whether the light source emits light. If *on* is `TRUE`, the light source is emitting light and may illuminate geometry in the scene. If *on* is `FALSE`, the light source does not emit light and does not illuminate any geometry.

The *radius* field specifies the radial extent of the solid angle and the maximum distance from *location* that may be illuminated by the light source. The light source does not emit light outside this radius. The *radius* shall be greater than or equal to zero.

Both radius and location are affected by ancestors' transformations (scales affect *radius* and transformations affect *location*).

The *cutOffAngle* field specifies the outer bound of the solid angle. The light source does not emit light outside of this solid angle. The *beamWidth* field specifies an inner solid angle in which the light source emits light at uniform full intensity. The light source's emission intensity drops off from the inner solid angle (*beamWidth*) to the outer solid angle (*cutOffAngle*) as described in the following equations:

```

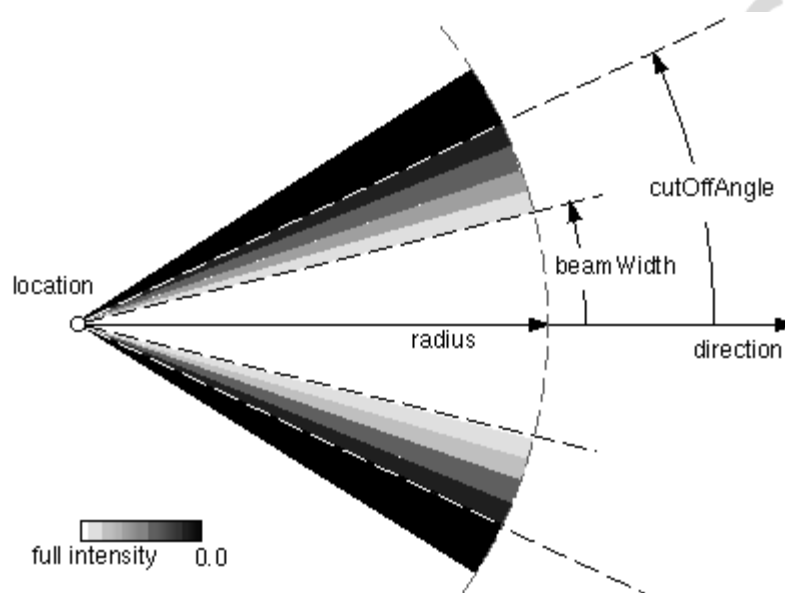
angle = the angle between the Spotlight's direction vector
         and the vector from the Spotlight location to the point
         to be illuminated

if (angle ≥ cutOffAngle):
  multiplier = 0
else if (angle ≤ beamWidth):
  multiplier = 1
else:
  multiplier = (angle - cutOffAngle) / (beamWidth - cutOffAngle)
intensity(angle) = Spotlight.intensity × multiplier

```

If the *beamWidth* is greater than the *cutOffAngle*, *beamWidth* is defined to be equal to the *cutOffAngle* and the light source emits full intensity within the entire solid angle defined by *cutOffAngle*. Both *beamWidth* and *cutOffAngle* shall be greater than 0.0 and less than or equal to  $\pi/2$ . [Figure 17.1](#) depicts the *beamWidth*, *cutOffAngle*, *direction*,

*location*, and *radius* fields of the SpotLight node.



**Figure 17.1 — SpotLight node**

SpotLight illumination falls off with distance as specified by three *attenuation* coefficients. The attenuation factor is:

$$1/\max(\text{attenuation}[0] + \text{attenuation}[1] \times r + \text{attenuation}[2] \times r^2, 1)$$

where *r* is the distance from the light to the surface being illuminated. The default is no attenuation. An *attenuation* value of (0, 0, 0) is identical to (1, 0, 0). Attenuation values shall be greater than or equal to zero. A detailed description of X3D's lighting equations is contained in [17.2.2 Lighting model](#).

## 17.5 Support levels

The Lighting component provides three levels of support as specified in [Table 17.6](#).

**Table 17.6 — Lighting component support levels**

Level	Prerequisites	Nodes/Features	Support
1	Core 1 Shape 1		
		<i>X3DLightNode</i> (abstract)	n/a
		DirectionalLight	Not scoped by parent Group or Transform.
2	Core 1 Shape 1		
		All Level 1 Lighting	All fields as supported in



		nodes	Level 1.
		PointLight	<i>radius</i> optionally supported. Linear attenuation.
		SpotLight	<i>beamWidth</i> optionally supported. <i>radius</i> optionally supported. Linear attenuation.
<b>3</b>	Core 1 Shape 1		
		All Level 2 Lighting nodes	All fields fully supported.







## Extensible 3D (X3D) Part 1: Architecture and base components

### 38 Picking component

---



#### 38.1 Introduction

##### 38.1.1 Name

The name of this component is "Picking". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 38.1.2 Overview

This component provides the ability to test for arbitrary object collision in a somewhat limited form. In traditional 3D graphics terminology, this is termed picking. The intention is not to support full n-body object collision, but to provide an extended set of basic capabilities to provide some limited custom interactions, such as terrain following. [Table 38.1](#) provides links to the major topics in this clause.

**Table 38.1 — Topics**

- [38.1 Introduction](#)
  - [38.1.1 Name](#)
  - [38.1.2 Overview](#)
- [38.2 Concepts](#)
  - [38.2.1 Overview](#)
  - [38.2.2 Event model interaction](#)
  - [38.2.3 Transformation Hierarchy](#)
- [38.3 Abstract types](#)
  - [38.3.1 X3DPickableObject](#)
  - [38.3.2 X3DPickSensorNode](#)
- [38.4 Node reference](#)
  - [38.4.1 LinePickSensor](#)
  - [38.4.2 PickableGroup](#)
  - [38.4.3 PointPickSensor](#)
  - [38.4.4 PrimitivePickSensor](#)
  - [38.4.5 VolumePickSensor](#)

- [38.5 Support levels](#)
- [Figure 38.1 — Illustration of the different conditions of intersections of lines and coplanar polygons](#)
- [Table 38.1 — Topics](#)
- [Table 38.2 — Picking component support levels](#)

## 38.2 Concepts

### 38.2.1 Overview

This component provides a means of testing for object intersection that permits a greater degree of programmable interaction of content. Various types of geometrical elements may be used to test for intersection between the renderable scene graph and the nodes provided by this component. When one or more intersections are found, the results are reported using the sensor model of this International Standard and are then available for further processing by the event model.

Intersection testing consists of two parts: an object representing the type of intersection to be created and a scene graph tree to be tested. The intersecting object is represented by nodes that extend the *X3DPickSensorNode* abstract type. Instances of *X3DPickableObject* mark a scene graph subtree as a target for testing.

### 38.2.2 Event model interaction

Picking is performed between rendered frames of the event model. A user sets up the picking request in one frame by placing, in the desired location, a node derived from *X3DPickSensorNode*. Such a node is termed a *pick sensor*. At the start of the next frame any intersections are reported from the pick sensor.

Picking notification is performed at the start of the frame for all enabled pick sensors when all other sensors are processed (see [4.4.8.3 Execution model](#) step b). Disabled pick sensors do not need to be evaluated. This allows the user to manipulate geometry and have the pick results returned at the start of the frame, thus ensuring a fixed, known state at all times.

### 38.2.3 Transformation Hierarchy

Testing for intersection tests is a global action within each execution context. pick sensors may report intersections with contained contexts, but only to the wrapper and not the contents of that context.

**EXAMPLE** Picking against a scene that contains an Inline node will return the Inline node as the picked geometry rather than a node from the contents of the geometry.

A pick sensor is located at the desired position and orientation in the scene graph using the transformation hierarchy. The pick sensor is effected by translation, orientation and scale operations. If a non-uniform scale is applied to the pick sensor, the results are dependent on the selected component level.

The picked objects are those that have been given to that specific pick sensor instance in its *pickTarget* field. All transformations above those picked objects are applied to the picking process. Picking is performed in world coordinate space after transformations have been applied to both the pick sensor and the target nodes.

Sections of the scene graph contained by a *X3DPickableObject* are used for additional filtering of the picking operations. The pickable object has a set of flags defined in the *objectType* field that can be used to classify sections of the scene graph so that picking will only report intersections in those classifications.

EXAMPLE A pickable object classifies itself as a "WATER" object and the pick sensor declares that it is picking for "GROUND" objects. Even though the pick sensor intersects with the picking object, no result is returned because the pickable object and the pick sensor do not have the same object type category.

When reporting results requires specific geometry intersection points, the results are reported in the local coordinate space of the pick sensor.

## 38.3 Abstract types

### 38.3.1 *X3DPickableObject*

```
X3DPickableObject {
  MFString [in,out] objectType "ALL" ["ALL","NONE","TERRAIN",...]
  SFBool [in,out] pickable TRUE
}
```

The *X3DPickableObject* abstract **object type interface** marks a node as being capable of having customized picking performed on its contents or children.

The *pickable* field is used to independently control whether picking may be performed on this node or its children. Setting the value to `FALSE` will remove the children from the list of potential matches for picking. This only affects children that are accessed through the transformation hierarchy of the parent. If one or more of the children of this instance is accessible through another transformation hierarchy through DEF/USE that still has picking enabled, they shall still be pickable through that path only. **Object picking according to the *pickable* field occurs even if the object is not rendered visibly.**

The *objectType* field specifies a label that is used in the picking process. Each string specified is treated as an independent label that needs to be matched against the same type in one of the pick sensor instances.

EXAMPLE Labeling a group with the value "WATER" and then attempting to intersect a pick sensor with *objectType* "GROUND" would fail as the types are not matching.

The special object type "ALL" means that it is available for picking regardless of the type specified by the pick sensor. The special value "NONE" overrides the presence of any other string values in this *objectType* field, thereby disabling picking for this node. The presence of the "ALL" special value indicates that all *objectTypes* defined within the scope defined by the construct derived from *X3DPickableObject* are eligible. If both "NONE" and "ALL" are specified, the special value "NONE" applies. The user may define any value for *objectType*.

## 38.3.2 X3DPickSensorNode

```

X3DPickSensorNode : X3DSensorNode {
  SFBool [in,out] enabled      TRUE
  SFNode  [in,out] metadata    NULL      [X3DMetadataObject]
  SFString [in,out] matchCriterion "MATCH_ANY" ["MATCH_ANY"|"MATCH EVERY"|"MATCH_ONLY_ONE"]
  MFString [in,out] objectType  "ALL"     ["ALL","NONE","TERRAIN",...]
  SFNode  [in,out] pickingGeometry NULL    [X3DGeometryNode]
  MFNode  [in,out] pickTarget   []        [X3DGroupingNode|X3DShapeNode|Inline]
  MFNode  [out]   pickedGeometry
  SFBool  [out]   isActive
  SFString []     intersectionType "BOUNDS" ["GEOMETRY"|"BOUNDS"|...]
  SFString []     sortOrder       "CLOSEST" ["ANY"|"CLOSEST"|"ALL"|"ALL_SORTED"]
}

```

The *X3DPickSensorNode* abstract node type is the base **node** type that represents the lowest common denominator of picking capabilities. An *X3DPickSensorNode* is a type of *X3DSensorNode*. The field *isActive* is `TRUE` whenever there is a picked item available. If the intersecting object is not picked by the picking geometry, the pick sensor is not active.

The *intersectionType* field specifies the precision of the collision computation. **When testing intersections, "BOUNDS" indicates that the *pickingGeometry* is intersected with the bounding box of the pickable object, whereas "GEOMETRY" indicates that the *pickingGeometry* is intersected with the geometry of the pickable object.** The *intersectionType* constants may be extended by the individual concrete node to provide additional options.

EXAMPLE 1 An *intersectionType* may be used to specify the specific algorithm used for the detection.

The *objectType* field lists the types of object that are to be tested for intersections. The special value "NONE" overrides the presence of any other string values in this *objectType* field, thereby disabling picking for this node. The presence of the "ALL" special value indicates that all *objectTypes* are potential pick targets. If both "NONE" and "ALL" are specified, the value "NONE" applies. An arbitrary label **(such as "TERRAIN")** may be specified here as well as the predefined types. **Such a label indicates that only pickable objects with an identical label may be picked.**

The *matchCriterion* field defines whether the *X3DPickSensorNode* pick matches one or more *objectType* value(s), as follows:

- "MATCH\_ANY" means that any match of *objectType* values is acceptable.
- "MATCH EVERY" means that every *objectType* value in the *X3DPickSensorNode* shall match an *objectType* value in the *X3DPickableObject*.
- "MATCH\_ONLY\_ONE" means that one and only one *objectType* value can match.

The *pickingGeometry* field specifies the exact coordinates of the geometry that will be performing the intersection testing. The acceptable range of node types and how they are to be interpreted shall be defined by the individual concrete nodes.

The *pickTarget* field specifies the list of nodes against which the picking operation should be performed. All nodes declared in this field and their descendents shall be evaluated for intersections based on the specific sensor definition. If a descendent of the nodes declared in this field includes another *X3DPickSensorNode* instance, the children of the descendent *X3DPickSensorNode*'s *pickTarget* field are not considered for picking.

The *pickedGeometry* field communicates the node or nodes that have been found to intersect with the picking geometry from the last time this node performed a picking operation. The values provided shall be dependent on the setting of the *sortOrder* field.

The values of the *sortOrder* has four predefined values.

- a. "ANY" Any single object that satisfies the picking conditions for this pick sensor. Consistency of results is not guaranteed.
- b. "ALL" Every object that satisfies the picking conditions for this pick sensor shall be returned.
- c. "ALL\_SORTED" Every object that satisfies the picking conditions for this pick sensor shall be returned with the order of the output fields provided in a distance-sorted order from closest to farthest away. The exact algorithm for sorting is defined by the individual node definitions.
- d. "CLOSEST" The closest object by distance that satisfies the conditions of this pick sensor. The exact algorithm for distance determination shall be defined by the individual node definitions.

Browser implementations may define additional values and algorithms beyond these four required values.

## 38.4 Node reference

### 38.4.1 LinePickSensor

```

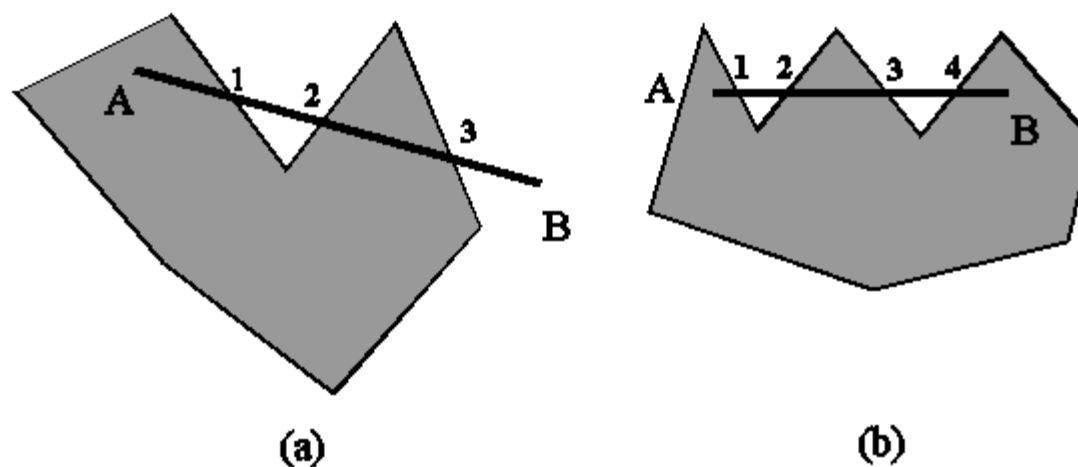
LinePickSensor : X3DPickSensorNode {
  SFBool [in,out] enabled          TRUE
  SFNode  [in,out] metadata        NULL [X3DMetadataObject]
  MFString [in,out] objectType     "ALL" ["ALL","NONE","TERRAIN",...]
  SFNode  [in,out] pickingGeometry NULL [IndexedLineSet|LineSet]
  MFNode  [in,out] pickTarget      [] [X3DGroupingNode|X3DShapeNode|Inline]
  SFBool  [out]  isActive
  MFNode  [out]  pickedGeometry
  MFVec3f [out]  pickedNormal
  MFVec3f [out]  pickedPoint
  MFVec3f [out]  pickedTextureCoordinate
  SFString []   intersectionType   "BOUNDS" ["GEOMETRY"|"BOUNDS"...]
  SFString []   sortOrder          "CLOSEST" ["ANY"|"CLOSEST"|"ALL"|"ALL_SORTED"]
}

```

The LinePickSensor node picks one or more line segments as the test object with which to pick. As a line intersect generates a known point in space, normal, geometry and texCoord information can be returned that is useful.

Line picking, for sort order determination is based on the pair of coordinates that defines the line segment. The first declared vertex of the segment is defined to be the start of the line to which the intersection points are closest.

When the picking line segment intersects a coplanar polygon and one vertex lies outside the polygon, the intersection point(s) will be those on the edge(s) of the polygon (see Figure 38.1 (a)). If the entire segment lies entirely within the polygon then the intersection point shall be defined to be the start point of the segment. For concave polygons where both ends of the segment lie in the polygon but the line exits the polygon for some portion (see [Figure 38.1](#) (b)), the intersection points are the intersecting edges of the polygon, where sort order is defined as in the previous paragraph.



**Figure 38.1 — Illustration of the different conditions of intersections of lines and coplanar polygons. (a) One end point contained in the polygon and one external. (b) Both end points internal to the polygon. Point A is the start point of the line and the numbers indicate the sort order that shall be returned.**

Picked texture coordinates are in three dimensions. If the target object has multiple textures defined, only the texture coordinates for the first texture are returned. All other textures are ignored. If the target texture coordinate has two dimensions, the third coordinate (z component of an SFVec3f) shall be zero.

### 38.4.2 PickableGroup

```
PickableGroup : X3DGroupingNode, X3DPickableObject {
  MFNode [in]  addChildren
  MFNode [in]  removeChildren
  MFNode [in,out] children [] [X3DChildNode]
  SFBool [in out] bboxDisplay FALSE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFString [in,out] objectType "ALL" ["ALL","NONE","TERRAIN",...]
  SFBool [in,out] pickable TRUE
  SFBool [in out] visible TRUE
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}
```

A PickableGroup node is an X3DGroupingNode that contains *children* that are marked as being of a given classification of picking types, as well as the ability to enable or disable picking of the *children*.

For field definitions, see [38.3.1 X3DPickableObject](#) and [10.3.2 X3DGroupingNode](#).

### 38.4.3 PointPickSensor

```
PointPickSensor : X3DPickSensorNode {
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFString [in,out] objectType "ALL" ["ALL","NONE","TERRAIN",...]
  SFNode [in,out] pickingGeometry NULL [PointSet]
  MFNode [in,out] pickTarget [] [X3DGroupingNode|X3DShapeNode|Inline]
  SFBool [out] isActive
  MFNode [out] pickedGeometry
  MFVec3f [out] pickedPoint
  SFString [] intersectionType "BOUNDS" ["GEOMETRY","BOUNDS",...]
  SFString [] sortOrder "CLOSEST" ["CLOSEST","ALL","ALL_SORTED"]
}
```



The PointPickSensor node tests one or more points in space as lying inside the provided target geometry. For each of the picked points intersecting the geometry, the point coordinate is returned as an element in the pickedPoint field, and the corresponding geometry node (inside which each intersection point lies) is returned as an element of the pickedGeometry field. For each point that lies inside the geometry, the point coordinate is returned in the *pickedGeometry* field with the corresponding geometry inside which the point lies.

Because points represent an infinitely small location in space, the "CLOSEST" and "ALL\_SORTED" sort orders are defined to mean "ANY" and "ALL" respectively.

### 38.4.4 PrimitivePickSensor

```
PrimitivePickSensor : X3DPickSensorNode {
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFString [in,out] objectType "ALL" ["ALL","NONE","TERRAIN",...]
  SFNode [in,out] pickingGeometry NULL [Cone|Cylinder|Sphere|Box]
  MFNode [in,out] pickTarget [] [X3DGroupingNode|X3DShapeNode|Inline]
  SFBool [out] isActive
  MFNode [out] pickedGeometry
  SFString [] intersectionType "BOUNDS" ["GEOMETRY"|"BOUNDS"...]
  SFString [] sortOrder "CLOSEST" ["ANY"|"CLOSEST"|"ALL"|"ALL_SORTED"]
}
```

The PrimitivePickSensor node picks against the target geometry using one of the basic primitive object types specified in the *pickingGeometry* field.

Boolean fields used to control visibility of subsections of a primitive are ignored when evaluating the picking routines.

EXAMPLE A cylinder missing the end caps is still treated as an enclosed cylinder.

Sorting is defined based on the primitive type as follows:

- For Cone, the closest picked primitive is defined to be that closest to the vertex point.
- For Cylinder, Box, and Sphere, the closest picked primitive is defined to be that closest to the centre.

### 38.4.5 VolumePickSensor

```
VolumePickSensor : X3DPickSensorNode {
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFString [in,out] objectType "ALL" ["ALL","NONE","TERRAIN",...]
  SFNode [in,out] pickingGeometry NULL [X3DGeometryNode]
  MFNode [in,out] pickTarget [] [X3DGroupingNode|X3DShapeNode|Inline]
  SFBool [out] isActive
  MFNode [out] pickedGeometry
  SFString [] intersectionType "BOUNDS" ["GEOMETRY"|"BOUNDS"...]
  SFString [] sortOrder "CLOSEST" ["ANY"|"CLOSEST"|"ALL"|"ALL_SORTED"]
}
```

The VolumePickSensor picks against an arbitrary volume defined by the geometry. The volume is defined by the convex hull of the enclosing planes of the provided geometry. If the provided volume is not manifold, the pick results are undefined.

A pick is successful if any vertex of the pickTarget geometry intersects the volume defined by the pickingGeometry. The sort order is based on the distance between the centers of the bounds of the picking geometry and the picked geometry.

## 38.5 Support levels

The Picking component provides three levels of support as specified in [Table 38.2](#).

**Table 38.2 — Picking component support levels**

Level	Prerequisites	Nodes/Features	Support
<b>1</b>	Core 1 Grouping 1 Shape 1 Rendering 1		
		<i>X3DPickSensorNode</i>	n/a
		<i>X3DPickableObject</i>	n/a
		LinePickSensor	All fields fully supported.
		PickableGroup	All fields fully supported.
		PointPickSensor	All fields fully supported.
<b>2</b>	Core 1 Grouping 1 Shape 1 Rendering 1		
		All Level 1 nodes	All fields fully supported.
		PrimitivePickSensor	All fields fully supported. Non uniform scale not supported.
<b>3</b>	Core 1 Grouping 1 Shape 1 Rendering 1		
		All Level 2 nodes	All fields fully supported.
		PrimitivePickSensor	All fields fully supported. Non-uniform scale supported.
		VolumePickSensor	All fields fully supported.







## Extensible 3D (X3D) Part 1: Architecture and bases

### Node index *Node, abstract node type, and abstract interface index*



#### General

This index lists the nodes in alphabetical order. Abstract *nodes and objects* node types and abstract interfaces are italicized.

<b>Node, abstract node type, and abstract interface</b>	<b>Specification Subclause</b>
<a href="#"><i>AcousticProperties</i></a>	12.4.1
<a href="#">Anchor</a>	9.4.1
<a href="#">Appearance</a>	12.4.2
<a href="#">Arc2D</a>	14.3.1
<a href="#">ArcClose2D</a>	14.3.2
<a href="#">AudioClip</a>	16.4.1
<a href="#">Background</a>	24.4.1
<a href="#">BallJoint</a>	37.4.1
<a href="#">Billboard</a>	23.4.1
<a href="#">BlendedVolumeStyle</a>	41.4.1
<a href="#">BooleanFilter</a>	30.4.1
<a href="#">BooleanSequencer</a>	30.4.2

<a href="#">BooleanToggle</a>	30.4.3
<a href="#">BooleanTrigger</a>	30.4.4
<a href="#">BoundaryEnhancementVolumeStyle</a>	41.4.2
<a href="#">BoundedPhysicsModel</a>	40.4.1
<a href="#">Box</a>	13.3.1
<a href="#">CADAssembly</a>	32.4.1
<a href="#">CADFace</a>	32.4.2
<a href="#">CADLayer</a>	32.4.3
<a href="#">CADPart</a>	32.4.4
<a href="#">CartoonVolumeStyle</a>	41.4.3
<a href="#">Circle2D</a>	14.3.3
<a href="#">ClipPlane</a>	11.4.1
<a href="#">CollidableOffset</a>	37.4.2
<a href="#">CollidableShape</a>	37.4.3
<a href="#">Collision</a>	23.4.2
<a href="#">CollisionCollection</a>	37.4.4
<a href="#">CollisionSensor</a>	37.4.5
<a href="#">CollisionSpace</a>	37.4.6
<a href="#">Color</a>	11.4.1
<a href="#">ColorChaser</a>	39.4.1
<a href="#">ColorDamper</a>	39.4.2
<a href="#">ColorInterpolator</a>	19.4.1
<a href="#">ColorRGBA</a>	11.4.2
<a href="#">ComposedCubeMapTexture</a>	34.4.1
<a href="#">ComposedShader</a>	31.4.1
<a href="#">ComposedTexture3D</a>	33.4.1

<a href="#">ComposedVolumeStyle</a>	41.4.4
<a href="#">Cone</a>	13.3.2
<a href="#">ConeEmitter</a>	40.4.2
<a href="#">Contact</a>	37.4.7
<a href="#">Contour2D</a>	27.4.1
<a href="#">ContourPolyline2D</a>	27.4.2
<a href="#">Coordinate</a>	11.4.3
<a href="#">CoordinateChaser</a>	39.4.3
<a href="#">CoordinateDamper</a>	39.4.4
<a href="#">CoordinateDouble</a>	27.4.3
<a href="#">CoordinateInterpolator</a>	19.4.2
<a href="#">CoordinateInterpolator2D</a>	19.4.3
<a href="#">Cylinder</a>	13.3.3
<a href="#">CylinderSensor</a>	20.4.1
<a href="#">DirectionalLight</a>	17.4.1
<a href="#">DISEntityManager</a>	28.3.1
<a href="#">DISEntityTypeMapping</a>	28.3.2
<a href="#">Disk2D</a>	14.3.4
<a href="#">DoubleAxisHingeJoint</a>	37.4.8
<a href="#">EaseInEaseOut</a>	19.4.4
<a href="#">EdgeEnhancementVolumeStyle</a>	41.4.5
<a href="#">ElevationGrid</a>	13.3.4
<a href="#">EspduTransform</a>	28.3.3
<a href="#">ExplosionEmitter</a>	40.4.3

<a href="#">Extrusion</a>	13.3.5
<a href="#">FillProperties</a>	12.4.3
<a href="#">FloatVertexAttribute</a>	31.4.2
<a href="#">Fog</a>	24.4.2
<a href="#">FogCoordinate</a>	24.4.3
<a href="#">FontStyle</a>	15.4.1
<a href="#">ForcePhysicsModel</a>	40.4.4
<a href="#">GeneratedCubeMapTexture</a>	34.4.2
<a href="#">GeoCoordinate</a>	25.3.1
<a href="#">GeoElevationGrid</a>	25.3.2
<a href="#">GeoLocation</a>	25.3.3
<a href="#">GeoLOD</a>	25.3.4
<a href="#">GeoMetadata</a>	25.3.5
<a href="#">GeoOrigin</a> (deprecated)	25.3.6
<a href="#">GeoPositionInterpolator</a>	25.3.7
<a href="#">GeoProximitySensor</a>	25.3.8
<a href="#">GeoTouchSensor</a>	25.3.9
<a href="#">GeoTransform</a>	25.3.10
<a href="#">GeoViewpoint</a>	25.3.11
<a href="#">Group</a>	10.4.1
<a href="#">HAnimDisplacer</a>	26.3.1
<a href="#">HAnimHumanoid</a>	26.3.2
<a href="#">HAnimJoint</a>	26.3.3
<a href="#">HAnimMotion</a>	26.3.4

<a href="#">HAnimSegment</a>	26.3.5
<a href="#">HAnimSite</a>	26.3.6
<a href="#">ImageCubeMapTexture</a>	34.4.3
<a href="#">ImageTexture</a>	18.4.1
<a href="#">ImageTexture3D</a>	33.4.2
<a href="#">IndexedFaceSet</a>	13.3.6
<a href="#">IndexedLineSet</a>	11.4.4
<a href="#">IndexedQuadSet</a>	32.4.5
<a href="#">IndexedTriangleFanSet</a>	11.4.5
<a href="#">IndexedTriangleSet</a>	11.4.6
<a href="#">IndexedTriangleStripSet</a>	11.4.7
<a href="#">Inline</a>	9.4.2
<a href="#">IntegerSequencer</a>	30.4.5
<a href="#">IntegerTrigger</a>	30.4.6
<a href="#">IsoSurfaceVolumeData</a>	41.4.6
<a href="#">KeySensor</a>	21.4.1
<a href="#">Layer</a>	35.4.1
<a href="#">LayerSet</a>	35.4.2
<a href="#">Layout</a>	36.4.1
<a href="#">LayoutGroup</a>	36.4.2
<a href="#">LayoutLayer</a>	36.4.3
<a href="#">LinePickSensor</a>	38.4.1
<a href="#">LineProperties</a>	12.4.4
<a href="#">LineSet</a>	11.4.8
<a href="#">LoadSensor</a>	9.4.3

<a href="#">LocalFog</a>	24.4.4
<a href="#">LOD</a>	23.4.3
<a href="#">Material</a>	12.4.5
<a href="#">Matrix3VertexAttribute</a>	31.4.3
<a href="#">Matrix4VertexAttribute</a>	31.4.4
<a href="#">MetadataBoolean</a>	7.4.1
<a href="#">MetadataDouble</a>	7.4.2
<a href="#">MetadataFloat</a>	7.4.3
<a href="#">MetadataInteger</a>	7.4.4
<a href="#">MetadataSet</a>	7.4.5
<a href="#">MetadataString</a>	7.4.6
<a href="#">MotorJoint</a>	37.4.8
<a href="#">MovieTexture</a>	18.4.2
<a href="#">MultiTexture</a>	18.4.3
<a href="#">MultiTextureCoordinate</a>	18.4.4
<a href="#">MultiTextureTransform</a>	18.4.5
<a href="#">NavigationInfo</a>	23.4.4
<a href="#">Normal</a>	11.4.9
<a href="#">NormalInterpolator</a>	19.4.5
<a href="#">NurbsCurve</a>	27.4.4
<a href="#">NurbsCurve2D</a>	27.4.5
<a href="#">NurbsOrientationInterpolator</a>	27.4.6
<a href="#">NurbsPatchSurface</a>	27.4.7
<a href="#">NurbsPositionInterpolator</a>	27.4.8

<a href="#">NurbsSet</a>	27.4.9
<a href="#">NurbsSurfaceInterpolator</a>	27.4.10
<a href="#">NurbsSweptSurface</a>	27.4.11
<a href="#">NurbsSwungSurface</a>	27.4.12
<a href="#">NurbsTextureCoordinate</a>	27.4.13
<a href="#">NurbsTrimmedSurface</a>	27.4.14
<a href="#">OpacityMapVolumeStyle</a>	41.4.7
<a href="#">OrientationChaser</a>	39.4.5
<a href="#">OrientationDamper</a>	39.4.6
<a href="#">OrientationInterpolator</a>	19.4.6
<a href="#">OrthoViewpoint</a>	23.4.5
<a href="#">PackedShader</a>	31.4.5
<a href="#">ParticleSystem</a>	40.4.5
<a href="#">PhysicalMaterial</a>	12.4.6
<a href="#">PickableGroup</a>	38.4.2
<a href="#">PixelTexture</a>	18.4.6
<a href="#">PixelTexture3D</a>	33.4.3
<a href="#">PlaneSensor</a>	20.4.2
<a href="#">PointEmitter</a>	40.4.6
<a href="#">PointLight</a>	17.4.2
<a href="#">PointPickSensor</a>	38.4.3
<a href="#">PointProperties</a>	12.4.7
<a href="#">PointSet</a>	11.4.10
<a href="#">Polyline2D</a>	14.3.5
<a href="#">PolylineEmitter</a>	40.4.7

<a href="#">Polypoint2D</a>	14.3.6
<a href="#">PositionChaser</a>	39.4.7
<a href="#">PositionChaser2D</a>	39.4.8
<a href="#">PositionDamper</a>	39.4.9
<a href="#">PositionDamper2D</a>	39.4.10
<a href="#">PositionInterpolator</a>	19.4.7
<a href="#">PositionInterpolator2D</a>	19.4.8
<a href="#">PrimitivePickSensor</a>	38.4.4
<a href="#">ProgramShader</a>	31.4.6
<a href="#">ProjectionVolumeStyle</a>	41.4.8
<a href="#">ProximitySensor</a>	22.4.1
<a href="#">QuadSet</a>	32.4.6
<a href="#">ReceiverPdu</a>	28.3.4
<a href="#">Rectangle2D</a>	14.3.7
<a href="#">RigidBody</a>	37.4.9
<a href="#">RigidBodyCollection</a>	37.4.10
<a href="#">ScalarChaser</a>	39.4.11
<a href="#">ScalarDamper</a>	39.4.12
<a href="#">ScalarInterpolator</a>	19.4.9
<a href="#">ScreenFontStyle</a>	36.4.4
<a href="#">ScreenGroup</a>	36.4.5
<a href="#">Script</a>	29.4.1
<a href="#">SegmentedVolumeData</a>	41.4.9
<a href="#">ShadedVolumeStyle</a>	41.4.10
<a href="#">ShaderPart</a>	31.4.7



<a href="#">ShaderProgram</a>	31.4.8
<a href="#">Shape</a>	12.4.8
<a href="#">SignalPdu</a>	28.3.5
<a href="#">SilhouetteEnhancementVolumeStyle</a>	41.4.11
<a href="#">SingleAxisHingeJoint</a>	37.4.11
<a href="#">SliderJoint</a>	37.4.12
<a href="#">Sound</a>	16.4.2
<a href="#">Sphere</a>	13.3.7
<a href="#">SphereSensor</a>	20.4.3
<a href="#">SplinePositionInterpolator</a>	19.4.10
<a href="#">SplinePositionInterpolator2D</a>	19.4.11
<a href="#">SplineScalarInterpolator</a>	19.4.12
<a href="#">SpotLight</a>	17.4.3
<a href="#">SquadOrientationInterpolator</a>	19.4.13
<a href="#">StaticGroup</a>	10.4.2
<a href="#">StringSensor</a>	21.4.2
<a href="#">SurfaceEmitter</a>	40.4.8
<a href="#">Switch</a>	10.4.3
<a href="#">TexCoordChaser2D</a>	39.4.13
<a href="#">TexCoordDamper2D</a>	39.4.14
<a href="#">Text</a>	15.4.2
<a href="#">TextureBackground</a>	24.4.3
<a href="#">TextureCoordinate</a>	18.4.7
<a href="#">TextureCoordinate3D</a>	33.4.4
<a href="#">TextureCoordinate4D</a>	33.4.5

<a href="#">TextureCoordinateGenerator</a>	18.4.8
<a href="#">TextureProjectorParallel</a>	42.4.1
<a href="#">TextureProjectorPerspective</a>	42.4.2
<a href="#">TextureProperties</a>	18.4.9
<a href="#">TextureTransform</a>	18.4.10
<a href="#">TextureTransform3D</a>	33.4.7
<a href="#">TextureTransformMatrix3D</a>	33.4.6
<a href="#">TimeSensor</a>	8.4.1
<a href="#">TimeTrigger</a>	30.4.7
<a href="#">ToneMappedVolumeStyle</a>	41.4.12
<a href="#">TouchSensor</a>	20.4.4
<a href="#">Transform</a>	10.4.4
<a href="#">TransformSensor</a>	22.4.2
<a href="#">TransmitterPdu</a>	28.3.6
<a href="#">TriangleFanSet</a>	11.4.11
<a href="#">TriangleSet</a>	11.4.12
<a href="#">TriangleSet2D</a>	14.3.8
<a href="#">TriangleStripSet</a>	11.4.13
<a href="#">TwoSidedMaterial</a>	12.4.9
<a href="#">UniversalJoint</a>	37.4.13
<a href="#">UnlitMaterial</a>	12.4.10
<a href="#">Viewpoint</a>	23.4.6
<a href="#">ViewpointGroup</a>	23.4.7
<a href="#">Viewport</a>	35.4.3
<a href="#">VisibilitySensor</a>	22.4.3

<a href="#">VolumeData</a>	41.4.13
<a href="#">VolumeEmitter</a>	40.4.7
<a href="#">VolumePickSensor</a>	38.4.5
<a href="#">WindPhysicsModel</a>	40.4.8
<a href="#">WorldInfo</a>	7.4.6
<a href="#">X3DAppearanceChildNode</a>	12.3.1
<a href="#">X3DAppearanceNode</a>	12.3.2
<a href="#">X3DBackgroundNode</a>	24.3.1
<a href="#">X3DBindableNode</a>	7.3.1
<a href="#">X3DBoundedObject</a>	10.3.1
<a href="#">X3DChaserNode</a>	39.3.1
<a href="#">X3DChildNode</a>	7.3.2
<a href="#">X3DColorNode</a>	11.3.1
<a href="#">X3DComposableVolumeRenderStyleNode</a>	41.3.1
<a href="#">X3DComposedGeometryNode</a>	11.3.2
<a href="#">X3DCoordinateNode</a>	11.3.3
<a href="#">X3DDamperNode</a>	39.3.2
<a href="#">X3DDragSensorNode</a>	20.3.1
<a href="#">X3DEnvironmentalSensorNode</a>	22.3.1
<a href="#">X3DEnvironmentTextureNode</a>	34.3.1
<a href="#">X3DFogObject</a>	24.3.2
<a href="#">X3DFollowerNode</a>	39.3.3
<a href="#">X3DFontStyleNode</a>	15.3.1
<a href="#">X3DGeometricPropertyNode</a>	11.3.4

<a href="#"><u>X3DGeometryNode</u></a>	11.3.5
<a href="#"><u>X3DGroupingNode</u></a>	10.3.2
<a href="#"><u>X3DInfoNode</u></a>	7.3.3
<a href="#"><u>X3DInterpolatorNode</u></a>	19.3.1
<a href="#"><u>X3DKeyDeviceSensorNode</u></a>	21.3.1
<a href="#"><u>X3DLayerNode</u></a>	35.3.1
<a href="#"><u>X3DLayoutNode</u></a>	36.3.1
<a href="#"><u>X3DLightNode</u></a>	17.3.1
<a href="#"><u>X3DMaterialNode</u></a>	12.3.3
<a href="#"><u>X3DMetadataObject</u></a>	7.3.3
<a href="#"><u>X3DNBodyCollidableNode</u></a>	37.3.1
<a href="#"><u>X3DNBodyCollisionSpaceNode</u></a>	37.3.2
<a href="#"><u>X3DNetworkSensorNode</u></a>	9.3.1
<a href="#"><u>X3DNode</u></a>	7.3.4
<a href="#"><u>X3DNormalNode</u></a>	11.3.6
<a href="#"><u>X3DNurbsControlCurveNode</u></a>	27.3.1
<a href="#"><u>X3DNurbsSurfaceGeometryNode</u></a>	27.3.2
<a href="#"><u>X3DOneSidedMaterialNode</u></a>	12.3.4
<a href="#"><u>X3DParametricGeometryNode</u></a>	27.3.3
<a href="#"><u>X3DParticleEmitterNode</u></a>	40.3.1
<a href="#"><u>X3DParticlePhysicsModelNode</u></a>	40.3.2
<a href="#"><u>X3DPickableObject</u></a>	38.3.1
<a href="#"><u>X3DPickSensorNode</u></a>	38.3.2
<a href="#"><u>X3DPointingDeviceSensorNode</u></a>	20.3.2
<a href="#"><u>X3DProductStructureChildNode</u></a>	32.3.1

<a href="#"><i>X3DProgrammableShaderObject</i></a>	31.3.1
<a href="#"><i>X3DPrototypeInstance</i></a>	7.3.5
<a href="#"><i>X3DRigidJointNode</i></a>	37.3.3
<a href="#"><i>X3DScriptNode</i></a>	29.3.1
<a href="#"><i>X3DSensorNode</i></a>	7.3.6
<a href="#"><i>X3DSequencerNode</i></a>	30.3.1
<a href="#"><i>X3DShaderNode</i></a>	31.3.2
<a href="#"><i>X3DShapeNode</i></a>	12.3.5
<a href="#"><i>X3DSoundNode</i></a>	16.3.1
<a href="#"><i>X3DSoundSourceNode</i></a>	16.3.2
<a href="#"><i>X3DSingleTextureCoordinateNode</i></a>	18.3.1
<a href="#"><i>X3DSingleTextureNode</i></a>	18.3.2
<a href="#"><i>X3DSingleTextureTransformNode</i></a>	18.3.3
<a href="#"><i>X3DTexture2DNode</i></a>	18.3.4
<a href="#"><i>X3DTexture3DNode</i></a>	33.3.1
<a href="#"><i>X3DTextureCoordinateNode</i></a>	18.3.5
<a href="#"><i>X3DTextureNode</i></a>	18.3.6
<a href="#"><i>X3DTextureProjectorNode</i></a>	42.3.1
<a href="#"><i>X3DTextureTransformNode</i></a>	18.3.7
<a href="#"><i>X3DTimeDependentNode</i></a>	8.3.1
<a href="#"><i>X3DTouchSensorNode</i></a>	20.3.3
<a href="#"><i>X3DTriggerNode</i></a>	30.3.2
<a href="#"><i>X3DUriObject</i></a>	9.3.2
<a href="#"><i>X3DVertexAttributeNode</i></a>	31.3.3

<a href="#"><u><i>X3DViewpointNode</i></u></a>	23.3.1
<a href="#"><u><i>X3DViewportNode</i></u></a>	35.3.2
<a href="#"><u><i>X3DVolumeDataNode</i></u></a>	41.3.2
<a href="#"><u><i>X3DVolumeRenderStyleNode</i></u></a>	41.3.3





## Extensible 3D (X3D) Part 1: Architecture and base components

### 18 Texturing component



#### 18.1 Introduction

##### 18.1.1 Name

The name of this component is "Texturing". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 18.1.2 Overview

This clause describes the Texturing component of this part of ISO/IEC 19775. This includes how textures are specified and how they are positioned on the subject geometry. [Table 18.1](#) provides links to the major topics in this clause.

**Table 18.1 — Topics**

- [18.1 Introduction](#)
  - [18.1.1 Name](#)
  - [18.1.2 Overview](#)
- [18.2 Concepts](#)
  - [18.2.1 Texture map formats](#)
  - [18.2.2 Texture map image formats](#)
  - [18.2.3 Texture coordinates](#)
  - [18.2.4 Multitexturing](#)
  - [18.2.5 Programmable shaders](#)
- [18.3 Abstract types](#)
  - [18.3.1 X3DSingleTextureCoordinateNode](#)
  - [18.3.2 X3DSingleTextureNode](#)
  - [18.3.3 X3DSingleTextureTransformNode](#)
  - [18.3.4 X3DTexture2DNode](#)
  - [18.3.5 X3DTextureCoordinateNode](#)
  - [18.3.6 X3DTextureNode](#)
  - [18.3.7 X3DTextureTransformNode](#)
- [18.4 Node reference](#)
  - [18.4.1 ImageTexture](#)
  - [18.4.2 MovieTexture](#)

- [18.4.3 MultiTexture](#)
- [18.4.4 MultiTextureCoordinate](#)
- [18.4.5 MultiTextureTransform](#)
- [18.4.6 PixelTexture](#)
- [18.4.7 TextureCoordinate](#)
- [18.4.8 TextureCoordinateGenerator](#)
- [18.4.9 TextureProperties](#)
- [18.4.10 TextureTransform](#)
- [18.5 Support levels](#)
- [Figure 18.1 — Texture map coordinate system](#)
- [Figure 18.2 — Lightmap example](#)
- [Table 18.1 — Topics](#)
- [Table 18.2 — Comparison of single texture and multitexture attributes](#)
- [Table 18.3 — Multitexture modes](#)
- [Table 18.4 — Values for the \*source\* field](#)
- [Table 18.5 — Values for the \*function\* field](#)
- [Table 18.6 — Texture coordinate generation modes](#)
- [Table 18.7 — Texture boundary modes](#)
- [Table 18.8 — Texture magnification modes](#)
- [Table 18.9 — Texture minification modes](#)
- [Table 18.10 — Texture compression modes](#)
- [Table 18.11 — Texturing component support levels](#)

## 18.2 Concepts

### 18.2.1 Texture map formats

Node types specifying texture maps include [Background](#), [ImageTexture](#), [MovieTexture](#), [MultiTexture](#), [PixelTexture](#), descendants of [X3DEnvironmentTextureNode](#), descendants of [X3DTexture3DNode](#). Texture maps are 2D or 3D or cubemap images that contain an array of colour values describing the texture.

Depending on the number of channels, the following texture types are possible:

- a. *Intensity textures* (one channel)
- b. *Intensity plus alpha opacity textures* (two channels)
- c. *Full RGB textures* (three channels)
- d. *Full RGB plus alpha opacity textures* (four channels)

Note that image formats specify alpha (*i.e.*, opacity), not transparency (where  $\text{alpha} = 1 - \text{transparency}$ ).

See [17.2.2.2 Texture sampling](#) for a description of how the various texture types are applied.

The textures described in this component, "*Texturing*", only support two-dimensional map formats. See [33 Texturing3D component](#) for a description of the use of 3D textures and [34 Cube map environmental texture component](#) for a description of the use of cube map



textures.

## 18.2.2 Texture map image formats

Texture nodes that require support for the PNG (see [2.\[115948\]](#)) image format shall interpret the PNG pixel formats in the following way:

- a. Greyscale pixels without alpha or simple transparency are treated as intensity textures.
- b. Greyscale pixels with alpha or simple transparency are treated as intensity plus alpha textures.
- c. RGB pixels without alpha channel or simple transparency are treated as full RGB textures.
- d. RGB pixels with alpha channel or simple transparency are treated as full RGB plus alpha textures.

If the image specifies colours as indexed-colour (*i.e.*, palettes or colourmaps), the following semantics shall be used (where `greyscale' refers to a palette entry with equal red, green, and blue values):

- e. If all the colours in the palette are greyscale and there is no transparency chunk, it is treated as an intensity texture.
- f. If all the colours in the palette are greyscale and there is a transparency chunk, it is treated as an intensity plus opacity texture.
- g. If any colour in the palette is not grey and there is no transparency chunk, it is treated as a full RGB texture.
- h. If any colour in the palette is not grey and there is a transparency chunk, it is treated as a full RGB plus alpha texture.

Texture nodes that require support for JPEG files (see [2.\[JPEG\]](#)) shall interpret JPEG files as follows:

- i. Greyscale files (number of components equals 1) are treated as intensity textures.
- j. YCbCr files are treated as full RGB textures.
- k. No other JPEG file types are required. It is recommended that other JPEG files are treated as a full RGB textures.

Texture nodes that support MPEG files (see [ISO/IEC 11172-1](#)) shall treat MPEG files as full RGB textures.

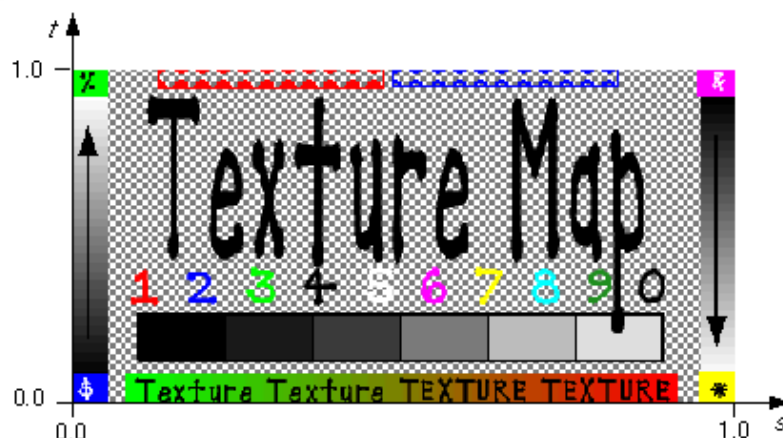
Texture nodes that recommend support for GIF files (see [\[GIF\]](#)) shall follow the applicable semantics described above for the PNG format.

Texture nodes that recommend support for Joint Photographic Experts Group (JPEG) 2000, Geographic Tagged Image File Format (GeoTIFF), National Imagery Transmission Format (NITF) or Basic Image Interchange Format (BIIF) formats shall follow the applicable semantics described above for the PNG format.

## 18.2.3 Texture coordinates

Texture maps are defined in a 2D coordinate system (s, t) that ranges from [0.0, 1.0] in both directions. The bottom edge of the image corresponds to the S-axis of the texture map, and left edge of the image corresponds to the T-axis of the texture map. The lower-left pixel of the image corresponds to s=0, t=0, and the top-right pixel of the image corresponds to

$s=1, t=1$ . Texture maps may be viewed as two dimensional colour functions that, given an  $(s, t)$  coordinate, return a colour value  $colour(s, t)$ . These relationships are depicted in [Figure 18.1](#).



**Figure 18.1 — Texture map coordinate system**

The texture map nodes [ImageTexture](#), [MovieTexture](#), and [PixelTexture](#) contain two fields, *repeatS* and *repeatT*, that specify how the texture wraps in the S and T directions. If *repeatS* is `TRUE` (the default), the texture map is repeated outside the  $[0.0, 1.0]$  texture coordinate range in the S direction so that it fills the shape. If *repeatS* is `FALSE`, the texture coordinates are clamped in the S direction to lie within the  $[0.0, 1.0]$  range. The *repeatT* field is analogous to the *repeatS* field.

Textures nodes with a *textureProperties* field allow fined grained control of the texture setup including further modes for handling clamping and repeating texture coordinates and specifying how a texture should be filtered. Texture nodes with a provided [TextureProperties](#) node shall ignore the settings of *repeatS* and *repeatT* and shall use the provided values in the *boundaryMode* fields.

Each vertex-based geometry node (e.g., [IndexedFaceSet](#) and [ElevationGrid](#)) uses a set of 2D texture coordinates that map textures to vertices. Texture coordinates for geometry nodes are specified using the [TextureCoordinate](#) and [TextureCoordinateGenerator](#) nodes. Texture map values ([ImageTexture](#), [MovieTexture](#), and [PixelTexture](#)) range from  $[0.0, 1.0]$  along the S-axis and T-axis. However, texture coordinate values may be in the range  $(-\infty, \infty)$ . Texture coordinates identify a location (and thus a colour value) in the texture map. The horizontal coordinate *s* is specified first, followed by the vertical coordinate *t*.

If the texture map is repeated in a given direction (S-axis or T-axis), a texture coordinate *C* (*s* or *t*) is mapped into a texture map that has *N* pixels in the given direction as follows:

$$\text{Texture map location} = (C - \text{floor}(C)) \times N$$

If the texture map is not repeated, the texture coordinates are clamped to the 0.0 to 1.0 range as follows:

$$\begin{aligned} \text{Texture map location} &= N, && \text{if } C > 1.0, \\ &= 0.0, && \text{if } C < 0.0, \\ &= C \times N, && \text{if } 0.0 \leq C \leq 1.0. \end{aligned}$$

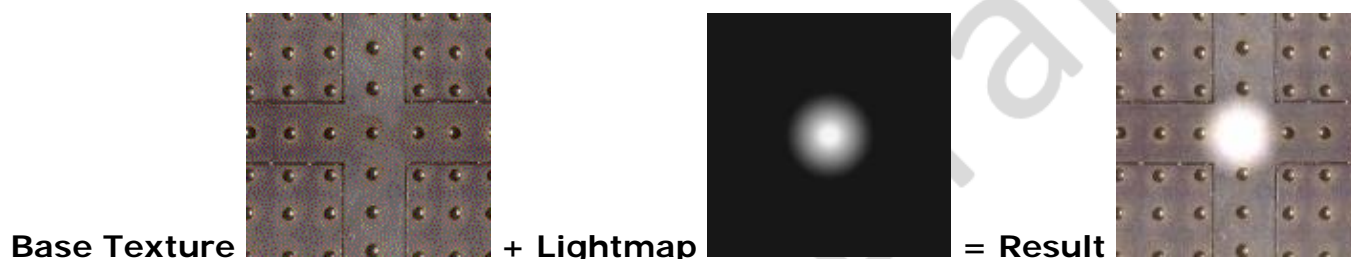
Texture coordinates may be transformed (scaled, rotated, translated) by supplying a [TextureTransform](#) node as a component of the texture's containing [Appearance](#) node.

Details on repeating textures are specific to texture map node types described in [ImageTexture](#), [MovieTexture](#), and [PixelTexture](#).

## 18.2.4 Multitexturing

Multiple textures may be applied to a single geometry node and blended according to a predefined set of operations. This enables a variety of visual effects that include light mapping and environment mapping. Multiple textures may be applied using multi-stage or multi-pass techniques, depending upon the available hardware. The number of textures to be blended may have a significant impact on performance, depending upon the available hardware.

[Figure 18.2](#) depicts an example of light mapping, simulating a pre-lit object. Texture 2 is added on top of texture 1.



**Figure 18.2 — Lightmap example**

Multitexturing is accomplished using the [MultiTexture](#), [MultiTextureCoordinate](#), and [MultiTextureTransform](#) nodes. [MultiTexture](#) specifies a grouping of single textures and texture transformations. [MultiTextureCoordinate](#) specifies a grouping of texture coordinates to be used with the associated textures. [MultiTextureTransform](#) specifies a grouping of texture transforms to be used with the associated textures.

[Table 18.2](#) compares the usage of single texture and multitexture attributes within [Appearance](#) and geometry nodes.

**Table 18.2: Comparison of single texture and multitexture attributes**

Texture Node <code>appearance.texture</code>	Texture Transform	Texture coordinate <code>geometry.texCoord</code>	Texture mode
<code>ImageTexture { ... }</code>	<code>appearance.textureTransform</code> <code>TextureTransform { }</code>	<code>TextureCoordinate {</code> <code>  coord [ ] }</code>	implicit in lighting model: [ "REPLACE" "MODULATE" ]
<code>MultiTexture {</code> <code>  texture [</code> <code>    ImageTexture { ... }</code> <code>    ImageTexture { ... }</code> <code>  ]}</code>	<code>MultiTexture {</code> <code>  textureTransform [</code> <code>    TextureTransform { ... }</code> <code>    TextureTransform { ... }</code> <code>  ]}</code>	<code>MultiTextureCoordinate</code> <code>{</code> <code>  coord [</code> <code>    TextureCoordinate {</code> <code>      coord [ ]</code> <code>    }TextureCoordinate {</code> <code>      coord [ ] }</code> <code>  ]}</code>	<code>MultiTexture</code> <code>{</code> <code>  mode {</code> <code>    "MODULATE"</code> <code>    "MODULATE"</code> <code>  }</code> <code>}</code>

## 18.2.5 Programmable shaders

If a programmable shader is defined for the [Appearance](#) node containing textures, texture

mapping shall be disabled. Textures defined shall be considered as sources of input and/or output for a programmable shader. See [31.2.2.5 Per-object attributes](#) for details on how to map textures to shader program inputs.

## 18.3 Abstract types

### 18.3.1 X3DSingleTextureCoordinateNode

```
X3DSingleTextureCoordinateNode : X3DTextureCoordinateNode {
  SFString [in,out] mapping ""
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This abstract node type is the base type for all texture coordinate nodes which specify texture coordinates for a single texture. See [12.2.4 Texture mapping specified in material nodes](#) for a description how it interacts with texture specification inside materials.

This abstract type means *"any texture coordinate node except [MultiTextureCoordinate](#)"*.

### 18.3.2 X3DSingleTextureNode

```
X3DSingleTextureNode : X3DTextureNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This abstract node type is the base type for all texture node types that define a single texture. A single texture can be used to influence a parameter of various material nodes in the [Shape component](#), and it can be a child of [MultiTexture](#).

This abstract type means *"any texture node except [MultiTexture](#)"*.

### 18.3.3 X3DSingleTextureTransformNode

```
X3DSingleTextureTransformNode : X3DTextureTransformNode {
  SFString [in,out] mapping ""
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This abstract node type is the base type for all texture transform nodes which specify texture coordinate transformation for a single texture. See [12.2.4 Texture mapping specified in material nodes](#) for a description how it interacts with texture specification inside materials.

This abstract type means *"any texture transformation node except [MultiTextureTransform](#)"*.

### 18.3.4 X3DTexture2DNode

```
X3DTexture2DNode : X3DSingleTextureNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFBool [] repeatS TRUE
  SFBool [] repeatT TRUE
  SFNode [] textureProperties NULL [TextureProperties]
}
```

This abstract node type is the base type for all node types which specify 2D sources for texture images.

### 18.3.5 X3DTextureCoordinateNode

```
X3DTextureCoordinateNode : X3DGeometricPropertyNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This abstract node type is the base type for all node types which specify texture coordinates. It adds a new geometric property node type to those specified in [11 Rendering component](#).

### 18.3.6 X3DTextureNode

```
X3DTextureNode : X3DAppearanceChildNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This abstract node type is the base type for all node types which specify sources for texture images.

### 18.3.7 X3DTextureTransformNode

```
X3DTextureTransformNode : X3DAppearanceChildNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This abstract node type is the base type for all node types which specify a transformation of texture coordinates.

## 18.4 Node reference

### 18.4.1 ImageTexture

```
ImageTexture : X3DTexture2DNode, X3DUrlObject {
  SFString [in,out] description ""
  SFBool [in,out] load TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTime [in,out] refresh 0.0 [0,∞)
  MFString [in,out] url [] [URI]
  SFBool [] repeatS TRUE
  SFBool [] repeatT TRUE
  SFNode [] textureProperties NULL [TextureProperties]
}
```

The ImageTexture node defines a texture map by specifying an image file and general parameters for mapping to geometry.

The texture is read from the URL specified by the *url* field. When the *url* field contains no values ([ ]), texturing is disabled. Browsers shall support the JPEG (see [2. \[JPEG\]](#)) and PNG (see [ISO/IEC 15948](#)) image file formats. In addition, browsers may support other image formats (EXAMPLE CGM, [ISO/IEC 8632](#)) that can be rendered into a 2D image. Support for the GIF format (see [\[GIF\]](#)) is also recommended (including transparency). Details on the *url* field can be found in [9.2.1 URLs](#).

See [18.2 Concepts](#), for a general description of texture maps.

See [17 Lighting component](#) for a description of lighting equations and the interaction between textures, materials, and geometry appearance.

### 18.4.2 MovieTexture

```
MovieTexture : X3DTexture2DNode, X3DSoundSourceNode, X3DUrlObject {
  SFString [in,out] description ""
  SFBool [in,out] load TRUE
  SFBool [in,out] loop FALSE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTime [in,out] pauseTime 0 (-∞,∞)
  SFFloat [in,out] pitch 1.0 (0,∞)
  SFTime [in,out] refresh 0.0 [0,∞)
  SFTime [in,out] resumeTime 0 (-∞,∞)
  SFFloat [in,out] speed 1.0 (-∞,∞)
  SFTime [in,out] startTime 0 (-∞,∞)
  SFTime [in,out] stopTime 0 (-∞,∞)
  MFString [in,out] url [] [URI]
  SFTime [out] duration_changed
  SFTime [out] elapsedTime
```



```

SFBool [out] isActive
SFBool [out] isPaused
SFBool [] repeatS TRUE
SFBool [] repeatT TRUE
SFNode [] textureProperties NULL [TextureProperties]
}

```

The `MovieTexture` node defines a time dependent texture map (contained in a movie file) and parameters for controlling the movie and the texture mapping. A `MovieTexture` node can also be used as the source of sound data for a [Sound](#) node. In this special case, the `MovieTexture` node is not used for rendering.

The `url` field that defines the movie data shall support MPEG1-Systems (audio and video) or MPEG1-Video (video-only) movie file formats as defined in [ISO/IEC 11172-1](#). Details on the `url` field can be found in [9.2.1 URLs](#).

`MovieTexture` nodes can be referenced by an [Appearance](#) node's `texture` field (as a movie texture) and by a [Sound](#) node's `source` field (as an audio source only).

As soon as the movie is loaded, a `duration_changed` field is sent. This indicates the duration of the movie in seconds. This field value can be read (for instance, by a [Script](#) node) to determine the duration of a movie. A value of "-1" implies the movie has not yet loaded or the value is unavailable for some reason.

The `loop`, `pauseTime`, `resumeTime`, `startTime`, and `stopTime` inputOutput fields and the `elapsedTime`, `isActive`, and `isPaused` outputOnly fields, and their effects on the `MovieTexture` node, are discussed in detail in [8 Time component](#). The cycle of a `MovieTexture` node is the length of time in seconds for one playing of the movie at the specified `speed`.

The `speed` field indicates how fast the movie shall be played. A `speed` of 2 indicates the movie plays twice as fast. The `duration_changed` output is not affected by the `speed` field. `set_speed` events are ignored while the movie is playing. A negative `speed` implies that the movie will play backwards.

If a `MovieTexture` node is inactive when the movie is first loaded, frame 0 of the movie texture is displayed if `speed` is non-negative or the last frame of the movie texture is shown if `speed` is negative (see [8.2.4 Time-dependent nodes](#)). A `MovieTexture` node shall display frame 0 if `speed` = 0. For positive values of `speed`, an active `MovieTexture` node displays the frame at movie time  $t$  as follows (*i.e.*, in the movie's local time system with frame 0 at time 0 with `speed` = 1):

$$t = (\text{now} - \text{startTime}) \bmod (\text{duration}/\text{speed})$$

If `speed` is negative, the `MovieTexture` node displays the frame at movie time:

$$t = \text{duration} - ((\text{now} - \text{startTime}) \bmod |\text{duration}/\text{speed}|)$$

When a `MovieTexture` node becomes inactive, the frame corresponding to the time at which the `MovieTexture` became inactive will remain as the texture.

See [18.2 Concepts](#), for a general description of texture maps.

[17 Lighting component](#) contains details on lighting equations and the interaction between textures, materials, and geometries.

### 18.4.3 MultiTexture

```

MultiTexture : X3DTextureNode {
  SFFloat [in,out] alpha 1 [0,1]
  SFColor [in,out] color 1 1 1 [0,1]
  MFString [in,out] function []
}

```

```

SFNode [in,out] metadata NULL [X3DMetadataObject]
MFString [in,out] mode []
MFString [in,out] source []
MFNode [in,out] texture [] [X3DSingleTextureNode]
MFNode [in,out] texture [] [X3DTextureNode]
}

```

The MultiTexture node specifies the application of several individual textures to a 3D object to achieve a more complex visual effect. MultiTexture can be used as a value for the texture field in an [Appearance](#) node.

The *texture* field contains a list of texture nodes (e.g., [ImageTexture](#), [PixelTexture](#), and [MovieTexture](#)). The texture field may not contain another MultiTexture node.

The *color* and *alpha* fields define base RGB and alpha values for `SELECT` mode operations.

The *mode* field controls the type of blending operation. The available modes include `MODULATE` for a lit Appearance, `REPLACE` for an unlit Appearance, and several variations of the two. The value chosen for the *mode* field may also specify the blending mode for the alpha channel.

EXAMPLE The mode value `"MODULATE"`, `"REPLACE"` specifies  $\text{Color} = (\text{Arg1.color} \times \text{Arg2.color}, \text{Arg1.alpha})$ .

The number of used texture stages is determined by the length of the texture field. If there are fewer mode values, the default mode is `"MODULATE"`.

[Table 18.3](#) lists possible multitexture modes.

**Table 18.3 — Multitexture modes**

MODE	Description
<code>"MODULATE"</code> (default)	Multiply texture color with current color $\text{Arg1} \times \text{Arg2}$
<code>"REPLACE"</code>	Replace current color $\text{Arg2}$
<code>"MODULATE2X"</code>	Multiply the components of the arguments, and shift the products to the left 1 bit (effectively multiplying them by 2) for brightening.
<code>"MODULATE4X"</code>	Multiply the components of the arguments, and shift the products to the left 2 bits (effectively multiplying them by 4) for brightening.
<code>"ADD"</code>	Add the components of the arguments $\text{Arg1} + \text{Arg2}$
<code>"ADDSIGNED"</code>	Add the components of the arguments with a -0.5 bias, making the effective range of values from -0.5 through 0.5.
<code>"ADDSIGNED2X"</code>	Add the components of the arguments with a -0.5 bias, and shift the products to the left 1 bit.
<code>"SUBTRACT"</code>	Subtract the components of the second argument from those of the first argument. $\text{Arg1} - \text{Arg2}$
	Add the first and second arguments, then subtract their

"ADDSMOOTH"	product from the sum. $\text{Arg1} + \text{Arg2} - \text{Arg1} \times \text{Arg2} = \text{Arg1} + (1 - \text{Arg1}) \times \text{Arg2}$
"BLENDDIFFUSEALPHA"	Linearly blend this texture stage, using the interpolated alpha from each vertex. $\text{Arg1} \times (\text{Alpha}) + \text{Arg2} \times (1 - \text{Alpha})$
"BLENDTEXTUREALPHA"	Linearly blend this texture stage, using the alpha from this stage's texture. $\text{Arg1} \times (\text{Alpha}) + \text{Arg2} \times (1 - \text{Alpha})$
"BLENDFACTORALPHA"	Linearly blend this texture stage, using the alpha factor from the MultiTexture node. $\text{Arg1} \times (\text{Alpha}) + \text{Arg2} \times (1 - \text{Alpha})$
"BLENDCURRENTALPHA"	Linearly blend this texture stage, using the alpha taken from the previous texture stage. $\text{Arg1} \times (\text{Alpha}) + \text{Arg2} \times (1 - \text{Alpha})$
"MODULATEALPHA_ADDCOLOR"	Modulate the color of the second argument, using the alpha of the first argument; then add the result to argument one. $\text{Arg1}.\text{RGB} + \text{Arg1}.\text{A} \times \text{Arg2}.\text{RGB}$
"MODULATEINVALPHA_ADDCOLOR"	Similar to MODULATEALPHA_ADDCOLOR, but use the inverse of the alpha of the first argument. $(1 - \text{Arg1}.\text{A}) \times \text{Arg2}.\text{RGB} + \text{Arg1}.\text{RGB}$
"MODULATEINVCOLOR_ADDALPHA"	Similar to MODULATEALPHA_ADDALPHA, but use the inverse of the color of the first argument. $(1 - \text{Arg1}.\text{RGB}) \times \text{Arg2}.\text{RGB} + \text{Arg1}.\text{A}$
"OFF"	Turn off the texture unit
"SELECTARG1"	Use color argument 1 Arg1
"SELECTARG2"	Use color argument 1 Arg2
"DOTPRODUCT3"	Modulate the components of each argument (as signed components), add their products, then replicate the sum to all color channels, including alpha. This can do either diffuse or specular bump mapping with correct input. Performs the function $(\text{Arg1}.\text{R} \times \text{Arg2}.\text{R} + \text{Arg1}.\text{G} \times \text{Arg2}.\text{G} + \text{Arg1}.\text{B} \times \text{Arg2}.\text{B})$ where each component has been scaled and offset to make it signed. The result is replicated into all four (including alpha) channels.

The *source* field determines the colour source for the second argument. [Table 18.4](#) lists valid values for the *source* field. Typically, there are the same number of *source* field values as textures. Otherwise, the default *source* field value is used.

**Table 18.4 — Values for the *source* field**

MODE	Description



" " (default)	The second argument color (ARG2) is the color from the previous rendering stage (DIFFUSE for first stage).
"DIFFUSE"	The texture argument is the diffuse color interpolated from vertex components during Gouraud shading.
"SPECULAR"	The texture argument is the specular color interpolated from vertex components during Gouraud shading.
"FACTOR"	The texture argument is the factor (color, alpha) from the texture provided for the current stage of the MultiTexture node.

The *function* field defines an optional function to be applied to the argument after the mode has been evaluated. [Table 18.5](#) lists valid values for the *function* field. Typically, there are the same number of *function* field values as textures. Otherwise, the default *function* field value is used.

**Table 18.5 — Values for the *function* field**

Operator	Description
"" (default)	No function is applied.
"COMPLEMENT"	Invert the argument so that, if the result of the argument were referred to by the variable x, the value would be 1.0 minus x.
"ALPHAREPLICATE"	Replicate the alpha information to all color channels before the operation completes.

## 18.4.4 MultiTextureCoordinate

```
MultiTextureCoordinate : X3DTextureCoordinateNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFNode [in,out] texCoord NULL [X3DSingleTextureCoordinateNode]
  MFNode [in,out] texCoord NULL [X3DTextureCoordinateNode]
}
```

MultiTextureCoordinate supplies multiple texture coordinates per vertex. This node can be used to set the texture coordinates for the different texture channels. **It can be used to provide texture coordinates:**

- For the texture specified in the *Appearance.texture* field. This includes a *MultiTexture* node. In this case the order of the texture coordinates must match the order of texture nodes within the *MultiTexture.texture* list.
- For any texture specified within material nodes using fields like *Material.diffuseTexture*, *PhysicalMaterial.baseTexture*. In this case, the *mapping* field of the child [X3DSingleTextureCoordinateNode](#) node must correspond to the appropriate *xxxTextureMapping* value in the material. See [12.2.4 Texture mapping specified in material nodes](#) for details.

Each entry in the texCoord field may contain a [TextureCoordinate](#), [TextureCoordinateGenerator](#) or other [X3DSingleTextureCoordinateNode](#) descendant node.

Example:

```
Shape {
```

```

appearance Appearance {
  texture MultiTexture {
    mode [ "MODULATE" "MODULATE" ]
    texture [
      ImageTexture { url "brick.jpg" }
      ImageTexture { repeatS FALSE repeatT FALSE url "light_gray.png" }
    ]
  }
  geometry IndexedFaceSet {
    ...
    texCoord MultiTextureCoordinate {
      texCoord [
        TextureCoordinate { ... }
        TextureCoordinate { ... }
      ]
    }
  }
}

```

If using a *MultiTexture* node with a geometry node without a *MultiTextureTransform* node, identity matrices are assumed for all channels. If there are too few entries in the *textureTransform* field, identity matrices shall be used for all remaining undefined channels.

Using a *MultiTextureCoordinate* with exactly one child is always equivalent to using this child directly. That is, these two constructs (in X3D classic encoding) are *exactly equivalent* for the purpose of texture coordinate determination:

1. 

```
IndexedFaceSet {
  texCoord TextureCoordinate {
    point [ 0 0, 1 1 ]
  }
}
```
2. 

```
IndexedFaceSet {
  texCoord MultiTextureCoordinate {
    texCoord [
      TextureCoordinate {
        point [ 0 0, 1 1 ]
      }
    ]
  }
}
```

When the *MultiTexture* node is used in *Appearance.texture* field, and there is not enough texture coordinates in the *MultiTextureCoordinate*, texture coordinates for the last channel are replicated along the other channels.

## 18.4.5 MultiTextureTransform

```

MultiTextureTransform : X3DTextureTransformNode {
  SFNode [in.out] metadata NULL [X3DMetadataObject]
  MFNode [in.out] textureTransform NULL [X3DSingleTextureTransformNode]
  MFNode [in.out] textureTransform NULL [X3DTextureTransformNode]
}

```

*MultiTextureTransform* supplies multiple texture transforms per appearance. This node can be used to set the texture transform for each of the different texture channels. It can be used to transform texture coordinates:

- For the texture specified in the *Appearance.texture* field. This includes a *MultiTexture* node. In this case the order of the texture transformations must match the order of texture nodes within the *MultiTexture.texture* list.
- For any texture specified within material nodes using fields like *Material.diffuseTexture*, *PhysicalMaterial.baseTexture*. In this case, the *mapping* field of the child [X3DSingleTextureTransformNode](#) node must correspond to the appropriate *xxxTextureMapping* value in the material. See [12.2.4 Texture mapping specified in material nodes](#) for details.

Each entry in the *textureTransform* field shall contain an [X3DSingleTextureTransformNode](#) or NULL.

Example:

```
Shape {
  appearance Appearance {
    texture MultiTexture {
      mode [ "MODULATE" "MODULATE" ]
      texture [
        ImageTexture { url "brick.jpg" }
        ImageTexture { repeatS FALSE repeatT FALSE url "light_gray.png" }
      ]
    }
    textureTransform MultiTextureTransform {
      textureTransform [
        TextureTransform {}
        TextureTransform { scale 0.5 0.5 }
      ]
    }
  }
}
```

If using `MultiTexture` with an `IndexedFaceSet` without a `MultiTextureTransform` node, texture coordinates for channel 0 are replicated along the other channels. Similarly, if there are too few entries in the `textureTransform` field, the last entry is replicated.

Note that we treat a `MultiTextureTransform` with a single child always the same as using this child directly. That, is these two constructs are equivalent, for the purpose of texture transformation determination:

```
1. Appearance {
  textureTransform TextureTransform {
    scale 10 10
  }
}

2. Appearance {
  textureTransform MultiTextureTransform {
    textureTransform [
      TextureTransform {
        scale 10 10
      }
    ]
  }
}
```

When the `MultiTexture` node is used in `Appearance.texture` field, and there is not enough texture coordinates in the `MultiTextureTransform`, identity matrices (no transformation) shall be used for all remaining channels.

## 18.4.6 PixelTexture

```
PixelTexture : X3DTexture2DNode {
  SFImage [in,out] image      0 0 0
  SFNode [in,out] metadata    NULL [X3DMetadataObject]
  SFBool [] repeatS          TRUE
  SFBool [] repeatT          TRUE
  SFNode [] textureProperties NULL [TextureProperties]
}
```

The `PixelTexture` node defines a 2D image-based texture map as an explicit array of pixel values (`image` field) and parameters controlling tiling repetition of the texture onto geometry.

The `repeatS` and `repeatT` fields specify how the texture wraps in the S and T directions. If `repeatS` is `TRUE` (the default), the texture map is repeated outside the 0-to-1 texture coordinate range in the S direction so that it fills the shape. If `repeatS` is `FALSE`, the texture coordinates are clamped in the S direction to lie within the 0.0 to 1.0 range. The `repeatT` field is analogous to the `repeatS` field.

See [18.2 Concepts](#), for a general description of texture maps.

See [17 Lighting component](#) for a description of how the texture values interact with the

appearance of the geometry. [5.7 SFImage and MFImage](#) describes the specification of an image.

## 18.4.7 TextureCoordinate

```
TextureCoordinate : X3DSingleTextureCoordinateNode {
TextureCoordinate : X3DTextureCoordinateNode {
  SFString [in,out] mapping ""
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFVec2f [in,out] point [] (-∞,∞)
}
```

The TextureCoordinate node is a geometry property node that specifies a set of 2D texture coordinates used by vertex-based geometry nodes (EXAMPLE [IndexedFaceSet](#) and [ElevationGrid](#)) to map textures to vertices.

## 18.4.8 TextureCoordinateGenerator

```
TextureCoordinateGenerator : X3DSingleTextureCoordinateNode {
TextureCoordinateGenerator : X3DTextureCoordinateNode {
  SFString [in,out] mapping ""
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFString [in,out] mode "SPHERE" [see Table 18.6]
  MFFloat [in,out] parameter [] [see Table 18.6]
}
```

TextureCoordinateGenerator supports the automatic generation of texture coordinates for geometric shapes.

This node can be used to set the texture coordinates for a node with a texCoord field.

The *mode* field describes the algorithm used to compute texture coordinates, as depicted in [Table 18.6](#).

**Table 18.6 — Texture coordinate generation modes**

Mode	Description
"SPHERE"	Creates texture coordinates for a spherical environment or "chrome" mapping based on the vertex normals transformed to camera space. $u = N_x/2 + 0.5$ $v = N_y/2 + 0.5$ where $u$ and $v$ are the texture coordinates being computed, and $N_x$ and $N_y$ are the $x$ and $y$ components of the camera-space vertex normal. If the normal has a positive $x$ component, the normal points to the right, and the $u$ coordinate is adjusted to address the texture appropriately. Likewise for the $v$ coordinate: positive $y$ indicates that the normal points up. The opposite is of course true for negative values in each component. If the normal points directly at the camera, the resulting coordinates should receive no distortion. The $+0.5$ bias to both coordinates places the point of zero-distortion at the center of the sphere map, and a vertex normal of $(0, 0, z)$ addresses this point. Note that this formula doesn't take account for the $z$ component of the normal.
"CAMERASPACE NORMAL"	Use the vertex normal, transformed to camera space, as input texture coordinates, resulting coordinates are in $-1$ to $1$ range.

"CAMERASPACEPOSITION"	Use the vertex position, transformed to camera space, as input texture coordinates
"CAMERASPACEREFLECTIONVECTOR"	Use the reflection vector, transformed to camera space, as input texture coordinates. The reflection vector is computed from the input vertex position and normal vector. $R = 2 \times \text{DotProd}(E, N) \times N - E$ ; In the preceding formula, R is the reflection vector being computed, E is the normalized position-to-eye vector, and N is the camera-space vertex normal. Resulting coordinates are in -1 to 1 range.
"SPHERE-LOCAL"	Sphere mapping but in local coordinates
"COORD"	use vertex coordinates
"COORD-EYE"	use vertex coordinates transformed to camera space
"NOISE"	computed by applying Perlin solid noise function on vertex coordinates, parameter contains scale and translation [ <i>scale.x scale.y scale.z translation.x translation.y translation.z</i> ]
"NOISE-EYE"	same as above but transform vertex coordinates to camera space first
"SPHERE-REFLECT"	similar to "CAMERASPACEREFLECTIONVECTOR" with optional index of refraction, parameter[0] contains index of refraction  Resulting coordinates are in -1 to 1 range.
"SPHERE-REFLECT-LOCAL"	Similar to "SPHERE-REFLECT", parameter[0] contains index of refraction, parameter[1 to 3] the eye point in local coordinates. By animating parameter [1 to 3] the reflection changes with respect to the point. Resulting coordinates are in -1 to 1 range.

Some modes may be hardware accelerated. Some modes are view dependent.

## 18.4.9 TextureProperties

```
TextureProperties : X3DNode
  SFFloat [in,out] anisotropicDegree 1.0 [1,∞)
  SFColorRGBA [in,out] borderColor 0 0 0 0 [0,1]
  SFInt32 [in,out] borderWidth 0 [0,1]
  SFString [in,out] boundaryModeS "REPEAT" [see Table 18.7]
  SFString [in,out] boundaryModeT "REPEAT" [see Table 18.7]
  SFString [in,out] boundaryModeR "REPEAT" [see Table 18.7]
  SFString [in,out] magnificationFilter "FASTEST" [see Table 18.8]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFString [in,out] minificationFilter "FASTEST" [see Table 18.9]
  SFString [in,out] textureCompression "FASTEST" [see Table 18.10]
  SFFloat [in,out] texturePriority 0 [0,1]
  SFBool [ ] generateMipMaps FALSE
}
```

TextureProperties allows fine control over a texture's application.

This node can be used to set the texture properties for a node with a *textureProperties* field. A texture with a TextureProperties node will ignore the *repeatS* and *repeatT* fields on the texture.

The *anisotropicDegree* field describes the minimum degree of anisotropy to account for in texture filtering. A value of 1 implies no anisotropic filtering. Values above the system's maximum supported value will be clamped to the maximum allowed. Browsers are allowed to use higher values as deemed appropriate.

The *borderColor* field describes the color to use for border pixels.

The *borderWidth* field describes the number of pixels to use for a texture border.

The *boundaryModeS* field describes the way S texture coordinate boundaries are handled, as depicted in [Table 18.7](#).

The *boundaryModeT* field describes the way T texture coordinate boundaries are handled, as depicted in [Table 18.7](#).

The *boundaryModeR* field describes the way R texture coordinate boundaries are handled, as depicted in [Table 18.7](#). This field only applies to three dimensional textures and shall be ignored by other texture types.

The *magnificationFilter* field describes the way textures are filtered when the image is smaller than the screen space representation. Valid values are depicted in [Table 18.8](#).

The *minificationFilter* field describes the way textures are filtered when the image is larger than the screen space representation. Valid values are depicted in [Table 18.9](#). Modes with MIPMAP in the name require mipmaps. If mipmaps are not provided, the mode shall pick the corresponding non-mipmapped mode (e.g., `AVG_PIXEL_NEAREST_MIPMAP` becomes `AVG_PIXEL`).

The *texturePriority* field describes the texture residence priority for allocating texture memory. Zero indicates the lowest priority and 1 indicates the highest priority. Values are clamped to the range [0,1].

The *textureCompression* field specifies the preferred image compression method to be used during rendering. Valid values are in [Table 18.10](#).

The *generateMipMaps* field describes whether mipmaps should be generated for the texture. Mipmaps are required for filtering modes with MIPMAP in their value.

**Table 18.7 — Texture boundary modes**

Mode	Description
"CLAMP"	Clamp texture coordinates to the range [0,1]
"CLAMP_TO_EDGE"	Clamp texture coordinates such that a border texel is never sampled. Coordinates are clamped to the range $[1/(2N), 1 - 1/(2N)]$ , where N is the size of the texture in the direction of clamping.
"CLAMP_TO_BOUNDARY"	Clamp texture coordinates such that texture samples are border texels for fragments whose corresponding texture coordinate is sufficiently outside the range [0,1]. Texture coordinates are clamped to the range $[-1/(2N), 1 + 1/(2N)]$ .
	Texture coordinates are mirrored and then clamped as in

"MIRRORED_REPEAT"	CLAMP_TO_EDGE
"REPEAT"	Repeat a texture across the fragment. Ignore the integer part of the texture coordinates, using only the fractional part.

**Table 18.8 — Texture magnification modes**

Mode	Description
"AVG_PIXEL"	Select the weighted average of the four texture elements that are closest to the center of the pixel being textured.
"DEFAULT"	Select the browser-specified default magnification mode.
"FASTEST"	Select the fastest method available.
"NEAREST_PIXEL"	Select the <b>pixel texture element</b> that is nearest to the center of the pixel being textured.
"NICEST"	Select the highest quality method available.

**Table 18.9 — Texture minification modes**

Mode	Description
"AVG_PIXEL"	Select the weighted average of the four texture elements that are closest to the center of the pixel being textured.
"AVG_PIXEL_AVG_MIPMAP"	Performs tri-linear filtering. Choose the two mipmaps that most closely match the size of the pixel being textured and use the weighted average of the four texture elements that are closest to the center of the pixel to produce a texture value from each mipmap. The final texture value is a weighted average of those two values.
"AVG_PIXEL_NEAREST_MIPMAP"	Choose the mipmap that most closely matches the size of the pixel being textured and use the weighted average of the four texture elements that are closest to the center of the pixel to produce a texture value.
"DEFAULT"	Select the browser-specified default minification mode.
"FASTEST"	Select the fastest method available. Mipmaps shall be used, if available.
"NEAREST_PIXEL"	Select the <b>pixel texture element</b> that is nearest to the center of the pixel being textured.
"NEAREST_PIXEL_AVG_MIPMAP"	Choose the two mipmaps that most closely match the size of the pixel being textured and use the texture element nearest to the center of the pixel to produce a texture value from each mipmap. The final texture value is a weighted average of those two values.
	Choose the mipmap that most closely matches the size of



"NEAREST_PIXEL_NEAREST_MIPMAP"	the pixel being textured and use the texture element nearest to the center of the pixel) to produce a texture value.
"NICEST"	Select the highest quality method available. Mipmaps shall be used, if available.

Table 18.10 — Texture compression modes

Mode	Description
"DEFAULT"	Select the browser-specified default compression mode.
"FASTEST"	Select the fastest compression mode available.
"HIGH"	Select the compression mode with the greatest amount of compression.
"LOW"	Select the compression mode with the least amount of compression.
"MEDIUM"	Select a compression mode with a moderate amount of compression.
"NICEST"	Select the compression mode that produces the <b>nicesthighest quality</b> effect.

## 18.4.10 TextureTransform

```
TextureTransform : X3DSingleTextureTransformNode {
TextureTransform : X3DTextureTransformNode {
  SFVec2f [in,out] center 0 0 (-∞,∞)
  SFString [in,out] mapping ""
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFFloat [in,out] rotation 0 (-∞,∞)
  SFVec2f [in,out] scale 1 1 (-∞,∞)
  SFVec2f [in,out] translation 0 0 (-∞,∞)
}
```

The `TextureTransform` node defines a 2D transformation that is applied to texture coordinates (see [TextureCoordinate](#)). This node affects the way textures coordinates are applied to the geometric surface. The transformation consists of (in order):

- a translation;
- a rotation about the centre point;
- a non-uniform scale about the centre point.

These parameters support changes to the size, orientation, and position of textures on shapes. Note that these operations appear reversed when viewed on the surface of geometry. For example, a *scale* value of (2 2) will scale the texture coordinates and have the net effect of shrinking the texture size by a factor of 2 (texture coordinates are twice as large and thus cause the texture to repeat). A translation of (0.5 0.0) translates the texture coordinates +.5 units along the S-axis and has the net effect of translating the texture  $-0.5$  along the S-axis on the geometry's surface. A rotation of  $\pi/2$  of the texture coordinates results in a  $-\pi/2$  rotation of the texture on the geometry.

The *center* field specifies a translation offset in texture coordinate space about which the *rotation* and *scale* fields are applied. The *scale* field specifies a scaling factor in S and T of the texture coordinates about the *center* point. *scale* values shall be in the range  $(-\infty, \infty)$ . The *rotation* field specifies a rotation in angle base units of the texture coordinates about the *center* point after the scale has been applied. A positive rotation value makes the texture



coordinates rotate counterclockwise about the centre, thereby rotating the appearance of the texture itself clockwise. The *translation* field specifies a translation of the texture coordinates.

In matrix transformation notation, where  $T_c$  is the untransformed texture coordinate,  $T_c'$  is the transformed texture coordinate,  $C$  (*center*),  $T$  (*translation*),  $R$  (*rotation*), and  $S$  (*scale*) are the intermediate transformation matrices,

$$T_c' = -C \times S \times R \times C \times T \times T_c$$

NOTE This transformation order is the reverse of the [Transform](#) node transformation order since the texture coordinates, not the texture, are being transformed (*i.e.*, the texture coordinate system).

## 18.5 Support levels

The Texturing component provides three levels of support as specified in [Table 18.7](#).

**Table 18.11 — Texturing component support levels**

Level	Prerequisites	Nodes/Features	Support
<b>1</b>	Core 1 Grouping 1 Shape 1 Rendering 1		
		<i>X3DSingleTextureCoordinateNode</i> (abstract)	n/a
		<i>X3DSingleTextureNode</i> (abstract)	n/a
		<i>X3DSingleTextureTransformNode</i> (abstract)	n/a
		<i>X3DTextureCoordinateNode</i> (abstract)	n/a
		<i>X3DTextureNode</i> (abstract)	n/a
		<i>X3DTexture2DNode</i> (abstract)	n/a
		<i>X3DTextureTransformNode</i> (abstract)	n/a
		ImageTexture	All fields fully supported.
		PixelTexture	All fields fully supported.
		TextureCoordinate	All fields fully supported.
		TextureTransform	All field fully supported.

<b>2</b>	Core 1 Grouping 1 Shape 1 Rendering 1		
		All Level 1 Texturing nodes	All fields as supported in Level 1.
		MultiTextureCoordinate	All fields fully supported.
		MultiTextureTransform	All fields fully supported.
		TextureCoordinateGenerator	All fields fully supported.
		TextureProperties	All fields fully supported.
<b>3</b>	Core 1 Grouping 1 Shape 1 Rendering 1		
		All Level 2 Texturing nodes	All fields as supported in Level 2.
		MultiTexture	All fields fully supported.
<b>4</b>	Core 1 Grouping 1 Shape 1 Rendering 1		
		All Level 3 Texturing nodes	All fields as supported in Level 3.
		MovieTexture	All fields fully supported.



## Extensible 3D (X3D) Part 1: Architecture and base components

### 39 Followers component



#### 39.1 Introduction

##### 39.1.1 Name

The name of this component is "Followers". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 39.1.2 Overview

This clause describes the Followers component of this part of ISO/IEC 19775. This includes how Followers are specified and how they behave. [Table 39.1](#) provides links to the major topics in this clause.

**Table 39.1 — Topics**

- [39.1 Introduction](#)
  - [39.1.1 Name](#)
  - [39.1.2 Overview](#)
- [39.2 Concepts](#)
- [39.3 Abstract types](#)
  - [39.3.1 X3DChaserNode](#)
  - [39.3.2 X3DDamperNode](#)
  - [39.3.3 X3DFollowerNode](#)
- [39.4 Node reference](#)
  - [39.4.1 ColorChaser](#)
  - [39.4.2 ColorDamper](#)
  - [39.4.3 CoordinateChaser](#)
  - [39.4.4 CoordinateDamper](#)
  - [39.4.5 OrientationChaser](#)
  - [39.4.6 OrientationDamper](#)
  - [39.4.7 PositionChaser](#)
  - [39.4.8 PositionChaser2D](#)

- 39.4.9 PositionDamper
  - [39.4.10 PositionDamper2D](#)
  - [39.4.11 ScalarChaser](#)
  - [39.4.12 ScalarDamper](#)
  - [39.4.13 TexCoordChaser2D](#)
  - [39.4.14 TexCoordDamper2D](#)
- [39.5 Support levels](#)
- [Figure 39.1 — Calculating the output of an X3DFollowerNode](#)
- [Figure 39.2 — Concept of an X3DDamperNode](#)
- [Figure 39.3 — Mode of operation of an X3DFollowerNode](#)
- [Table 39.1 — Topics](#)
- [Table 39.2 — Followers component support levels](#)

## 39.2 Concepts

The group of *Follower* nodes supports the creation of transitions of parameters at runtime (dynamically) by receiving a destination value upon which they create an animation that transitions their output value from its current value towards the newly set destination value.

In case a transition triggered by reception of a previous destination value is not yet finished while the new destination is received, both the new and old transition are merged, so that a smooth animation is created where the previous movement degrades and gradually becomes a movement towards the new destination which is then eventually reached.

*Follower* nodes accomplish the transition by implementing **finite impulse response** (FIR) filters and **infinite impulse response** (IIR) filters from the field of system theory. Due to this filter distinction, the *Follower* nodes are divided into *Chaser* nodes (FIR) and *Damper* nodes (IIR).

Like *TimeSensor* nodes, *Follower* nodes often send output events at times when they have not received input events. Their behaviour is completely determined by the events they receive from the scene graph itself at earlier times.

*Follower* nodes are not affected by their position in the transformation hierarchy nor are they affected by the state of containing [Switch](#) nodes, [LOD](#) nodes and other nodes that affect the visibility of their children.

## 39.3 Abstract types

### 39.3.1 X3DChaserNode

```
X3DChaserNode : X3DFollowerNode {
  [SIM]F<type> [in]  set_destination
  [SIM]F<type> [in]  set_value
  SFNode      [in,out] metadata      NULL [X3DMetadataObject]
  SFBool      [out]  isActive
  [SIM]F<type> [out]  value_changed
  SFTime      []    duration      1 [0,∞)
```

```

    [S|M]F<type> []    initialDestination
    [S|M]F<type> []    initialValue
}

```

The *X3DChaserNode* abstract node type calculates the output on *value\_changed* as a finite impulse response (FIR) based on the events received on *set\_destination* in the following manner.

Each time an event is received on *set\_destination*, a transition  $A_n$  from the previously received destination to the new destination is created according to Equation (1). The data types of all variables are floating point numbers, or integers in the case of indices, except for  $d_n$ ,  $d_{n-1}$ ,  $A_n(t)$  and  $O(t)$ . These variables have the data type of the node (e.g., *SFVec3f* or *SFColor*).

$$A_n(t) = \begin{cases} d_n - d_{n-1} & \forall t \geq T_n \\ (d_n - d_{n-1})R\left(\frac{t - T_n}{D}\right) & \forall T_n < t < T_n + D \\ 0 & \forall t \geq T_n + D \end{cases} \quad (1)$$

where:

$T_n$  is the point in time where the event has been received

$D$  is the value of the *duration* field

$d_n$  is the new destination value received with the event

$d_{n-1}$  is the value that was the destination before the event

$R(x)$  is the core function of the filter:

$$R(x) = \frac{1 - \cos \pi x}{2} \quad (2)$$

All the transitions created for every event on *set\_destination* are added together to form the output on *value\_changed*.

$$O(t) = d_{k-1} + \sum_{n=k}^l A_n(t) \quad (3)$$

where  $l$  is the number of events received so far on *set\_destination*. If  $k$  is set to 0,  $d_{-1}$  is the value of the *initialValue* field and  $d_0$  is the value of *initialDestination*. This way the initial transition determined by these two fields is produced.

Theoretically the start index  $k$  **could might** be always set to zero meaning that all *set\_destination* events since initialization are to be stored. However,  $k$  can be increased without changing the result  $O(t)$  as long as the time stamp  $T_{k-1}$  is more than  $D$  seconds before the current time stamp. This is due to the facts:

- after a period of  $D$  seconds (the *duration* field), the transitions  $A_n(t)$  are constantly  $d_n - d_{n-1}$ ; and
- $d_{k-1}$  is the sum of all differences  $d_n - d_{n-1}$  so far.

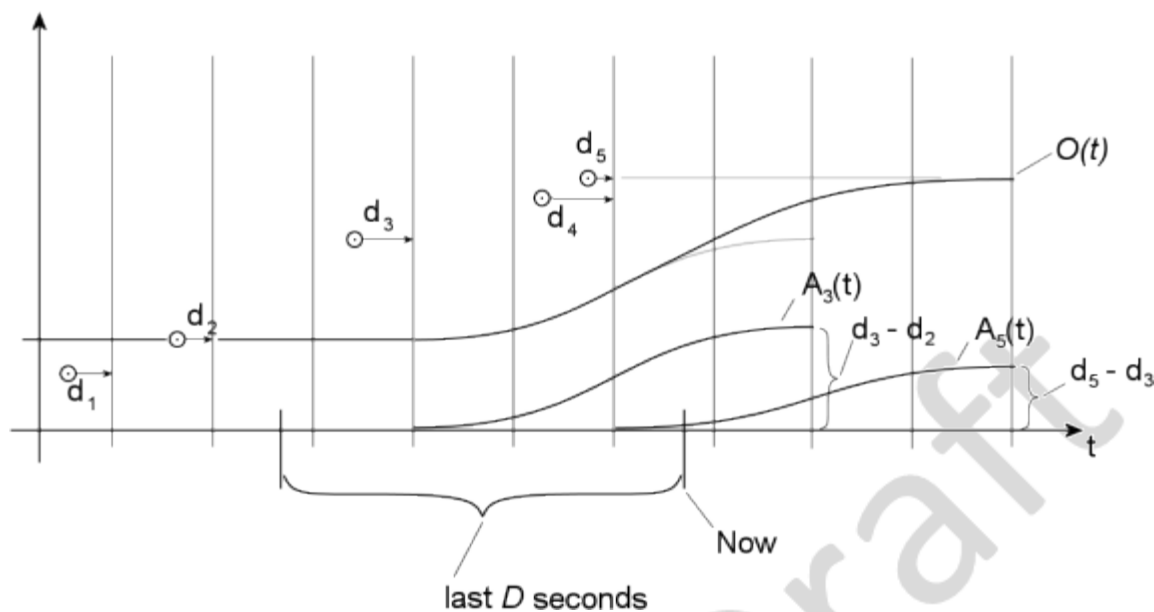
This way the *X3DChaserNode* implementation remembers the values and time stamps of all *set\_destination* events received in the last period of *duration* seconds plus the

value received latest before that period. For calculating the current value of *value\_changed*, the *X3DChaserNode* uses that latest received value as a starting point ( $d_{k-1}$ ) and adds to it all transitions  $A_n(t)$  generated by the stored events.

A more optimal implementation **could** **might** divide the time-line into equidistant time-slots and store only the latest *set\_destination* event received for each time-slot. This way a fixed length array **could** **might** be used for describing the input during the period of the last *duration* seconds. This however can create little jumps in the animation created at *value\_changed* since a *set\_destination* event may cause the beginning of a transition being produced and may then be replaced by a later event received in the same time-slot. To avoid this, events are associated with the end of the time-slot rather than with the time-stamp when they are received.

Thus, the output reaches the value received at *set\_destination* up to the length of a time-slot later than is dictated by the *duration* field. To compensate, an implementation shall subtract the length of a time-slot from *duration* and use the result for *D*.

It is suggested that the implementation uses (about) 10 time-slots per duration *duration* as depicted in [Figure 39.1](#).



**Figure 39.1 — Calculating the output of an *X3DFollowerNode***

The above diagram illustrates how an implementation calculates the output at an arbitrary point in time. Figure 39.1 depicts only four time-slots per duration  $D$ . The period goes from the current time-stamp *Now* back by  $D$  seconds, not necessarily matching the grid of the time slots. The events  $d_1$  and  $d_2$  have happened before this period and are therefore summarized by the value of  $d_2$ . The event  $d_3$  however falls into the period of  $D$  seconds. It is moved towards the end of the time-slot it falls into and generates the transition  $A_3(t)$  with the amplitude  $d_3 - d_2$ . The event  $d_4$  gets ignored because it is followed by  $d_5$  in the same time-slot. Therefore only  $d_5$  generates a transition, which is  $A_5(t)$ . The amplitude of  $A_5(t)$  is  $d_5 - d_3$  because  $d_4$  got ignored. The output  $O(t)$  is thus calculated as specified in Equation (4):

$$O(t) = d_2 + A_3(t) + A_5(t) \quad (4)$$

When the current time-stamp has advanced until after the end of curve  $A_3(t)$ , which is when the time-slot containing event  $d_3$  is no longer part of the last  $D$  seconds, the start value for the addition  $d_2$  is replaced with  $d_3$  and the curve  $A_3(t)$  is removed from the addition, so that  $O(t) = d_3 + A_5(t)$ .

The above diagram uses four time-slots per duration  $D$ . With the above recommendations of making  $D$  one time-slot shorter than the *duration* field specifies, this means that a time-slot is a fifth of what is specified by *duration*.

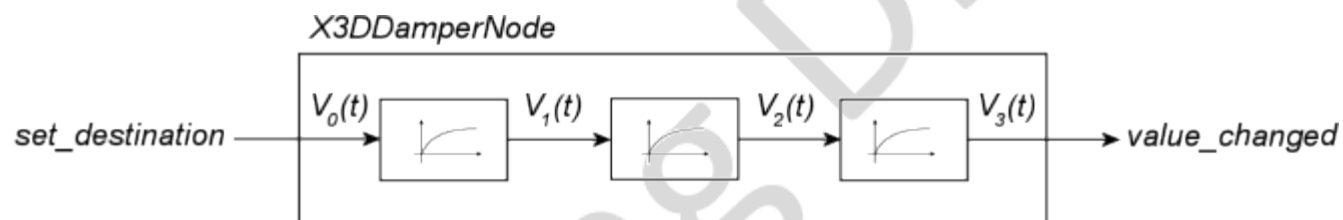
### 39.3.2 X3DDamperNode

```

X3DDamperNode : X3DFollowerNode {
  [S|M]F<type> [in]  set_destination
  [S|M]F<type> [in]  set_value
  SFNode      [in,out] metadata      NULL [X3DMetadataObject]
  SFTime      [in,out] tau           0.3 [0,∞)
  SFFloat     [in,out] tolerance     -1 -1 or [0,∞)
  SFBool      [out]  isActive
  [S|M]F<type> [out]  value_changed
  [S|M]F<type> []    initialDestination
  [S|M]F<type> []    initialValue
  SFInt32     []    order           3 [0..5]
}
    
```

The *X3DDamperNode* abstract node type creates an IIR response that approaches the destination value according to the shape of the *e*-function only asymptotically but very quickly.

An *X3DDamperNode* node is parameterized by the *tau*, *order* and *tolerance* fields. Internally, it consists of a set of linear first-order filters each of which processes the output of the previous filter as shown in [Figure 39.2](#). The input of the first filter is fed by the values received on *set\_destination* and the output of the last filter goes to the *value\_changed* field.



**Figure 39.2 — Concept of an *X3DDamperNode***

The calculations of the output for the current time-stamp  $T_n$  for each filter are based on the output of that filter from the previous time-stamp  $T_{n-1}$  and the current input using the Equation (5):

$$o_n = d_n + (o_{n-1} - d_n) e^{-\frac{\Delta T}{\tau}} \quad (5)$$

$$o_n = V_k(T_n)$$

$$o_{n-1} = V_k(T_{n-1})$$

$$d_n = V_{k-1}(T_n)$$

$$\Delta T = T_n - T_{n-1}$$

The field *order* specifies the number of such internal filters. Specifying zero for *order*

means that no filter is used. In this case the events received on *set\_destination* are forwarded directly to *output\_changed*. The larger the value for *order*, the smoother the output on *value\_changed* will be, but the more delay will be introduced. Since values larger than five do not introduce any more smoothing, the range for *order* is limited to a maximum of five.

The field *tau* specifies the time-constant of the internal filters and thus the speed that the output of an *X3DDamperNode* responds to the input. Its value is assigned to the variable  $\tau$  in the above equation. A value of zero for *tau* means immediate response and the events received on *set\_destination* are forwarded directly to *output\_changed*. The field *tau* specifies how long it takes the output of an internal filter to reach the value of its input by 63% ( $1 - 1/e$ ). The remainder after that period is reduced by 63% during another period of *tau* seconds provided that the input of the filter does not change. This behavior can be exposed if *order* is set to one.

Since the output of an *X3DDamperNode* approaches the input value only asymptotically, there must be a means to determine when the destination value can be assumed to be reached and the node can stop emitting values and set *isActive* to `FALSE`. This is governed by the *tolerance* field. If *tolerance* is set to its default value -1, the browser implementation is allowed to find a good way for detecting the end of a transition. Browsers that do not have an elaborate algorithm can just use .001 as the tolerance value instead. If a value larger than zero is specified for *tolerance*, the browser shall calculate the difference between output and input for each internal filter being used and stop the animation only when all filters fall below that limit or are equal to it. If zero is specified for *tolerance*, a transition should be stopped only if input and output match exactly for all internal filters. This can happen if *set\_value* receives an event.

An implementation shall test for end of transition before it calculates the new output value. Then, the implementation shall either assign the *destination value* to the *output value*, if the difference falls below the tolerance limit, or calculate an updated output value.

### 39.3.3 X3DFollowerNode

```
X3DFollowerNode : X3DChildNode {
  [S|M]F<type> [in]  set_destination
  [S|M]F<type> [in]  set_value
  SFNode      [in,out] metadata      NULL [X3DMetadataObject]
  SFBool      [out]  isActive
  [S|M]F<type> [out]  value_changed
  [S|M]F<type> []    initialDestination
  [S|M]F<type> []    initialValue
}
```

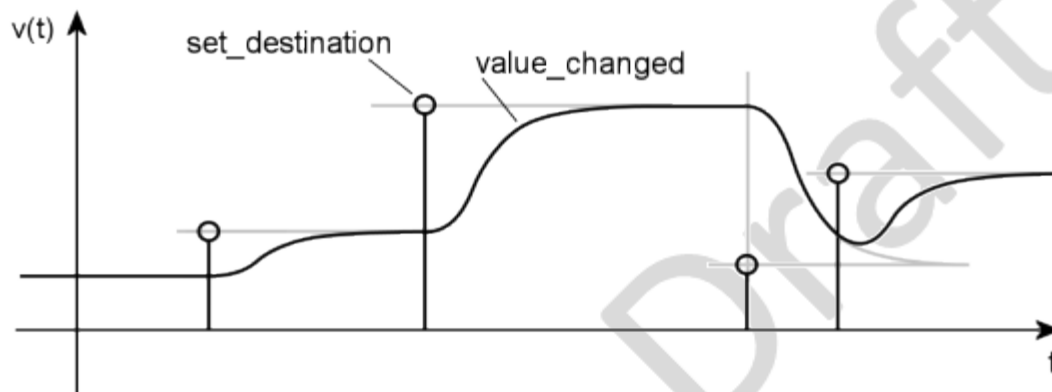
The abstract node *X3DFollowerNode* forms the basis for all nodes specified in this clause. The data type place holder *[S|M]F<type>* evaluates to the same data type for all fields of a specialization of the abstract node class *X3DFollowerNode*.

An *X3DFollowerNode* maintains an internal state that consists of a *current value* and a *destination value*. Both values are of the same data type into which the term *[S|M]F<type>* evaluates for a given specialization. It is the '*data type of the node*'. In certain cases of usage, the terms *input* and *output* fit better for *destination value* and *current value*, respectively.

Whenever the *current value* differs from the *destination value*, the current value



gradually changes until it reaches the *destination value* producing a smooth transition. It generally moves towards the *destination value* but, if a transition triggered by a previous *destination value* is still in progress, it may take a short while until the movement becomes a movement towards the new destination value. [Figure 39.3](#) depicts this action.



**Figure 39.3 — Mode of operation of an *X3DFollowerNode***

The *value\_changed* outputOnly field outputs the current value of the internal state.

The *set\_destination* inputOnly field receives new destination values, resulting in the *value\_changed* field sending output values in most cases.

The initializeOnly fields, *initialDestination* and *initialValue*, initialize the internal state of the *X3DFollowerNode*. The *current value* receives the value of *initialValue* and the *destination value* receives the value of *initialDestination*. If both fields have the same values, the *X3DFollowerNode* sends that value through the *value\_changed* field in a single event upon initialization. If both fields have different values, the *X3DFollowerNode* creates an animation from the value of *initialValue* towards the value of *initialDestination*. The shape of that transition is the same as if the *current value* internal state had always been at the value of *initialValue* and the node had just received the *destination value*.

With the *set\_value* inputOnly field, one can immediately force the *current value* towards a certain value. When the *X3DFollowerNode* receives a value on *set\_value*, any current transition is stopped and the *current value* assumes that value. The *value\_changed* field outputs that value and then moves towards the value currently set for the *destination value*. This animation has the same shape as if the *current value* had already been at the newly received value for a long time and the node had just received an event on *set\_destination* carrying the value of the currently set *destination value*.

One can achieve various results by sending certain values to *set\_value*, *set\_destination* or both at the same time:

- ***set\_destination* and *set\_value* receive different values:**  
A transition is created that goes from the value of *set\_value* towards the value of *set\_destination*. The transition is independent of the previous history of the node. With most parameter settings, the transition starts with zero speed and then accelerates towards the destination.
- ***set\_destination* and *set\_value* receive the same value:**

*Output\_changed* assumes the specified value immediately and stays there. No transition is created.

- **set\_value receives the value value\_changed currently has:**  
*Value\_changed* stops moving immediately and begins a new transition towards the currently set *destination value*. With most parameter settings, the result is that *value\_changed* stops moving and then accelerates towards the *destination value* to which it was already targeted.
- **set\_value receives the value currently set as destination:**  
The *output\_changed* value jumps to the *destination value* immediately.
- **set\_destination and set\_value both receive the current value of value\_changed:**  
The transition produced comes to an immediate halt at its current value.

The *isActive* outputOnly field identifies the beginning and end of a transition. It sends `TRUE` before *set\_value* begins animating and it sends `FALSE` after *set\_value* has reached the *destination* or has been stopped by another means. When *set\_value* receives an event while *isActive* is `TRUE`, *isActive* sends `FALSE` after *value\_changed* has output the received value. If *isActive* is `FALSE` at that moment, *isActive* generates no event.

## 39.4 Node reference

### 39.4.1 ColorChaser

```
ColorChaser: X3DChaserNode {
  SFColor [in]  set_destination
  SFColor [in]  set_value
  SFNode [in,out] metadata      NULL    [X3DMetadataObject]
  SFBool [out]  isActive
  SFColor [out] value_changed
  SFTime []    duration          1      [0,∞)
  SFColor []   initialDestination 0.8 0.8 0.8 [0,1]
  SFColor []   initialValue       0.8 0.8 0.8 [0,1]
}
```

The *ColorChaser* animates transitions for single colour values. Whenever the *set\_destination* field receives a floating point number, the *value\_changed* creates a transition from its current value to the newly set number. It creates a smooth transition that ends *duration* seconds after the last number has been received.

When *set\_value* receives a colour value, any transition currently in process is stopped and *value\_changed* sends this value immediately, creating a jump. The field *initialValue* can be used to set the initial value of *value\_changed*. The field *initialDestination* should be set to the same value unless a transition to a certain value is to be created right after the scene is loaded or right after the *ColorChaser* node is created dynamically.

### 39.4.2 ColorDamper

```
ColorDamper : X3DDamperNode {
  SFColor [in]  set_destination
  SFColor [in]  set_value
  SFNode [in,out] metadata      NULL    [X3DMetadataObject]
  SFTime [in,out] tau           0.3      [0,∞)
  SFFloat [in,out] tolerance    -1      -1 or [0,∞)
  SFBool [out]  isActive
  SFColor [out] value_changed
  SFColor []   initialDestination 0.8 0.8 0.8 [0,1]
  SFColor []   initialValue       0.8 0.8 0.8 [0,1]
  SFInt32 []   order             3        [0..5]
}
```

The ColorDamper animates colour values. Whenever the *set\_destination* field receives a colour, the ColorDamper node creates a transition from the current colour to the newly set colour. The transition created approaches the newly set position asymptotically during a time period of approximately three to four times the value of the field *tau* depending on the desired accuracy and the value of *order*. The *order* field specifies the smoothness of the transition.

When *set\_value* receives a colour, any transition currently in process is stopped and *value\_changed* sends this value immediately, creating a jump to the new colour. The field *initialValue* can be used to set the initial colour. The field *initialDestination* should be set to the same value unless a transition to a certain colour is to be created right after the scene is loaded or right after the ColorDamper node is created dynamically.

### 39.4.3 CoordinateChaser

```
CoordinateChaser: X3DChaserNode {
  MFVec3f [in]  set_destination
  MFVec3f [in]  set_value
  SFNode [in,out] metadata      NULL [X3DMetadataObject]
  SFBool [out]  isActive
  MFVec3f [out] value_changed
  SFTime []    duration          1 [0,∞)
  MFVec3f []   initialDestination 0 0 0
  MFVec3f []   initialValue       0 0 0
}
```

The CoordinateChaser animates transitions for array of 3D vectors (e.g., the coordinates of a mesh). Whenever the *set\_destination* field receives an array of 3D vectors, the *value\_changed* creates a transition from its current value to the newly set number. It creates a smooth transition that ends *duration* seconds after the last number has been received.

When *set\_value* receives an array of 3D vectors, any transition currently in process is stopped and *value\_changed* sends this value immediately, creating a jump. The field *initialValue* can be used to set the initial value of *value\_changed*. The field *initialDestination* should be set to the same value unless a transition to a certain value is to be created right after the scene is loaded or right after the CoordinateChaser node is created dynamically.

### 39.4.4 CoordinateDamper

```
CoordinateDamper : X3DDamperNode {
  MFVec3f [in]  set_destination
  MFVec3f [in]  set_value
  SFNode [in,out] metadata      NULL [X3DMetadataObject]
  SFTime [in,out] tau           0.3 [0,∞)
  SFFloat [in,out] tolerance    -1 -1 or [0,∞)
  SFBool [out]  isActive
  MFVec3f [out] value_changed
  MFVec3f []   initialDestination 0 0 0
  MFVec3f []   initialValue       0 0 0
  SFInt32 []   order             3 [0..5]
}
```

The CoordinateDamper animates transitions for an array of 3D vectors (e.g., the coordinates of a mesh). Whenever the *set\_destination* field receives an array of 3D vectors, *value\_changed* begins sending an array of the same length, where each element moves from its current value towards the value at the same position in the array received. Each element approaches its destination value asymptotically during a time period of approximately three to four times the value of the field *tau* depending on the desired accuracy and the value of *order*. The *order* field specifies the smoothness of the transition. The transition ends when all elements have reached their destination.

When *set\_value* receives an event, any transition currently in process is stopped and *value\_changed* sends this array immediately, creating a jump. The field *initialValue* can be used to set the initial value of *value\_changed*. The field *initialDestination* should be set to the same value unless a transition to a certain 3D vector value is to be created right after the scene is loaded or right after the CoordinateDamper node is created dynamically.

The MFVec3f arrays that are sent to the *set\_destination* or *set\_value* field shall have the same length (number of elements). The length of the arrays shall not change over time. Values assigned to *initialDestination* or *initialValue* shall either be an empty array or an array with the same number of elements as is sent to the *set\_destination* or *set\_value* fields. In any other case, the behavior is not defined.

### 39.4.5 OrientationChaser

```
OrientationChaser : X3DChaserNode {
  SFRotation [in]  set_destination
  SFRotation [in]  set_value
  SFNode [in,out] metadata      NULL [X3DMetadataObject]
  SFBool [out]    isActive
  SFRotation [out] value_changed
  SFTime []      duration      1 [0,∞)
  SFRotation []  initialDestination 0 1 0 0
  SFRotation []  initialValue      0 1 0 0
}
```

The OrientationChaser animates transitions for orientations. If the *value\_changed* field is routed to a *rotation* field of a *Transform* node that contains an object, whenever the *set\_destination* field receives an orientation, the OrientationChaser node rotates the object from its current orientation to the newly set orientation. It creates a smooth transition that ends *duration* seconds after the last orientation has been received.

When *set\_value* receives an orientation, any transition currently in process is stopped and the object jumps directly to the given orientation. The field *initialValue* can be used to set the initial orientation of the object. The field *initialDestination* should be set to the same value unless a transition to a certain orientation is to be created right after the scene is loaded or right after the OrientationChaser node is created dynamically.

The OrientationChaser node can be implemented by combining Equations (1), (2), and (3) to form Equation (6):

$$O(t) = d_{k-1} + \sum_{n=k}^l (d_n - d_{n-1})R(\dots) \quad (6)$$

This leads to the following loop denoted in pseudo code:

```
var Result = dk-1;
for (var n from k to l) {
  var Delta = dn - dn-1;
  Result = Result + Delta × R(...);
}
O(t) = Result;
```

Since  $d_{k-1}$ ,  $d_n$ ,  $d_{n-1}$  and thus `Result` contain rotation values (SFRotation), the above code must be converted to use operations available for rotations. This can be achieved using

the *slerp* operation. For the following, let  $\text{slerp}(A, B, t)$  be a function that calculates the linear spherical interpolation from  $A$  to  $B$  by the amount  $t$ . Let also  $\text{core}(\cdot)$  be a function that calculates  $R(\dots)$  and let  $\text{Buffer}$  be an array so that  $\text{Buffer}[i]$  evaluates to  $d_i$ . Then, the above loop can be implemented as:

```
var Result = Buffer[k-1];
for(var n from k to l) {
  var Delta = Buffer[n-1].inverse().multiply(Buffer[n]);
  Result = slerp(Result, Result.multiply(Delta), Core(...));
}
O(t) = Result;
```

### 39.4.6 OrientationDamper

```
OrientationDamper : X3DDamperNode {
  SFRotation [in]  set_destination
  SFRotation [in]  set_value
  SFNode [in,out] metadata      NULL [X3DMetadataObject]
  SFTime [in,out] tau           0.3 [0,∞)
  SFFloat [in,out] tolerance    -1 -1 or [0..∞]
  SFBool [out] isActive
  SFRotation [out] value_changed
  SFRotation [] initialDestination 0 1 0 0
  SFRotation [] initialValue 0 1 0 0
  SFInt32 [] order 3 [0..5]
}
```

The OrientationDamper animates transitions of orientations. If the *value\_changed* field is routed to an *orientation* field of a [Transform](#) node that contains an object, then, whenever the *set\_destination* field receives an orientation, the OrientationDamper node rotates the object from its current orientation to the newly set orientation. It creates a transition that approaches the newly set orientation asymptotically during a time period of approximately three to four times the value of the field *tau* depending on the desired accuracy and the value of *order*. Through this asymptotic approach of the destination orientation, a very smooth transition is created. The *order* field specifies the smoothness of the transition.

When *set\_value* receives an orientation, any transition currently in process is stopped and the object jumps directly to the given orientation. The field *initialValue* can be used to set the initial orientation of the object. The field *initialDestination* should be set to the same value unless a transition to a certain orientation is to be created right after the scene is loaded or right after the OrientationDamper node is created dynamically.

The OrientationDamper node is implemented by calculating Equation (5) for each internal filter. For SFRotation values, the equation is equivalent to the following term:

$$\text{output} = \text{input.slerp}(\text{output}, \alpha);$$

where:

output:  $o_n$  or  $o_{n-1}$ , respectively

input:  $d_n$

alpha:  $e^{-\Delta T/\tau}$

### 39.4.7 PositionChaser

```
PositionChaser : X3DChaserNode {
```

```

SFVec3f [in]  set_destination
SFVec3f [in]  set_value
SFNode [in,out] metadata      NULL [X3DMetadataObject]
SFBool [out]  isActive
SFVec3f [out] value_changed
SFTime [] duration            1 [0,∞)
SFVec3f [] initialDestination 0 0 0
SFVec3f [] initialValue       0 0 0
}

```

The PositionChaser animates transitions for 3D vectors. If the *value\_changed* field is routed to a *translation* field of a [Transform](#) node that contains an object, then, whenever the *set\_destination* field receives a 3D position, the PositionChaser node moves the object from its current position to the newly set position. It creates a smooth transition that ends *duration* seconds after the last position has been received.

When *set\_value* receives a position, any transition currently in process is stopped and the object jumps directly to the given position. The field *initialValue* can be used to set the initial position of the object. The field *initialDestination* should be set to the same value unless a transition to a certain position is to be created right after the scene is loaded or right after the PositionChaser node is created dynamically.

### 39.4.8 PositionChaser2D

```

PositionChaser2D : X3DChaserNode {
SFVec2f [in]  set_destination
SFVec2f [in]  set_value
SFNode [in,out] metadata      NULL [X3DMetadataObject]
SFBool [out]  isActive
SFVec2f [out] value_changed
SFTime [] duration            1 [0,∞)
SFVec2f [] initialDestination 0 0
SFVec2f [] initialValue       0 0
}

```

The PositionChaser2D animates transitions for 2D vectors. Whenever the *set\_destination* field receives a 2D vector the *value\_changed* creates a transition from its current 2D vector value to the newly set value. It creates a smooth transition that ends *duration* seconds after the last 2D vector has been received.

When *set\_value* receives a 2D vector, any transition currently in process is stopped and *value\_changed* sends this value immediately. The field *initialValue* can be used to set the initial initial value of *value\_changed*. The field *initialDestination* should be set to the same value unless a transition to a certain 2D vector value is to be created right after the scene is loaded or right after the PositionChaser2D node is created dynamically.

### 39.4.9 PositionDamper

```

PositionDamper : X3DDamperNode {
SFVec3f [in]  set_destination
SFVec3f [in]  set_value
SFNode [in,out] metadata      NULL [X3DMetadataObject]
SFTime [in,out] tau           0.3 [0,∞)
SFFloat [in,out] tolerance    -1 -1 or [0,∞)
SFBool [out]  isActive
SFVec3f [out] value_changed
SFVec3f [] initialDestination 0 0 0
SFVec3f [] initialValue       0 0 0
SFInt32 [] order              3 [0..5]
}

```

The PositionDamper animates transitions for 3D vectors. If the *value\_changed* field is routed to a *translation* field of a [Transform](#) node that contains an object, then, whenever the *set\_destination* field receives a 3D position, the PositionDamper node moves the object from its current position to the newly set position. It creates a transition that approaches the newly set position asymptotically during a time period of

approximately three to four times the value of the field *tau* depending on the desired accuracy and the value of *order*. Through this asymptotic approach of the destination value, a smooth transition is created. The *order* field specifies the smoothness of the transition.

When *set\_value* receives a position, any transition currently in process is stopped and the object jumps directly to the given position. The field *initialValue* can be used to set the initial position of the object. The field *initialDestination* should be set to the same value unless a transition to a certain position is to be created right after the scene is loaded or right after the PositionDamper node is created dynamically.

### 39.4.10 PositionDamper2D

```
PositionDamper2D : X3DDamperNode {
  SFVec2f [in]  set_destination
  SFVec2f [in]  set_value
  SFNode [in,out] metadata      NULL [X3DMetadataObject]
  SFTime [in,out] tau          0.3 [0,∞)
  SFFloat [in,out] tolerance    -1 -1 or [0..∞]
  SFBool [out]  isActive
  SFVec2f [out] value_changed
  SFVec2f []   initialDestination 0 0
  SFVec2f []   initialValue        0 0
  SFInt32 []   order                3 [0..5]
}
```

The PositionDamper2D animates transitions for 2D vectors. Whenever the *set\_destination* field receives a 2D vector, the *value\_changed* creates a transition from its current 2D vector value to the newly set value. It creates a transition that approaches the newly set 2D vector asymptotically during a time period of approximately three to four times the value of the field *tau* depending on the desired accuracy and the value of *order*. The *order* field specifies the smoothness of the transition.

When *set\_value* receives a 2D vector, any transition currently in process is stopped and *value\_changed* sends this value immediately, creating a jump. The field *initialValue* can be used to set the initial initial value of *value\_changed*. The field *initialDestination* should be set to the same value unless a transition to a certain 2D vector value is to be created right after the scene is loaded or right after the PositionChaser2D node is created dynamically.

### 39.4.11 ScalarChaser

```
ScalarChaser : X3DChaserNode {
  SFFloat [in]  set_destination
  SFFloat [in]  set_value
  SFNode [in,out] metadata      NULL [X3DMetadataObject]
  SFBool [out]  isActive
  SFFloat [out] value_changed
  SFTime []    duration          1 [0,∞)
  SFFloat []   initialDestination 0
  SFFloat []   initialValue       0
}
```

The ScalarChaser animates transitions for single float values. Whenever the *set\_destination* field receives a floating point number, the *value\_changed* creates a transition from its current value to the newly set number. It creates a smooth transition that ends *duration* seconds after the last number has been received.

When *set\_value* receives a floating point number, any transition currently in process is stopped and *value\_changed* sends this value immediately, creating a jump. The field *initialValue* can be used to set the initial initial value of *value\_changed*. The field



*initialDestination* should be set to the same value unless a transition to a certain value is to be created right after the scene is loaded or right after the ScalarChaser node is created dynamically.

### 39.4.12 **ScalerDamper** **ScalarDamper**

```
ScalarDamper: X3DDamperNode {
  SFFloat [in]  set_destination
  SFFloat [in]  set_value
  SFNode [in,out] metadata      NULL [X3DMetadataObject]
  SFTime [in,out] tau           0.3 [0,∞)
  SFFloat [in,out] tolerance    -1 -1 or [0,∞)
  SFBool [out]  isActive
  SFFloat [out] value_changed
  SFFloat []    initialDestination 0
  SFFloat []    initialValue       0
  SFInt32 []    order              3 [0..5]
}
```

The ScalarDamper animates transitions for single float values. If the *value\_changed* field is routed to a *transparency* field of a Material node, then, whenever the *set\_destination* field receives a single float value, the ScalarDamper node creates a transition from its current value to the newly set value. It creates a transition that approaches the newly set value asymptotically during a time period of approximately three to four times the value of the field *tau* depending on the desired accuracy and the value of order. Through this asymptotic approach of the destination value, a smooth transition is created. The *order* field specifies the smoothness of the transition.

When *set\_value* receives a value, any transition currently in process is stopped and *value\_changed* sends this value immediately, creating a jump to the new value. The field *initialValue* can be used to set the initial value of the node. The field *initialDestination* should be set to the same value unless a transition to a certain value is to be created right after the scene is loaded or right after the ScalarDamper node is created dynamically.

### 39.4.13 **TexCoordChaser2D**

```
TexCoordChaser2D: X3DChaserNode {
  MFVec2f [in]  set_destination
  MFVec2f [in]  set_value
  SFNode [in,out] metadata      NULL [X3DMetadataObject]
  SFBool [out]  isActive
  MFVec2f [out] value_changed
  SFTime []    duration         1 [0,∞)
  MFVec2f []    initialDestination []
  MFVec2f []    initialValue    []
}
```

The TexCoordChaser2D animates transitions for an array of 2D vectors (e.g., the texture coordinates of a mesh). Whenever the *set\_destination* field receives an array of 2D vectors, the *value\_changed* creates a transition from its current value to the newly set number. It creates a smooth transition that ends *duration* seconds after the last number has been received.

When *set\_value* receives an array of 2D vectors, any transition currently in process is stopped and *value\_changed* sends this value immediately, creating a jump. The field *initialValue* can be used to set the initial value of *value\_changed*. The field *initialDestination* should be set to the same value unless a transition to a certain value is to be created right after the scene is loaded or right after the TexCoordChaser2D node is created dynamically.



## 39.4.14 TexCoordDamper2D

```

TexCoordDamper2D : X3DDamperNode {
  MFVec2f [in]  set_destination
  MFVec2f [in]  set_value
  SFNode [in,out] metadata      NULL [X3DMetadataObject]
  SFTime [in,out] tau           0.3 [0,∞)
  SFFloat [in,out] tolerance    -1 -1 or [0..∞]
  SFBool [out]  isActive
  MFVec2f [out] value_changed
  MFVec2f []   initialDestination []
  MFVec2f []   initialValue        []
  SFInt32 []   order               3 [0..5]
}

```

The `TexCoordDamper2D` node animates transitions for an array of 2D vectors (e.g., the texture coordinates of a mesh). Whenever the `set_destination` field receives an array of 2D vectors, `value_changed` begins sending an array of the same length, where each element moves from its current value towards the value at the same position in the array received. Each element approaches its destination value asymptotically during a time period of approximately three to four times the value of the field `tau` depending on the desired accuracy and the value of `order`. The `order` field specifies the smoothness of the transition. The transition ends when all elements have reached their destination.

When `set_value` receives an event, any transition currently in process is stopped and `value_changed` sends this array immediately, creating a jump. The field `initialValue` can be used to set the initial value of `value_changed`. The field `initialDestination` should be set to the same value unless a transition to a certain 2D vector value is to be created right after the scene is loaded or right after the `CoordinateDamper` node is created dynamically.

The `MFVec2f` arrays that are sent to the `set_destination` or `set_value` field shall have the same length (number of elements). The length of the arrays shall not change over time. Values assigned to `initialDestination` or `initialValue` shall either be an empty array or an array with the same number of elements as is sent to the `set_destination` or `set_value` fields. In any other case, the behavior is not defined.

## 39.5 Support levels

The Followers component provides one level of support as specified in [Table 39.2](#).

**Table 39.2 — Followers component support levels**

Level	Prerequisites	Nodes/Features	Support
1	Core 1 Grouping 1 Shape 1 Rendering 1		
		<code>X3DChaserNode</code>	n/a
		<code>X3DDamperNode</code>	n/a
		<code>X3DFollowerNode</code>	n/a
			All fields fully

		ColorChaser	supported.
		ColorDamper	All fields fully supported.
		CoordinateChaser	All fields fully supported.
		CoordinateDamper	All fields fully supported.
		OrientationChaser	All fields fully supported.
		OrientationDamper	All fields fully supported.
		PositionChaser	All fields fully supported.
		PositionChaser2D	All fields fully supported.
		PositionDamper	All fields fully supported.
		PositionDamper2D	All fields fully supported.
		<del>ScalerChaser</del> ScalerChaser	All fields fully supported.
		<del>ScalerDamper</del> ScalerDamper	All fields fully supported.
		TexCoordChaser	All fields fully supported.
		TexCoordDamper	All fields fully supported.





## Extensible 3D (X3D) Part 1: Architecture and base components

### 19 Interpolation component



#### 19.1 Introduction

##### 19.1.1 Name

The name of this component is "Interpolation". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 19.1.2 Overview

This subclause describes the Interpolation component of this part of ISO/IEC 19775. [Table 19.1](#) provides links to the major topics in this subclause.

**Table 19.1 — Topics**

- [19.1 Introduction](#)
  - [19.1.1 Name](#)
  - [19.1.2 Overview](#)
- [19.2 Concepts](#)
  - [19.2.1 Interpolators](#)
  - [19.2.2 Linear interpolation](#)
  - [19.2.3 Non-linear interpolation](#)
  - [19.2.4 Hermite spline interpolation](#)
- [19.3 Abstract types](#)
  - [19.3.1 X3DInterpolatorNode](#)
- [19.4 Node reference](#)
  - [19.4.1 ColorInterpolator](#)
  - [19.4.2 CoordinateInterpolator](#)
  - [19.4.3 CoordinateInterpolator2D](#)
  - [19.4.4 EaseInEaseOut](#)
  - [19.4.5 NormalInterpolator](#)
  - [19.4.6 OrientationInterpolator](#)
  - [19.4.7 PositionInterpolator](#)

- [19.4.8 PositionInterpolator2D](#)
- [19.4.9 ScalarInterpolator](#)
- [19.4.10 SplinePositionInterpolator](#)
- [19.4.11 SplinePositionInterpolator2D](#)
- [19.4.12 SplineScalarInterpolator](#)
- [19.4.13 SquadOrientationInterpolator](#)
- [19.5 Support levels](#)
- [Table 19.1 — Topics](#)
- [Table 19.2 — Interpolation component support levels](#)
- [Figure 19.1 — EaseInEaseOut algorithm illustration](#)

## 19.2 Concepts

This clause includes six Interpolator nodes all of which provide keyframe-based animation capability.

### 19.2.1 Interpolators

The Interpolator nodes provide interpolation between animation key frame values. The following node types are Interpolator nodes, each based on the type of value that is interpolated:

- [ColorInterpolator](#)
- [CoordinateInterpolator](#)
- [CoordinateInterpolator2D](#)
- [NormalInterpolator](#)
- [OrientationInterpolator](#)
- [PositionInterpolator](#)
- [PositionInterpolator2D](#)
- [ScalarInterpolator](#)
- [SplinePositionInterpolator](#)
- [SplinePositionInterpolator2D](#)
- [SplineScalarInterpolator](#)
- [SquadOrientationInterpolator](#)
- [EaseInEaseOut](#)

All Interpolator nodes are based on the abstract type [X3DInterpolatorNode](#).

### 19.2.2 Linear interpolation

The X3D interpolator nodes specified in this clause are designed for linear key framed animation. Each of these nodes defines a piecewise-linear function,  $f(t)$ , on the interval  $(-\infty, \infty)$ . The piecewise-linear function is defined by  $n$  values of  $t$ , called *key*, and the  $n$  corresponding values of  $f(t)$ , called *keyValue*. The keys shall be monotonically non-decreasing, otherwise the results are undefined.

An interpolator node evaluates  $f(t)$  given any value of  $t$  (via the *fraction* field) as follows: Let the  $n$  keys  $t_0, t_1, t_2, \dots, t_{n-1}$  partition the domain  $(-\infty, \infty)$  into the  $n+1$  subintervals given by  $(-\infty, t_0), [t_0, t_1), [t_1, t_2), \dots, [t_{n-1}, +\infty)$ . Also, let the  $n$  values  $v_0, v_1, v_2, \dots, v_{n-1}$  be the values of  $f(t)$  at the associated key values. The piecewise-linear interpolating function,  $f(t)$ , is defined to be:

$$\begin{aligned} f(t) &= v_0, & \text{if } t \leq t_0, \\ &= v_{n-1}, & \text{if } t \geq t_{n-1}, \\ &= \mathit{linterp}(t, v_i, v_{i+1}), & \text{if } t_i \leq t \leq t_{i+1}, \end{aligned}$$

where  $\mathit{linterp}(t, x, y)$  is the linear interpolant,  $i$  belongs to  $\{0, 1, \dots, n-2\}$ .

The third conditional value of  $f(t)$  allows the defining of multiple values for a single key, (*i.e.*, limits from both the left and right at a discontinuity in  $f(t)$ ). The first specified value is used as the limit of  $f(t)$  from the left, and the last specified value is used as the limit of  $f(t)$  from the right. The value of  $f(t)$  at a multiply defined key is indeterminate, but should be one of the associated limit values.

### 19.2.3 Non-linear interpolation

This component also provides non-linear interpolator nodes that provide for smoother animation than the linear interpolator nodes. Linear interpolators tend to produce animations that have a discontinuous velocity vectors. The transitions at the keys produce a noticeably jerky effect which will not occur when using non-linear interpolator nodes.

The non-linear interpolator nodes consist of three spline interpolator nodes for 3D, 2D, and scalar interpolation. These three nodes use the Hermite spline interpolation with adjustments to accommodate non-uniform key intervals (see [19.2.4 Hermite spline interpolation](#)). The [SquadOrientationInterpolator](#) node supports non-linear orientation interpolation.

Each of non-linear interpolator nodes provides a SFBool *closed* field that specifies whether the interpolator should provide a closed loop, with continuous velocity vectors as the interpolator transitions from the last key to the first key. If the velocity vectors at the first and last keys are specified, the closed field is ignored. If the keyValues at the first and last key are not identical, the closed field is ignored.

The SFBool *normalizeVelocity* field specifies whether the velocity vectors are to be transformed into tangency vectors. If the *normalizeVelocity* field has value `TRUE`, the *keyVelocity* values are normalized, thus converting them to tangency vectors. In this case, the vectors are normalized to produce smooth speed transitions, as described mathematically below in [19.2.4 Hermite spline interpolation](#). The magnitude of the specified velocity vectors is ignored.

If the *normalizeVelocity* field has value `FALSE`, the units specified in the velocity field are defined to be `length/cycleInterval`.

EXAMPLE Using a [SplinePositionInterpolator](#), in which the velocity at a key is specified to be (0, 0, 1), and the

cycleInterval that drives the interpolator is 4 seconds, the actual speed of the object at that key will be 0.25 metres per second (assuming the initial base units have been specified).

In addition to the interpolation nodes, this component provides a node that modifies the time fraction that is typically fed from the [TimeSensor](#) node into the interpolator node. This is the [EaseInEaseOut](#) node. It allows for a deceleration as the interpolator approaches a key, and an acceleration as the interpolator exits a key. Authors can route time fraction events into the EaseInEaseOut node. The EaseInEaseOut node will then send out a modified time fraction, which can then be routed into one or more interpolators.

## 19.2.4 Hermite spline interpolation

The [SplinePositionInterpolator](#), [SplinePositionInterpolator2D](#), and the [SplineScalarInterpolator](#) nodes all use the Hermite spline interpolation with adjustments to accommodate non-uniform key intervals. These three nodes all use the same algorithm which is described below.

The algorithm used by these interpolators is as follows. It defines the output value sent from the *value\_changed* field for a given segment of the interpolation, between key(i), and key(i+1). This segment is valid when the fraction value satisfies ( $t_i \leq \text{fraction} < t_{i+1}$ ), where  $t_i$  is the key at (i), and  $t_{i+1}$  is the key at (i+1).

The local fraction will vary from zero to one between the two keys, as follows:

$$s = (t - t_i) / (t_{i+1} - t_i)$$

The velocity vectors at key (i) and key (i+1) are denoted by  $\mathbf{T}_i$  and  $\mathbf{T}_{i+1}$  respectively. These velocity vectors need not be unit vectors. The magnitude of these vectors specifies the relative speed of the interpolation.

If the size of the *keyVelocity* field is equal to the size of the *keyValue* field, the values of  $\mathbf{T}$  used below should come from the *keyVelocity* field. If the size of the *keyVelocity* field is 2, the first value is used as the velocity vector for the first key, and the second value is used as the velocity for the last key. If the size of the *keyVelocity* field is anything other than those two values, the *keyVelocity* field is ignored. Any velocity vectors that are not specified will be calculated using the following algorithm:

The *keyValue* at key (i) is denoted as  $\mathbf{v}_i$  and the *keyValue* at key (i+1) is denoted as  $\mathbf{v}_{i+1}$ .

With those parameters defined, the *value\_changed* value ( $\mathbf{v}_s$ ) can be calculated as follows:

$$\mathbf{v}_s = \mathbf{S}^T \mathbf{H} \mathbf{C}$$

where

$$\mathbf{S} = \begin{bmatrix} s^3 \\ s^2 \end{bmatrix} \quad \mathbf{H} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} \mathbf{v}_i \\ \mathbf{v}_{i+1} \end{bmatrix}$$

$$\begin{vmatrix} s \\ 1 \end{vmatrix} \quad \begin{vmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{vmatrix} \quad \begin{vmatrix} \mathbf{T}_i^0 \\ \mathbf{T}_{i+1}^1 \end{vmatrix}$$

The values of  $\mathbf{T}_i^0$  and  $\mathbf{T}_{i+1}^1$  are defined as follows:

The standard Hermite spline assumes that the keys are equally spaced. Since this is not a valid assumption, these values are adjusted to calculate  $\mathbf{T}_i^0$  and  $\mathbf{T}_{i+1}^1$  as follows. If the velocity vector is specified by the author, the value of  $\mathbf{T}_i$  is extracted from the *keyVelocity* field for the specific key.

If the velocity vector is not specified, it is calculated as follows:

$$\mathbf{T}_i = (\mathbf{v}_{i+1} - \mathbf{v}_{i-1}) / 2$$

There are special cases as specified below:

If the velocity vector is specified, and the *normalizeVelocity* flag has value `FALSE`, the velocity at the key is set to the corresponding value of the *keyVelocity* field:

$$\mathbf{T}_i = \text{keyVelocity}[i]$$

If the velocity vector is specified, and the *normalizeVelocity* flag is `TRUE`, the velocity at the key is set using the corresponding value of the *keyVelocity* field:

$$\mathbf{T}_i = \text{keyVelocity}[i] \times (D_{\text{tot}} / |\text{keyVelocity}[i]|)$$

where:

$D_{\text{tot}}$  is the sum of the distance between all adjacent keys.

or

$$D_{\text{tot}} = \text{SUM}\{i=0, i < n-1\} (|\mathbf{v}_i - \mathbf{v}_{i+1}|)$$

Lastly, to accommodate the non-uniform key intervals, the values of  $\mathbf{T}_i^0$  and  $\mathbf{T}_i^1$  are calculated as follows:

$$\mathbf{T}_i^0 = \mathbf{F}^{+i} \mathbf{T}_i$$

$$\mathbf{T}_i^1 = \mathbf{F}^{-i} \mathbf{T}_i$$

where:

$$\mathbf{F}^{-i} = 2 (t_{i+1} - t_i) / (t_{i+1} - t_{i-1})$$

$$\mathbf{F}^{+i} = 2 (t_i - t_{i-1}) / (t_{i+1} - t_{i-1})$$

If the interpolator is closed, the values of the *key* and *keyValue* used in these calculations should wrap appropriately:

$$t_{-1} = t_{N-2}$$

$$V_{-1} = V_{N-2}$$

$$t_N = t_1$$

$$V_N = t_1$$

If the interpolator is not closed, and the first and last velocity vectors are not specified by the author, the values are calculated as follows:

$$\mathbf{T}^0_0 = \mathbf{T}^1_0 = \mathbf{T}^0_{N-1} = \mathbf{T}^1_{N-1} = 0$$

If the interpolator is not closed, and the first and last velocity vectors are specified by the author, the values are calculated as follows:

$$\mathbf{T}^0_0 = \mathbf{T}_0$$

$$\mathbf{T}^1_{N-1} = \mathbf{T}_{N-1}$$

where N is the size of the *keyValue* field.

Additional information on the Hermite algorithm is available in [\[CATROM\]](#).

## 19.3 Abstract types

### 19.3.1 *X3DInterpolatorNode*

```
X3DInterpolatorNode : X3DChildNode {
  SFFloat [in] set_fraction (-∞,∞)
  MFFloat [in,out] key [] (-∞,∞)
  MF<type> [in,out] keyValue []
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  [S|M]F<type> [out] value_changed
}
```

The abstract node *X3DInterpolatorNode* forms the basis for all types of interpolators specified in this clause.

The *key* field contains the list of key times, which **could** **might** appear as:

```
key [0 0.25 0.65 0.75 1]
```

to indicate there are five key frames in this node. The *keyValue* field contains values for the target field, one complete set of values for each key. Interpolator nodes containing no keys in the *key* field shall not produce any events. However, an input event that replaces an empty *key* field with one that contains keys will cause the interpolator node to produce events the next time that a *set\_fraction* event is received..

The *set\_fraction* inputOnly field receives an SFFloat event and causes the interpolator node function to evaluate, resulting in a *value\_changed* output event of the specified type with the same timestamp as the *set\_fraction* event.

The contents of the *keyValue* and *value\_changed* fields are dependent on the type of the node (e.g., the PositionInterpolator fields use MFVec3f values). Each value or set of values in the *keyValue* field corresponds in order to the parameter value in the *key* field.



For interpolator nodes that produce a single value, results are undefined if the number of values in the *key* field is not the same as the number of values in the *keyValue* field.

For interpolator nodes that produce multiple values, the *keyValue* field is an  $n \times m$  array of values, where  $n$  is the number of values in the *key* field and  $m$  is the number of values at each key frame. Each  $m$  values in the *keyValue* field correspond, in order, to a parameter value in the *key* field. Each *value\_changed* event shall contain  $m$  interpolated values. Results are undefined if the number of values in the *keyValue* field is not a positive integer multiple of the number of values in the *key* field.

If an *X3DInterpolatorNode* *value\_changed* outputOnly field is read before it receives any inputs, *keyValue*[0] is returned if *keyValue* is not empty. If *keyValue* is empty (*i.e.*, [ ]), the initial value for the respective field type is returned (EXAMPLE (0, 0, 0) for SFVec3f); see [5 Field type reference](#) for initial event values.

The location of an *X3DInterpolatorNode* in the transformation hierarchy has no effect on its operation. For example, if a parent of an interpolator node is a [Switch](#) node with *whichChoice* set to  $-1$  (*i.e.*, ignore its children), the interpolator continues to operate as specified (receives and sends events).

A typical simplified structure for a key frame animation implementation involves a [TimeSensor](#), ROUTEs, and the target node.

```

Transform {
  Shape
  IndexedFaceSet { coordIndex='... -1 ... >'
    Coordinate DEF='Moved' point [ x y z, ... ] # t0Geometry
  }
}

CoordinateInterpolator DEF='Mover'
key [t0 t1 t2] # list of key times, 0 to 1
keyValue ' x y z, ... ' # one geometry per key time

TimeSensor DEF='Timer' cycleInterval 5 loop TRUE

ROUTE Timer.fraction_changed TO Mover.set_value
ROUTE Mover.value_changed TO Moved.point

```

In typical operation, the key frame *set\_fraction* event arrives from a *TimeSensor* to signal that the time value has advanced. This value varies from 0 to 1 depending upon where the *TimeSensor* is in its cycle time.

EXAMPLE If the *TimeSensor* has a cycleTime of 10 seconds, and 5 seconds has elapsed in its cycle, the *set\_fraction* value will be 0.5.

In this sample structure, the [IndexedFaceSet](#) contains a [Coordinate](#) field named *Moved*. This defines the time equals zero geometry for the node. The [CoordinateInterpolator](#) node named *Mover* contains the list of key frame times and the corresponding sets of coordinates in the *keyValue* field. When the *set\_fraction* event arrives for *key*, the corresponding interpolated *keyValue* is sent to the target *Coordinate* node for rendering.

## 19.4 Node reference

### 19.4.1 ColorInterpolator

```

ColorInterpolator : X3DInterpolatorNode {
  SFFloat [in] set_fraction (-∞,∞)

```

```

MFFloat [in,out] key      [] (-∞,∞)
MFColor [in,out] keyValue [] [0,1]
SFNode [in,out] metadata  NULL [X3DMetadataObject]
SFColor [out] value_changed
}

```

The ColorInterpolator node interpolates among a list of MFColor key values to produce an SFColor (RGB) *value\_changed* event. The number of colours in the *keyValue* field shall be equal to the number of key frames in the *key* field. The *keyValue* field and *value\_changed* events are defined in RGB colour space. A linear interpolation using the value of *set\_fraction* as input is performed in HSV space (see [\[FOLEY\]](#) for description of RGB and HSV colour spaces). The results are undefined when interpolating between two consecutive keys with complementary hues.

## 19.4.2 CoordinateInterpolator

```

CoordinateInterpolator : X3DInterpolatorNode {
  SFFloat [in] set_fraction (-∞,∞)
  MFFloat [in,out] key      [] (-∞,∞)
  MFVec3f [in,out] keyValue [] (-∞,∞)
  SFNode [in,out] metadata  NULL [X3DMetadataObject]
  MFVec3f [out] value_changed
}

```

The CoordinateInterpolator node linearly interpolates among a list of MFVec3f values to produce an MFVec3f *value\_changed* event. The number of coordinates in the *keyValue* field shall be an integer multiple of the number of key frames in the *key* field. That integer multiple defines how many coordinates will be contained in the *value\_changed* events.

## 19.4.3 CoordinateInterpolator2D

```

CoordinateInterpolator2D : X3DInterpolatorNode {
  SFFloat [in] set_fraction (-∞,∞)
  MFFloat [in,out] key      [] (-∞,∞)
  MFVec2f [in,out] keyValue [] (-∞,∞)
  SFNode [in,out] metadata  NULL [X3DMetadataObject]
  MFVec2f [out] value_changed
}

```

This node linearly interpolates among a list of MFVec2f values to produce an MFVec2f *value\_changed* event. The number of coordinates in the *keyValue* field shall be an integer multiple of the number of key frames in the *key* field. That integer multiple defines how many coordinates will be contained in the *value\_changed* events.

## 19.4.4 EaseInEaseOut

```

EaseInEaseOut : X3DNode {
  SFFloat [in] set_fraction (-∞,∞)
  MFVec2f [in,out] easeInEaseOut [] (-∞,∞)
  MFFloat [in,out] key          [] (-∞,∞)
  SFNode [in,out] metadata      NULL [X3DMetadataObject]
  SFFloat [out] modifiedFraction_changed
}

```

The EaseInEaseOut node supports controlled gradual transitions by specifying modifications for TimeSensor node fractions. The EaseInEaseOut node receives a *set\_fraction* field event. It uses the values of the *key* field and the *easeInEaseOut* field to modify that fraction which is then issued as a *modifiedFraction\_changed* event.

The first components of each pair of *easeInEaseOut* field Vec2f values correspond to the easeIn/easeOut features following each key as the interpolator progresses. The second components of each pair of *easeInEaseOut* field Vec2f values correspond to the

easeIn/easeOut features as the interpolator progresses between *keyValues*.

The values of the *easeInEaseOut* field range from zero to one. At zero, there is no modification of the fraction.

The scope of the easeOut effect on the local fraction is equal to the easeOut value.

**EXAMPLE 1** If the easeOut value is 0.4, the object will accelerate out of the previous key and reach a constant speed at 40% of the way from the previous key to the next key.

The scope of the easeIn effect on the local fraction begins when the local fraction reaches  $(1.0 - \text{easeIn})$ .

**EXAMPLE 2** If the easeIn value is 0.3, the object will transition from a constant speed, and begin to decelerate when unmodified local fraction reaches 0.7 (70% of the way from the previous key to the next key).

If the sum of the previous easeIn value, plus the next easeIn value is greater than 1.0, both values are scaled by the same amount so that the sum of the values is equal 1.0. In that case, there is no period of constant speed.

The algorithm for computing the *modifiedFraction\_changed* value is:

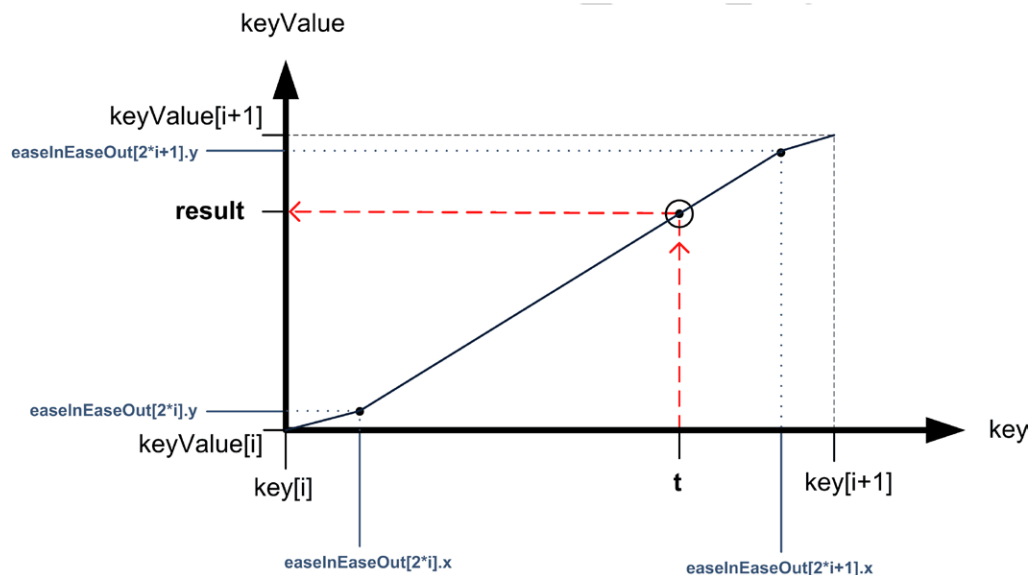
- a. Let  $u$  be the value of the *set\_fraction* field, representing the proportion the distance between  $\text{key}_i$  and  $\text{key}_{i+1}$ .
- b. Let  $e_{\text{out}}$  be the easeOut value for  $\text{key}_i$ ; (i.e.,  $\text{easeOut} = \text{easeInEaseOut}_{i.y}$ ).
- c. Let  $e_{\text{in}}$  be the easeIn value for  $\text{key}_{i+1}$ ; (i.e.,  $\text{easeIn} = \text{easeInEaseOut}_{i+1.x}$ ).
- d. Let  $S$  be the sum of  $e_{\text{in}}$  and  $e_{\text{out}}$ .
- e. If  $S < 0$ , *modifiedFraction\_changed* is set to  $u$ .
- f. If  $S > 1.0$ , divide  $e_{\text{in}}$  and  $e_{\text{out}}$  by  $S$ .
- g. Compute  $t = 1.0 / (2.0 - e_{\text{out}} - e_{\text{in}})$ .
- h. If  $u < e_{\text{out}}$ , *modifiedFraction\_changed* is set to:  

$$(t / e_{\text{out}}) \times u^2$$
- i. If  $u < 1.0 - e_{\text{in}}$ , *modifiedFraction\_changed* is set to:  

$$(t \times (2u - e_{\text{out}}))$$
- j. Else, *modifiedFraction\_changed* is set to:  

$$1.0 - ((t \times (1.0 - u)^2) / e_{\text{in}})$$

[Figure 19.1](#) illustrates the algorithm above.



**Figure 19.1 — EaseInEaseOut algorithm illustration**

The `easeInEaseOut` field values shall be monotonically non-decreasing, otherwise results are undefined.

### 19.4.5 NormalInterpolator

```
NormalInterpolator : X3DInterpolatorNode {
  SFFloat [in]  set_fraction  (-∞,∞)
  MFFloat [in,out] key      [] (-∞,∞)
  MFVec3f [in,out] keyValue [] (-∞,∞)
  SFNode [in,out] metadata  NULL [X3DMetadataObject]
  MFVec3f [out]  value_changed
}
```

The `NormalInterpolator` node interpolates among a list of normal vector sets specified by the `keyValue` field to produce an `MFVec3f value_changed` event. The output vector, `value_changed`, shall be a set of normalized vectors.

Values in the `keyValue` field shall be of unit length. The number of normals in the `keyValue` field shall be an integer multiple of the number of key frames in the `key` field. That integer multiple defines how many normals will be contained in the `value_changed` events.

Normal interpolation shall be performed on the surface of the unit sphere. That is, the output values for a linear interpolation from a point *P* on the unit sphere to a point *Q* also on the unit sphere shall lie along the shortest arc (on the unit sphere) connecting points *P* and *Q*. Also, equally spaced input fractions shall result in arcs of equal length. The results are undefined if *P* and *Q* are diagonally opposite.

### 19.4.6 OrientationInterpolator

```
OrientationInterpolator : X3DInterpolatorNode {
  SFFloat [in]  set_fraction  (-∞,∞)
  MFFloat [in,out] key      [] (-∞,∞)
  MFRotation [in,out] keyValue [] [-1,1] or (-∞,∞)
  SFNode [in,out] metadata  NULL [X3DMetadataObject]
  SFRotation [out] value_changed
}
```

The `OrientationInterpolator` node interpolates among a list of rotation values specified in the `keyValue` field to produce an `SFRotation value_changed` event. These rotations

are absolute in object space and therefore are not cumulative. The *keyValue* field shall contain exactly as many rotations as there are key frames in the *key* field.

An orientation represents the final position of an object after a rotation has been applied. An OrientationInterpolator interpolates between two orientations by computing the shortest path on the unit sphere between the two orientations. The interpolation is linear in arc length along this path. The results are undefined if the two orientations are diagonally opposite.

If two consecutive *keyValue* values exist such that the arc length between them is greater than  $\pi$ , the interpolation will take place on the arc complement. For example, the interpolation between the orientations

(0, 1, 0, 0) and (0, 1, 0, 5.0)

is equivalent to the rotation between the orientations

(0, 1, 0,  $2\pi$ ) and (0, 1, 0, 5.0).

### 19.4.7 PositionInterpolator

```
PositionInterpolator : X3DInterpolatorNode {
  SFFloat [in]  set_fraction  (-∞,∞)
  MFFloat [in,out] key        [] (-∞,∞)
  MFVec3f [in,out] keyValue   [] (-∞,∞)
  SFNode [in,out] metadata    NULL [X3DMetadataObject]
  SFVec3f [out]  value_changed
}
```

The PositionInterpolator node linearly interpolates among a list of 3D vectors to produce an SFVec3f *value\_changed* event. The *keyValue* field shall contain exactly as many values as in the *key* field.

### 19.4.8 PositionInterpolator2D

```
PositionInterpolator2D : X3DInterpolatorNode {
  SFFloat [in]  set_fraction  (-∞,∞)
  MFFloat [in,out] key        [] (-∞,∞)
  MFVec2f [in,out] keyValue   [] (-∞,∞)
  SFNode [in,out] metadata    NULL [X3DMetadataObject]
  SFVec2f [out]  value_changed
}
```

The PositionInterpolator node linearly interpolates among a list of 2D vectors to produce an SFVec2f *value\_changed* event. The *keyValue* field shall contain exactly as many values as in the *key* field.

### 19.4.9 ScalarInterpolator

```
ScalarInterpolator : X3DInterpolatorNode {
  SFFloat [in]  set_fraction  (-∞,∞)
  MFFloat [in,out] key        [] (-∞,∞)
  MFFloat [in,out] keyValue   [] (-∞,∞)
  SFNode [in,out] metadata    NULL [X3DMetadataObject]
  SFFloat [out]  value_changed
}
```

The ScalarInterpolator node linearly interpolates among a list of SFFloat values to produce an SFFloat *value\_changed* event. This interpolator is appropriate for any parameter defined using a single floating point value.

EXAMPLE 1 width fields

**EXAMPLE 2** radius fields**EXAMPLE 3** intensity fields

The *keyValue* field shall contain exactly as many numbers as there are key frames in the *key* field.

## 19.4.10 SplinePositionInterpolator

```
SplinePositionInterpolator : X3DInterpolatorNode {
  SFFloat [in]  set_fraction      (-∞,∞)
  SFBool  [in,out] closed        FALSE
  MFFloat [in,out] key           []  (-∞,∞)
  MFVec3f [in,out] keyValue      []  (-∞,∞)
  MFVec3f [in,out] keyVelocity   []  (-∞,∞)
  SFNode  [in,out] metadata      NULL [X3DMetadataObject]
  SFBool  [in,out] normalizeVelocity FALSE
  SFVec3f [out]  value_changed
}
```

The SplinePositionInterpolator node non-linearly interpolates among a list of 3D vectors to produce an SFVec3f *value\_changed* event. The *keyValue*, *keyVelocity*, and *key* fields shall each have the same number of values.

## 19.4.11 SplinePositionInterpolator2D

```
SplinePositionInterpolator2D : X3DInterpolatorNode {
  SFFloat [in]  set_fraction      (-∞,∞)
  SFBool  [in,out] closed        FALSE
  MFFloat [in,out] key           []  (-∞,∞)
  MFVec2f [in,out] keyValue      []  (-∞,∞)
  MFVec2f [in,out] keyVelocity   []  (-∞,∞)
  SFNode  [in,out] metadata      NULL [X3DMetadataObject]
  SFBool  [in,out] normalizeVelocity FALSE
  SFVec2f [out]  value_changed
}
```

The SplinePositionInterpolator2D node non-linearly interpolates among a list of 2D vectors to produce an SFVec2f *value\_changed* event. The *keyValue*, *keyVelocity*, and *key* fields shall each have the same number of values.

## 19.4.12 SplineScalarInterpolator

```
SplineScalarInterpolator : X3DInterpolatorNode {
  SFFloat [in]  set_fraction      (-∞,∞)
  SFBool  [in,out] closed        FALSE
  MFFloat [in,out] key           []  (-∞,∞)
  MFFloat [in,out] keyValue      []  (-∞,∞)
  MFFloat [in,out] keyVelocity   []  (-∞,∞)
  SFNode  [in,out] metadata      NULL [X3DMetadataObject]
  SFBool  [in,out] normalizeVelocity FALSE
  SFFloat [out]  value_changed
}
```

The SplineScalarInterpolator node non-linearly interpolates among a list of floats to produce an SFFloat *value\_changed* event. The *keyValue*, *keyVelocity*, and *key* fields shall each have the same number of values.

## 19.4.13 SquadOrientationInterpolator

```
SquadOrientationInterpolator : X3DInterpolatorNode {
  SFFloat [in]  set_fraction      (-∞,∞)
  MFFloat [in,out] key           []  (-∞,∞)
  MFRotation [in,out] keyValue   []  (-∞,∞)
  SFNode  [in,out] metadata      NULL [X3DMetadataObject]
  SFBool  [in,out] normalizeVelocity FALSE
  SFRotation [out]  value_changed
}
```

The SquadOrientationInterpolator node non-linearly interpolates among a list of rotations to produce an SFRotation *value\_changed* event. The *keyValue* field shall have the same number of values and the *key* field.

The SquadOrientationInterpolator uses the industry standard Squad method for smoothly interpolating orientations. Squad is an acronym for Spherical Cubic Interpolation. The Linear [OrientationInterpolator](#) described in [19.4.6 OrientationInterpolator](#) provides spherical linear interpolation. The SquadOrientationInterpolator applies the spline interpolation approach described above to interpolation in quaternion space. For more information on Squad interpolation, see [\[SHOE\]](#).

## 19.5 Support levels

The Interpolation component provides three levels of support as specified in [Table 19.2](#).

**Table 19.2 — Interpolation component support levels**

Level	Prerequisites	Nodes/Features	Support
<b>1</b>	Core 1 Grouping 1 Shape 1		
		<i>X3DInterpolatorNode</i> (abstract)	n/a
		CoordinateInterpolator	All fields fully supported.
		OrientationInterpolator	All fields fully supported.
		PositionInterpolator	All fields fully supported.
		ScalarInterpolator	All fields fully supported.
<b>2</b>	Core 1 Grouping 1 Shape 1		
		All Level 1 Interpolator nodes	All fields fully supported.
		ColorInterpolator	All fields fully supported.
		NormalInterpolator	All fields fully supported.

<b>3</b>	Core 1 Grouping 1 Shape 1		
		All Level 2 Interpolator nodes	All fields fully supported.
		CoordinateInterpolator2D	All fields fully supported.
		PositionInterpolator2D	All fields fully supported.
<b>4</b>	Core 1 Grouping 1 Shape 1		
		All Level 3 Interpolator nodes	All fields fully supported.
		EaseInEaseOut	All fields fully supported.
		SplinePositionInterpolator	All fields fully supported.
		SplinePositionInterpolator2D	All fields fully supported.
		SplineScalarInterpolator	All fields fully supported.
<b>5</b>	Core 1 Grouping 1 Shape 1		
		All Level 4 Interpolator nodes	All fields fully supported.
		SquadOrientationInterpolator	All fields fully supported.







# Extensible 3D (X3D)

## Part 1: Architecture and base components

### 40 Particle systems component



## 40.1 Introduction

### 40.1.1 Name

The name of this component is "ParticleSystems". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

### 40.1.2 Overview

This component specifies how to model particles and their interactions through the application of basic physics principles to affect motion. [Table 40.1](#) provides links to the major topics in this clause.

**Table 40.1 — Topics**

- [40.1 Introduction](#)
  - [40.1.1 Name](#)
  - [40.1.2 Overview](#)
- [40.2 Concepts](#)
  - [40.2.1 Overview](#)
  - [40.2.2 Physics models](#)
  - [40.2.3 Colour ramps](#)
  - [40.2.4 Randomness and variation](#)
  - [40.2.5 Event model interaction](#)
  - [40.2.6 Interaction with other components](#)
- [40.3 Abstract types](#)
  - [40.3.1 X3DParticleEmitterNode](#)
  - [40.3.2 X3DParticlePhysicsModelNode](#)
- [40.4 Node reference](#)
  - [40.4.1 BoundedPhysicsModel](#)
  - [40.4.2 ConeEmitter](#)
  - [40.4.3 ExplosionEmitter](#)

- [40.4.4 ForcePhysicsModel](#)
- [40.4.5 ParticleSystem](#)
- [40.4.6 PointEmitter](#)
- [40.4.7 PolylineEmitter](#)
- [40.4.8 SurfaceEmitter](#)
- [40.4.9 VolumeEmitter](#)
- [40.4.10 WindPhysicsModel](#)
- [40.5 Support levels](#)
- [Table 40.1 — Topics](#)
- [Table 40.2 — Particle systems component support levels](#)

## 40.2 Concepts

### 40.2.1 Overview

A particle system specifies a process for rendering such effects as fire, smoke, and snow. Although various physics models are available, it is not meant to be used as a simulation engine for testing particle behaviour models. Thus, particle systems are designed for visual effects, not rigid analysis systems.

A particle system is a shape node type as both appearance and texturing need to be controlled in order to create realistic particle system effects. A particle system in and of itself is not geometry because it dynamically creates and destroys geometry on the fly. A particle system also has other factors feeding into the visual output different from a typical geometry node.

EXAMPLE Time-varying colour values

The geometry node of a shape node is not used at the first support level. If a node supports both levels of the specification, the geometry node takes preference over the *geometryType* field.

Particles are generated using the current geometry, allowing particle geometry to be changed over time. Old particles that are still current when the specified geometry changes continue to use the original geometry.

### 40.2.2 Physics models

To allow aggregating a collection of nodes with a particular physics model, an *X3DParticlePhysicsModelNode* abstract node type is provided that is used to derive nodes in which all children nodes of that group are bound by the specified physics model. This can be used to support simple effects such as gravity. The physics models are designed to be composable.

EXAMPLE The [BoundedPhysicsModel](#) node can be used with an extrusion as the geometry and then a [WindPhysicsModel](#) node can be used with the geometry to produce smoke tunneling effects.

### 40.2.3 Colour ramps

A colour ramp is used to specify the colour cycle over time. A colour ramp is a variation on the normal use of a colour interpolator. The key represents relative time values from the start of the lifetime of the particle. There should be the same number of colours in the node as there are key values. If not, the smaller number of the two values will be used. For particles that last longer than the last time value, the colour associated with the last time value will be continued for the rest of the lifetime of the particle.

#### 40.2.4 Randomness and variation

Many nodes describe emission as using a random pattern. The random generation model shall use a linear distribution of equal probability across the defined output spectrum.

Nodes that describe a variation field allow for deviation from the described main field value. The variation is the maximum bound of that value, described as a proportion of the original value. A variation value of zero does not allow any randomness.

**EXAMPLE** If field has a value of 10, a variation of 0.25 will allow values to be randomly generated in the range of 7.5 to 12.5. The same field with a value of 1 will only allow a range of randomly generated values between 0.75 and 1.25.

#### 40.2.5 Event model interaction

Evaluation of emission and interactions of particles are performed as specified in [4.4.8.3 Execution model](#). All changes made during current event cascade are first applied. Then, the particles are generated, particle system physics are applied, and rendering of the particles takes place.

Particle systems interact with navigation, pointing device sensors, collision detection and picking according to the underlying geometry type. Point particles, point sprite particles and line particles cannot be picked; other types of particles can be picked.

#### 40.2.6 Interaction with other components

There are many different ways to implement particle systems. This specification does not require any particular technology or means of implementation. However, a popular implementation is to use programmable shaders to perform the per-frame evaluation of the particles. While it is not advised, it is possible and legal to supply programmable shaders as part of the particle system's *appearance* field. If the user supplies a programmable shader, the implementation shall use a CPU-generated model for this case.

The physics model nodes in this component are independent of the physics evaluation defined in [37 Rigid body physics component](#). This part of ISO/IEC 19775 does not explicitly prohibit interaction between the two models, but does not cater for direct implementation of one by the other (e. g., use of hardware accelerated physics cards to implement particle systems).

### 40.3 Abstract types

### 40.3.1 *X3DParticleEmitterNode*

```
X3DParticleEmitterNode : X3DNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFFloat [in,out] speed 0 [0,∞)
  SFFloat [in,out] variation 0.25 [0,∞)
  SFFloat [] mass 0 [0,∞)
  SFFloat [] surfaceArea 0 [0,∞)
}
```

The *X3DParticleEmitterNode* abstract type represents any node that is an emitter of particles. The shape and distribution of particles is dependent on the type of the concrete node.

The *speed* field specifies an initial linear speed that will be imparted to all particles. It does not signify the direction of the particles. The directional component of the velocity is specified by the concrete node representation.

The *variation* field specifies a multiplier for the randomness that is used to control the range of possible output values. The bigger the value, the more random the output and the bigger the range of possible initial values possible. A variation of zero does not allow any randomness.

The *mass* field specifies the basic mass of each particle in mass base units. Mass is needed if gravity or other force-related calculations are to be performed per-particle.

The *surfaceArea* field specifies the surface area of the particle in area base units. Surface area is used for calculations such as wind effects per particle. The *surfaceArea* field value represents an average frontal area that would be presented to the wind, assuming a spherical model for each particle (*i.e.*, the surface area is the same regardless of direction).

### 40.3.2 *X3DParticlePhysicsModelNode*

```
X3DParticlePhysicsModelNode : X3DNode {
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

The *X3DParticlePhysicsModelNode* abstract type represents any node that applies a form of constraints on the particles after they have been generated.

The *enabled* field specifies whether this physics model is currently being applied to the particles.

## 40.4 Node reference

### 40.4.1 BoundedPhysicsModel

```
BoundedPhysicsModel : X3DParticlePhysicsModelNode {
  SFBool [in,out] enabled TRUE
  SFNode [in,out] geometry NULL [X3DGeometryNode]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

The BoundedPhysicsModel node specifies a physics model that applies a user-defined set of geometrical bounds to the particles.

The *geometry* field specifies a piece of geometry that models the bounds that constrain

the location of the particles. When a particle touches the surface of the bounds, it is reflected. The particles may be restricted to an inside location or an outside location. All geometry defined by the bounds are considered to be non-solid, regardless of the setting of the *solid* field. It does not matter whether the particle impacts the front or back side of the geometry. Particles are reflected at the same angle to the normal of the surface to which they impact, continuing in the same direction. The calculation of the correct normal is determined by the rules of the geometry that forms the bounds.

EXAMPLE A particle can be made to bounce off an elevation grid representing terrain.

## 40.4.2 ConeEmitter

```
ConeEmitter : X3DParticleEmitterNode {
  SFFloat [in,out] angle     $\pi/4$  [0, $\pi$ ]
  SFVec3f [in,out] direction 0 1 0
  SFNode [in,out] metadata  NULL [X3DMetadataObject]
  SFVec3f [in,out] position  0 0 0
  SFFloat [in,out] speed    0 [0, $\infty$ )
  SFFloat [in,out] variation 0.25 [0, $\infty$ )
  SFFloat []    mass        0 [0, $\infty$ )
  SFFloat []    surfaceArea 0 [0, $\infty$ )
}
```

The ConeEmitter node is an emitter that generates all the available particles from a specific point in space. Particles are emitted from the single point specified by the *position* field emanating in a direction randomly distributed within the cone specified by the *angle* and *direction* fields at the speed specified by the *speed* field.

## 40.4.3 ExplosionEmitter

```
ExplosionEmitter : X3DParticleEmitterNode {
  SFNode [in,out] metadata  NULL [X3DMetadataObject]
  SFVec3f [in,out] position  0 0 0
  SFFloat [in,out] speed    0 [0, $\infty$ )
  SFFloat [in,out] variation 0.25 [0, $\infty$ )
  SFFloat []    mass        0 [0, $\infty$ )
  SFFloat []    surfaceArea 0 [0, $\infty$ )
}
```

The ExplosionEmitter node is an emitter that generates all the available particles from a specific point in space at the initial time. Particles are emitted from the single point specified by the *position* field in all directions at the speed specified by the *speed* field.

## 40.4.4 ForcePhysicsModel

```
ForcePhysicsModel : X3DParticlePhysicsModelNode {
  SFBool [in,out] enabled TRUE
  SFVec3f [in,out] force  0 -9.8 0 ( $\infty$ , $\infty$ )
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

The ForcePhysicsModel node specifies a physics model that applies a constant force value to the particles. Force may act in any given direction vector at any strength.

The *force* field is used to indicate the strength and direction of the force (e.g., gravity) that should be applied. Force is specified in force base units. If the particles are defined to have zero mass by the emitter, the ForcePhysicsModel node has no effect.

## 40.4.5 ParticleSystem

```
ParticleSystem : X3DShapeNode {
  SFNode [in,out] appearance  NULL [X3DAppearanceNode]
  SFBool [in,out] createParticles TRUE
}
```

```

SFNode [in,out] geometry      NULL    [X3DGeometryNode]
SFBool  [in,out] enabled      TRUE
SFFloat [in,out] lifetimeVariation 0.25  [0,1]
SFInt32 [in,out] maxParticles  200    [0,∞)
SFNode  [in,out] metadata     NULL    [X3DMetadataObject]
SFFloat [in,out] particleLifetime 5     [0,∞)
SFVec2f [in,out] particleSize  0.02 0.02 [0,∞)
SFBool  [out]  isActive
SFVec3f []     bboxCenter      0 0 0
SFVec3f []     bboxSize       -1 -1 -1 (0,∞) or -1 -1 -1
SFNode  []     colorRamp      NULL    [X3DColorNode]
MFFloat []     colorKey       []       [0,∞)
SFNode  []     emitter        NULL    [X3DParticleEmitterNode]
SFString []    geometryType  "QUAD" ["LINE"|"POINT"|"QUAD"|"SPRITE"|"TRIANGLE"|"GEOMETRY"...]
MFNode  []     physics        []       [X3DParticlePhysicsModelNode]
SFNode  []     texCoordRamp   NULL    [TextureCoordinate]
MFFloat []     texCoordKey    []       [0,∞)
}

```

The `ParticleSystem` node specifies a complete particle system.

The `geometryType` field specifies the type of geometry that should be used to represent individual particles. Typically, a particle is calculated as a point in space at which the geometry is placed and then rendered using the appearance attributes.

The types of geometry are defined to render in the following way:

- `"LINE"`: A line is drawn along the particle's current velocity vector, for this frame, centered about the particle's position. The length of the line is specified by the particle's height from the `particleSize` field value.
- `"POINT"`: A point geometry is rendered at the particle's position.
- `"QUAD"`: A 2D quad is rendered aligned in the local coordinate space of the particle system with the face normal pointing along the positive Z axis. Individual quads are not aligned to the user's eye position but are affected in depth by the physics model. The particle's position is at the center of the quad.
- `"SPRITE"`: A point sprite that uses a 2D point position to locate a screen-aligned quad at the center of the particle's location is rendered.
- `"TRIANGLE"`: A 2D quad is rendered using a pair of triangles aligned in the local coordinate space of the particle system with the face normal pointing along the positive Z axis. Individual triangles are not aligned to the user's eye position, but are effected in depth by the physics model. The particle's position is at the center of the triangle.
- `"GEOMETRY"`: The geometry specified by the `geometry` field is rendered for each particle using the local coordinate system. Changing the value of the `geometry` field or the definition of the geometry node shall be applied during current computation of the next frame to be rendered.

The `geometry` field specifies the geometry to be used for each particle when the `geometryType` field has value `"GEOMETRY"`.

The `appearance` field holds information that is used for the geometry. All effects, such as material colours and/or multi-textures, are applied to each particle. If a texture coordinate ramp and key is supplied with this geometry, it shall be used in preference to any automatic texture coordinate generation. If automatic texture coordinate generation is used, results shall be based on the entire volume that the particles consume, not locally applied to each particle.

Procedural shaders may also be supplied. The particle system shall manage the position of all particles each frame. This position becomes the initial geometry input to the

shader.

The *emitter* field specifies the type of emitter geometry and properties that the particles are given for their initial positions. After being created, the individual particles are then manipulated according to the physics model(s) specified in the *physics* field.

The *colorRamp* and *colorKey* fields specify how to change the base colour of the particle over the lifetime of an individual particle. The *colorKey* field represents the time of the particle in seconds, while the *colorRamp* field holds a series of colour values to be used at the given key points in time. Between keys, colour values are interpreted in a linear HSV space, using the same rules defined for the [ColorInterpolator](#) node. The colour values are defined as per-vertex colour values. Consequently, if an appearance node with material is provided, the material properties will override the colour ramp.

The *isActive* outputOnly field indicates whether the particle system is currently running, based on the setup of the node.

**EXAMPLE** Using an explosion emitter that generates all of its particles at the first time and has them all die at a fixed time later, the particle system will only run for a short amount of time. After that, nothing is visible on-screen or the particle geometry does not need updating any more.

The *isActive* field sends a value of `FALSE` when activity has stopped occurring. A particle system without an emitter set can never be active. If the emitter is defined by an EXTERNPROTO that has not yet resolved, *isActive* shall initially be `FALSE`, until the point the EXTERNPROTO has loaded and is verified as being a correct node type. If these validity checks pass, *isActive* is set to `TRUE` and this defines the local time zero to start the particle effects.

The *enabled* field controls whether this ParticleSystem is currently active and rendering particles this frame. Setting this value to `FALSE` will immediately remove all visible particles from the scene from the next frame onwards. Setting the field to `TRUE` will start the system again from a local time zero. It does not start off from where it was previously. In doing so, it will issue another value of `TRUE` for *isActive*. If a value of `FALSE` is set for *enabled*, *isActive* will also be set to `FALSE`.

The *createParticles* field is used to control whether any further new particles should be created. This allows the user to stop production of new particles, but keep those already existing in the scene to continue to animate. This differs from the *enabled* field that would immediately remove all particles. The *createParticles* field keeps the existing particles in existence until the end of their lifetimes. If there are no particles left in the scene, the system is still considered both active and enabled.

The *maxParticles* field specifies the maximum number of particles to be generated at one time (subject to player limitations). Support for at least 10,000 particles is required.

The *particleLifetime* field specifies the nominal duration in seconds of any particle.

The *lifetimeVariation* field specifies the proportion of the total lifetime that is the amount of allowed linear random variation from the value specified by the *particleLifetime* field.

The *particleSize* field describes the dimensions in length base units of the width and



height of each particle. Changing this value dynamically will only change new particles created after the change. Particles created before this timestamp will remain at the old size. This field only effects particles using "LINE", "QUAD", "SPRITE", and "TRIANGLE" geometry types.

The *texCoordRamp* and *texCoordKey* fields control the texture coordinates of the provided texture(s) in the Appearance node, over time. Particle systems frequently like to change the texture on a particle as it ages, yet there is no good way of accomplishing this through standard interpolators because interpolators have no concept of particle time. This pair of fields hold time-dependent values for the texture coordinates to be applied to the particle. When a particle reaches the next time stamp it moves to the next set of texture coordinates. There is no interpolation of the texture coordinates, just sequenced according to the times defined by *texCoordKey*.

The node placed in *texCoordRamp* shall have enough values to work with the numbers required by *geometryType*. The following numbers and rules for mapping texture coordinates to the quad shall be used:

- "LINE": The coordinates are paired such that the coordinate with lowest value index is associated with the end of the line that is closest to the emitter location and the coordinate with the next higher index is associated with the end of the line furthest from the emitter location. Each timestamp increases the index into the ramp by two.
- "POINT": Texture coordinates are ignored.
- "QUAD": Assuming a quad facing the current viewer position, coordinates are defined in a counter-clockwise order starting at the lower-left corner.
- "SPRITE": Texture coordinates are ignored for this type. Each particle uses the entire supplied texture.
- "TRIANGLE": Assuming two triangles facing the user, only four coordinates are supplied, representing the four corners of the quad. The order is the same as for "QUAD".
- "GEOMETRY": Texture coordinates ramps are ignored for this type. Texture coordinates from the geometry representation are used or automatic texture coordinate generation from the appearance node is used.

## 40.4.6 PointEmitter

```
PointEmitter : X3DParticleEmitterNode {
  SFVec3f [in,out] direction 0 1 0
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFVec3f [in,out] position 0 0 0
  SFFloat [in,out] speed 0 [0,∞)
  SFFloat [in,out] variation 0.25 [0,∞)
  SFFloat [] mass 0 [0,∞)
  SFFloat [] surfaceArea 0 [0,∞)
}
```

The PointEmitter node is an emitter that generates particles from a specific point in space. Particles are emitted from a single point in the specified direction and speed.

The *direction* field specifies a direction along which the particles are to be emitted. If the vector is zero length (a value of (0,0,0), particles are emitted in random directions from this point in space.



## 40.4.7 PolylineEmitter

```

PolylineEmitter : X3DParticleEmitterNode {
  MFInt32 [in]  set_coordIndex
  SFNode [in,out] coord      NULL [X3DCoordinateNode]
  SFVec3f [in,out] direction  0 1 0 [-1,1]
  SFNode [in,out] metadata    NULL [X3DMetadataObject]
  SFFloat [in,out] speed      0 [0,∞)
  SFFloat [in,out] variation  0.25 [0,∞)
  MFInt32 []   coordIndex    -1 [0,∞) or -1
  SFFloat []   mass          0 [0,∞)
  SFFloat []   surfaceArea   0 [0,∞)
}

```

The PolylineEmitter node emits particles along a single polyline. The coordinates for the line along which particles should be randomly generated are taken from a combination of the *coord* and *coordIndex* fields. The starting point for generating particles is randomly distributed along this line and given the initial speed and direction. If no coordinates are available, the PolylineEmitter node shall act like a point source located at the local origin.

## 40.4.8 SurfaceEmitter

```

SurfaceEmitter : X3DParticleEmitterNode {
  MFInt32 [in]  set_coordIndex
  SFNode [in,out] metadata    NULL [X3DMetadataObject]
  SFFloat [in,out] speed      0 [0,∞)
  SFFloat [in,out] variation  0.25 [0,∞)
  MFInt32 []   coordIndex    -1 [0,∞) or -1
  SFFloat []   mass          0 [0,∞)
  SFNode []    surface       NULL [X3DGeometryNode]
  SFFloat []   surfaceArea   0 [0,∞)
}

```

The SurfaceEmitter node is an emitter that generates particles from the surface of an object. New particles are generated by first randomly choosing a face on the tessellated geometry and then a random position on that face. Particles are generated with an initial direction of the normal to that point (including any normal averaging due to *normalPerVertex* and *creaseAngle* field settings). If the surface is indicated as not being solid (*solid* field set to `FALSE`), randomly choose from which side of the surface to emit, negating the normal direction when generating from the back side. Only valid geometry shall be used.

The *surface* field specifies the geometry to be used as the emitting surface.

EXAMPLE A cylinder with both end caps turned off would only generate particles along the side of the cylinder. It would be an error to generate a particle with an initial direction that is not perpendicular to the axis.

## 40.4.9 VolumeEmitter

```

VolumeEmitter : X3DParticleEmitterNode {
  MFInt32 [in]  set_coordIndex
  SFNode [in,out] coord      NULL [X3DCoordinateNode]
  SFVec3f [in,out] direction  0 1 0 [-1,1]
  SFNode [in,out] metadata    NULL [X3DMetadataObject]
  SFFloat [in,out] speed      0 [0,∞)
  SFFloat [in,out] variation  0.25 [0,∞)
  MFInt32 []   coordIndex    -1 [0,∞) or -1
  SFBool []    internal      TRUE
  SFFloat []   mass          0 [0,∞)
  SFFloat []   surfaceArea   0 [0,∞)
}

```

A VolumeEmitter node emits particles from a random position confined within the given closed geometry volume. Otherwise, a VolumeEmitter node acts like a [PolylineEmitter](#) node.

## 40.4.10 WindPhysicsModel

```
WindPhysicsModel : X3DParticlePhysicsModelNode {
  SFVec3f [in,out] direction 0 0 0 (∞,∞)
  SFBool [in,out] enabled TRUE
  SFFloat [in,out] gustiness 0.1 (0,∞)
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFFloat [in,out] speed 0.1 (0,∞)
  SFFloat [in,out] turbulence 0 (0,1)
}
```

The `WindPhysicsModel` node specifies a physics model that applies a wind effect to the particles. The wind has a random variation factor that allows for the gustiness of the wind to be modelled.

The *direction* field specifies the direction in which the wind is travelling in the form of a normalized, unit vector.

The *speed* field specifies the current wind speed in length base units. From the wind speed, the force applied per unit-area on the particle is calculated using the following formula:

$$\text{pressure} = 10^{(2 \times \log(\text{speed}))} \times 0.64615$$

The *gustiness* specifies how much the wind speed varies from the average value defined by the *speed* field. The wind speed variation is calculated once per frame and applied equally to all particles.

The *turbulence* field specifies how much the wind acts directly in line with the direction, and how much variation is applied in directions other than the wind direction. This is determined per-particle to model how the particle is effected by turbulence.

## 40.5 Support levels

The Particle Systems component provides two levels of support as specified in [Table 40.2](#).

**Table 40.2 — Particle systems component support levels**

Level	Prerequisites	Nodes/Features	Support
1	Core 1 Grouping 1 Shape 1 Rendering 1		
		<i>X3DParticleEmitterNode</i>	n/a
		<i>X3DParticlePhysicsModelNode</i>	n/a
		ConeEmitter	All fields fully supported.
		ExplosionEmitter	All fields fully supported.

		GravityPhysicsModel	All fields fully supported.
		ParticleSystem	All fields fully supported, except "SPRITE" and "GEOMETRY" geometry types and the <i>geometry</i> field.
		PointEmitter	All fields fully supported.
		PolylineEmitter	All fields fully supported.
		WindPhysicsModel	All fields fully supported.
<b>2</b>	Core 1 Grouping 1 Shape 1 Rendering 1 Texturing 1		
		All Level 1 nodes	All fields supported as specified for Level 1 except as specified below.
		BoundedPhysicsModel	All fields fully supported.
		ParticleSystem	All fields fully supported, except "GEOMETRY" geometry type and the <i>geometry</i> field.
		SurfaceEmitter	All fields fully supported.
		VolumeEmitter	All fields fully supported.
<b>3</b>	Core 1 Grouping 1 Shape 1 Rendering 1 Texturing 1		
		All Level 2 nodes	All fields fully supported.
		ParticleSystem	All fields fully supported.





# Extensible 3D (X3D)

## Part 1: Architecture and base components

### 20 Pointing device sensor component



## 20.1 Introduction

### 20.1.1 Name

The name of this component is "PointingDeviceSensor". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

### 20.1.2 Overview

This clause describes the Pointing Device Sensor component of this part of ISO/IEC 19775. This includes how pointing device sensors operate conceptually as well as which varieties of pointing device sensors are provided. [Table 20.1](#) provides links to the major topics in this clause.

**Table 20.1 — Topics**

- [20.1 Introduction](#)
  - [20.1.1 Name](#)
  - [20.1.2 Overview](#)
- [20.2 Concepts](#)
  - [20.2.1 Overview of pointing device sensors](#)
  - [20.2.2 Drag sensors](#)
  - [20.2.3 Activating and manipulating pointing device sensors](#)
- [20.3 Abstract types](#)
  - [20.3.1 X3DDragSensorNode](#)
  - [20.3.2 X3DPointingDeviceSensorNode](#)
  - [20.3.3 X3DTouchSensorNode](#)
- [20.4 Node reference](#)
  - [20.4.1 CylinderSensor](#)
  - [20.4.2 PlaneSensor](#)
  - [20.4.3 SphereSensor](#)
  - [20.4.4 TouchSensor](#)

- [20.5 Support levels](#)
- [Table 20.1 — Topics](#)
- [Table 20.2 — Pointing device sensor component support levels](#)

## 20.2 Concepts

### 20.2.1 Overview of pointing device sensors

Pointing device sensors detect user pointing events such as the user clicking on a piece of geometry (*i.e.*, [TouchSensor](#)). The following node types are pointing device sensors:

- [CylinderSensor](#)
- [PlaneSensor](#)
- [SphereSensor](#)
- [TouchSensor](#)

The [Anchor](#) node is also considered a pointing device sensor for the purpose of detecting user picking. However, it does not extend from the [X3DPointingDeviceSensorNode](#) interface.

Other components may add additional pointing device sensors.

A pointing device sensor is activated when the user locates the pointing device over geometry that is influenced by that specific pointing device sensor. Pointing device sensors have influence over all geometry that is descended from the sensor's parent groups. In the case of the Anchor node, the Anchor node itself is considered to be the parent group. Typically, the pointing device sensor is a sibling to the geometry that it influences. In other cases, the sensor is a sibling to groups which contain geometry (*i.e.*, are influenced by the pointing device sensor).

The appearance properties of the geometry do not affect activation of the sensor. In particular, transparent materials or textures shall be treated as opaque with respect to activation of pointing device sensors.

For a given user activation, the lowest enabled pointing device sensor in the hierarchy is activated. All other pointing device sensors above the lowest enabled pointing device sensor are ignored. The hierarchy is defined by the geometry node over which the pointing device sensor is located and the entire hierarchy upward. If there are multiple pointing device sensors tied for lowest, each of these is activated simultaneously and independently, possibly resulting in multiple sensors activating and generating output simultaneously. This feature allows combinations of pointing device sensors (*e.g.*, [TouchSensor](#) and [PlaneSensor](#)). If a pointing device sensor appears in the transformation hierarchy multiple times (DEF/USE), it shall be tested for activation in all of the coordinate systems in which it appears.

If a pointing device sensor is not enabled when the pointing device button is activated, it will not generate events related to the pointing device until after the pointing device is deactivated and the sensor is enabled (*i.e.*, enabling a sensor in the middle of

dragging does not result in the sensor activating immediately).

## 20.2.2 Drag sensors

*Drag sensors* are a subset of pointing device sensors. There are three types of drag sensors: [CylinderSensor](#), [PlaneSensor](#), and [SphereSensor](#). Drag sensors have two outputOnly fields in common, *trackPoint\_changed* and *<value>\_changed*. These outputOnly fields send events for each movement of the activated pointing device according to their "virtual geometry" (e.g., cylinder for [CylinderSensor](#)). The *trackPoint\_changed* outputOnly field sends the intersection point of the *bearing* with the drag sensor's virtual geometry. The *<value>\_changed* outputOnly field sends the sum of the relative change since activation plus the sensor's *offset* field. The type and name of *<value>\_changed* depends on the drag sensor type: *rotation\_changed* for [CylinderSensor](#), *translation\_changed* for [PlaneSensor](#), and *rotation\_changed* for [SphereSensor](#).

To simplify the application of these sensors, each node has an *offset* and an *autoOffset* exposed field. When the sensor generates events as a response to the activated pointing device motion, *<value>\_changed* sends the sum of the relative change since the initial activation plus the *offset* field value. If *autoOffset* is `TRUE` when the pointing device is deactivated, the *offset* field is set to the sensor's last *<value>\_changed* value and *offset* sends an *offset\_changed* output event. This enables subsequent grabbing operations to accumulate the changes. If *autoOffset* is `FALSE`, the sensor does not set the *offset* field value at deactivation (or any other time).

## 20.2.3 Activating and manipulating pointing device sensors

The pointing device controls a pointer in the virtual world. While activated by the pointing device, a sensor will generate events as the pointer moves. Typically the pointing device may be categorized as either 2D (e.g., conventional mouse) or 3D (e.g., wand). It is suggested that the pointer controlled by a 2D device is mapped onto a plane a fixed distance from the viewer and perpendicular to the line of sight. The mapping of a 3D device may describe a 1:1 relationship between movement of the pointing device and movement of the pointer.

The position of the pointer defines a bearing which is used to determine which geometry is being indicated. When implementing a 2D pointing device it is suggested that the bearing is defined by the vector from the viewer position through the location of the pointer. When implementing a 3D pointing device it is suggested that the bearing is defined by extending a vector from the current position of the pointer in the direction indicated by the pointer.

In all cases the pointer is considered to be indicating a specific geometry when that geometry is intersected by the bearing. If the bearing intersects multiple sensors' geometries, only the sensor nearest to the pointer will be eligible for activation.

## 20.3 Abstract types

### 20.3.1 X3DDragSensorNode

```

X3DDragSensorNode : X3DPointingDeviceSensorNode {
  SFBool [in,out] autoOffset TRUE
  SFString [in,out] description ""
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFBool [out] isActive
  SFBool [out] isOver
  SFVec3f [out] trackPoint_changed
}

```

This abstract node type is the base type for all drag-style pointing device sensors.

## 20.3.2 X3DPointingDeviceSensorNode

```

X3DPointingDeviceSensorNode : X3DSensorNode {
  SFString [in,out] description ""
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFBool [out] isActive
  SFBool [out] isOver
}

```

This abstract node type is the base type for all pointing device sensors.

## 20.3.3 X3DTouchSensorNode

```

X3DTouchSensorNode : X3DPointingDeviceSensorNode {
  SFString [in,out] description ""
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFBool [out] isActive
  SFBool [out] isOver
  SFTime [out] touchTime
}

```

This abstract node type is the base type for all touch-style pointing device sensors.

## 20.4 Node reference

### 20.4.1 CylinderSensor

```

CylinderSensor : X3DDragSensorNode {
  SFBool [in,out] autoOffset TRUE
  SFRotation [in,out] axisRotation 0 1 0 0
  SFString [in,out] description ""
  SFFloat [in,out] diskAngle  $\pi/12$  [0, $\pi/2$ ]
  SFBool [in,out] enabled TRUE
  SFFloat [in,out] maxAngle -1 [-2 $\pi$ ,2 $\pi$ ]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFFloat [in,out] minAngle 0 [-2 $\pi$ ,2 $\pi$ ]
  SFFloat [in,out] offset 0 (- $\infty$ , $\infty$ )
  SFBool [out] isActive
  SFBool [out] isOver
  SFRotation [out] rotation_changed
  SFVec3f [out] trackPoint_changed
}

```

The `CylinderSensor` node maps pointer motion (*e.g.*, a mouse or wand) into a rotation on an invisible cylinder that is aligned with the Y-axis of the local sensor coordinate system. The local sensor coordinate system is created by applying the `axisRotation` field value to the local coordinate system. The `CylinderSensor` uses the descendent geometry of its parent node to determine whether it is liable to generate events.

The `description` field in the `CylinderSensor` node specifies a textual description of the `CylinderSensor` node. This may be used by browser-specific user interfaces that wish to present users with more detailed information about the `CylinderSensor`.

The `enabled` field enables and disables the `CylinderSensor` node. If `TRUE`, the sensor



reacts appropriately to user events. If `FALSE`, the sensor does not track user input or send events. If `enabled` receives a `FALSE` event and `isActive` is `TRUE`, the sensor becomes disabled and deactivated, and outputs an `isActive FALSE` event. If `enabled` receives a `TRUE` event the sensor is enabled and ready for user activation.

A `CylinderSensor` node generates events when the pointing device is activated while the pointer is indicating any descendent geometry nodes of the sensor's parent group. See [20.2.3 Activating and manipulating pointing device sensors](#), for more details on using the pointing device to activate the `CylinderSensor`.

Upon activation of the pointing device while indicating the sensor's geometry, an `isActive TRUE` event is sent. The initial acute angle between the bearing vector and the local sensor coordinate system Y-axis of the `CylinderSensor` node determines whether the sides of the invisible cylinder or the caps (disks) are used for manipulation. If the initial angle is less than the `diskAngle`, the geometry is treated as an infinitely large disk lying in the `Y=0` plane of the local sensor coordinate system and coincident with the initial intersection point. Dragging motion is mapped into a rotation around the `+Y`-axis vector of the local sensor coordinate system. The perpendicular vector from the initial intersection point to this Y-axis defines zero rotation about the Y-axis of the local sensor coordinate system. For each subsequent position of the bearing, a `rotation_changed` event is sent that equals the sum of the rotation about the `+Y`-axis vector of the local sensor coordinate system (from the initial intersection to the new intersection) plus the `offset` value. `trackPoint_changed` events reflect the unclamped drag position on the surface of this disk. When the pointing device is deactivated and `autoOffset` is `TRUE`, `offset` is set to the last rotation angle and an `offset_changed` event is generated. See [20.2.2 Drag sensors](#), for a more general description of `autoOffset` and `offset` fields.

If the initial acute angle between the bearing vector and the local sensor coordinate system Y-axis of the `CylinderSensor` node is greater than or equal to `diskAngle`, the sensor behaves like a cylinder. The shortest distance between the point of intersection (between the bearing and the sensor's geometry) and the Y-axis of the parent group's local coordinate system determines the radius of an invisible cylinder used to map pointing device motion and marks the zero rotation value. For each subsequent position of the bearing, a `rotation_changed` event is sent that equals the sum of the right-handed rotation from the original intersection about the `+Y`-axis vector plus the `offset` value. `trackPoint_changed` events reflect the unclamped drag position on the surface of the invisible cylinder. When the pointing device is deactivated and `autoOffset` is `TRUE`, `offset` is set to the last rotation angle and an `offset_changed` event is generated. More details are available in [20.2.2 Drag sensors](#).

When the sensor generates an `isActive TRUE` event, it grabs all further motion events from the pointing device until it is released and generates an `isActive FALSE` event (other pointing device sensors shall not generate events during this time). Motion of the pointing device while `isActive` is `TRUE` is referred to as a "drag" operation. If a 2D pointing device is in use, `isActive` events will typically reflect the state of the primary button associated with the device (*i.e.*, `isActive` is `TRUE` when the primary button is pressed and `FALSE` when it is released). If a 3D pointing device (*e.g.*, a wand) is in use, `isActive` events will typically reflect whether the pointer is within or in contact with the sensor's geometry.

While the pointing device is activated, `trackPoint_changed` and `rotation_changed` events are output and are interpreted from pointing device motion based on the sensor's local

coordinate system at the time of activation. *trackPoint\_changed* events represent the unclamped intersection points on the surface of the invisible cylinder or disk. If the initial angle results in cylinder rotation (as opposed to disk behaviour) and if the pointing device is dragged off the cylinder while activated, browsers may interpret this in a variety of ways (e.g., clamp all values to the cylinder and continuing to rotate as the point is dragged away from the cylinder). Each movement of the pointing device while *isActive* is `TRUE` generates *trackPoint\_changed* and *rotation\_changed* events.

The *minAngle* and *maxAngle* fields clamp *rotation\_changed* events to a range of values. If *minAngle* is greater than *maxAngle*, *rotation\_changed* events are not clamped. The *minAngle* and *maxAngle* fields are restricted to the range  $[-2\pi, 2\pi]$ .

More information about this behaviour is described in [20.2 Concepts](#).

## 20.4.2 PlaneSensor

```
PlaneSensor : X3DDragSensorNode {
  SFBool [in,out] autoOffset      TRUE
  SFRotation [in,out] axisRotation  0 0 1 0
  SFString [in,out] description    ""
  SFBool [in,out] enabled         TRUE
  SFVec2f [in,out] maxPosition     -1 -1 (-∞,∞)
  SFNode [in,out] metadata        NULL [X3DMetadataObject]
  SFVec2f [in,out] minPosition     0 0 (-∞,∞)
  SFVec3f [in,out] offset         0 0 0 (-∞,∞)
  SFBool [out] isActive
  SFBool [out] isOver
  SFVec3f [out] trackPoint_changed
  SFVec3f [out] translation_changed
}
```

The *PlaneSensor* node maps pointing device motion into two-dimensional translation in a plane parallel to the  $Z=0$  plane of the local sensor coordinate system. The local sensor coordinate system is created by applying the *axisRotation* field value to the local coordinate system. The *PlaneSensor* node uses the descendent geometry of its parent node to determine whether it is liable to generate events.

The *description* field in the *PlaneSensor* node specifies a textual description of the *PlaneSensor* node. This may be used by browser-specific user interfaces that wish to present users with more detailed information about the *PlaneSensor*.

The *enabled* field enables and disables the *PlaneSensor*. If *enabled* is `TRUE`, the sensor reacts appropriately to user events. If *enabled* is `FALSE`, the sensor does not track user input or send events. If *enabled* receives a `FALSE` event and *isActive* is `TRUE`, the sensor becomes disabled and deactivated, and outputs an *isActive* `FALSE` event. If *enabled* receives a `TRUE` event, the sensor is enabled and made ready for user activation.

The *PlaneSensor* node generates events when the pointing device is activated while the pointer is indicating any descendent geometry nodes of the sensor's parent group. See [20.2.3 Activating and manipulating pointing device sensors](#), for details on using the pointing device to activate the *PlaneSensor*.

Upon activation of the pointing device (e.g., mouse button down) while indicating the sensor's geometry, an *isActive* `TRUE` event is sent. Pointer motion is mapped into relative translation in the *tracking plane*, (a plane parallel to the local sensor coordinate system  $Z=0$  plane and coincident with the initial point of intersection). For each subsequent movement of the bearing, a *translation\_changed* event is output which corresponds to the sum of the relative translation from the original intersection point to the intersection

point of the new bearing in the plane plus the *offset* value. The sign of the translation is defined by the  $Z=0$  plane of the local sensor coordinate system. *trackPoint\_changed* events reflect the unclamped drag position on the surface of this plane. When the pointing device is deactivated and *autoOffset* is `TRUE`, *offset* is set to the last *translation\_changed* value and an *offset\_changed* event is generated. More details are provided in [20.2.2 Drag sensors](#).

When the sensor generates an *isActive* `TRUE` event, it grabs all further motion events from the pointing device until it is deactivated and generates an *isActive* `FALSE` event. Other pointing device sensors shall not generate events during this time. Motion of the pointing device while *isActive* is `TRUE` is referred to as a "drag" operation. If a 2D pointing device is in use, *isActive* events typically reflect the state of the primary button associated with the device (*i.e.*, *isActive* is `TRUE` when the primary button is pressed, and is `FALSE` when it is released). If a 3D pointing device (*e.g.*, wand) is in use, *isActive* events typically reflect whether the pointer is within or in contact with the sensor's geometry.

*minPosition* and *maxPosition* may be set to clamp *translation\_changed* events to a range of values as measured from the origin of the  $Z=0$  plane of the local sensor coordinate system. If the X or Y component of *minPosition* is greater than the corresponding component of *maxPosition*, *translation\_changed* events are not clamped in that dimension. If the X or Y component of *minPosition* is equal to the corresponding component of *maxPosition*, that component is constrained to the given value. This technique provides a way to implement a line sensor that maps dragging motion into a translation in one dimension.

While the pointing device is activated and moved, *trackPoint\_changed* and *translation\_changed* events are sent. *trackPoint\_changed* events represent the unclamped intersection points on the surface of the tracking plane. If the pointing device is dragged off of the tracking plane while activated (*e.g.*, above horizon line), browsers may interpret this in a variety of ways (*e.g.*, clamp all values to the horizon). Each movement of the pointing device, while *isActive* is `TRUE`, generates *trackPoint\_changed* and *translation\_changed* events.

Further information about this behaviour can be found in [20.2 Concepts](#).

### 20.4.3 SphereSensor

```
SphereSensor : X3DDragSensorNode {
  SFBool   [in,out] autoOffset   TRUE
  SFString [in,out] description ""
  SFBool   [in,out] enabled     TRUE
  SFNode   [in,out] metadata    NULL [X3DMetadataObject]
  SFRotation [in,out] offset    0 1 0 0 [-1,1],(-∞,∞)
  SFBool   [out]   isActive
  SFBool   [out]   isOver
  SFRotation [out] rotation_changed
  SFVec3f  [out]   trackPoint_changed
}
```

The `SphereSensor` node maps pointing device motion into spherical rotation about the origin of the local coordinate system. The `SphereSensor` node uses the descendent geometry of its parent node to determine whether it is liable to generate events.

The *description* field in the `SphereSensor` node specifies a textual description of the `SphereSensor` node. This may be used by browser-specific user interfaces that wish to present users with more detailed information about the `SphereSensor`.

The *enabled* field enables and disables the SphereSensor node. If *enabled* is `TRUE`, the sensor reacts appropriately to user events. If *enabled* is `FALSE`, the sensor does not track user input or send events. If *enabled* receives a `FALSE` event and *isActive* is `TRUE`, the sensor becomes disabled and deactivated, and outputs an *isActive* `FALSE` event. If *enabled* receives a `TRUE` event the sensor is enabled and ready for user activation.

The SphereSensor node generates events when the pointing device is activated while the pointer is indicating any descendent geometry nodes of the sensor's parent group. See [20.2.3 Activating and manipulating pointing device sensors](#), for details on using the pointing device to activate the SphereSensor.

Upon activation of the pointing device (e.g., mouse button down) over the sensor's geometry, an *isActive* `TRUE` event is sent. The vector defined by the initial point of intersection on the SphereSensor's geometry and the local origin determines the radius of the sphere that is used to map subsequent pointing device motion while dragging. The virtual sphere defined by this radius and the local origin at the time of activation is used to interpret subsequent pointing device motion and is not affected by any changes to the sensor's coordinate system while the sensor is active. For each position of the bearing, a *rotation\_changed* event is sent which corresponds to the sum of the relative rotation from the original intersection point plus the *offset* value. *trackPoint\_changed* events reflect the unclamped drag position on the surface of this sphere. When the pointing device is deactivated and *autoOffset* is `TRUE`, *offset* is set to the last *rotation\_changed* value and an *offset\_changed* event is generated. See [20.2 Concepts](#), for more details.

When the sensor generates an *isActive* `TRUE` event, it grabs all further motion events from the pointing device until it is released and generates an *isActive* `FALSE` event (other pointing device sensors shall not generate events during this time). Motion of the pointing device while *isActive* is `TRUE` is termed a "drag" operation. If a 2D pointing device is in use, *isActive* events will typically reflect the state of the primary button associated with the device (i.e., *isActive* is `TRUE` when the primary button is pressed and `FALSE` when it is released). If a 3D pointing device (e.g., wand) is in use, *isActive* events will typically reflect whether the pointer is within (or in contact with) the sensor's geometry.

While the pointing device is activated, *trackPoint\_changed* and *rotation\_changed* events are output. *trackPoint\_changed* events represent the unclamped intersection points on the surface of the invisible sphere. If the pointing device is dragged off the sphere while activated, browsers may interpret this in a variety of ways (e.g., clamp all values to the sphere or continue to rotate as the point is dragged away from the sphere). Each movement of the pointing device while *isActive* is `TRUE` generates *trackPoint\_changed* and *rotation\_changed* events.

Further information about this behaviour can be found in [20.2 Concepts](#).

## 20.4.4 TouchSensor

```
TouchSensor : X3DTouchSensorNode {
  SFString [in,out] description      ""
  SFBool   [in,out] enabled         TRUE
  SFNode   [in,out] metadata        NULL [X3DMetadataObject]
  SFVec3f  [out]   hitNormal_changed
  SFVec3f  [out]   hitPoint_changed
  SFVec2f  [out]   hitTexCoord_changed
```



```

SFBool [out] isActive
SFBool [out] isOver
SFTime [out] touchTime
}

```

A TouchSensor node tracks the location and state of the pointing device and detects when the user points at geometry contained by the TouchSensor node's parent group.

The *description* field in the TouchSensor node specifies a textual description of the TouchSensor node. This may be used by browser-specific user interfaces that wish to present users with more detailed information about the TouchSensor.

A TouchSensor node can be enabled or disabled by sending it an *enabled* event with a value of `TRUE` or `FALSE`. If the TouchSensor node is disabled, it does not track user input or send events.

The TouchSensor generates events when the pointing device points toward any geometry nodes that are descendants of the TouchSensor's parent group. See [20.2.3 Activating and manipulating pointing device sensors](#), for more details on using the pointing device to activate the TouchSensor.

The *isOver* field reflects the state of the pointing device with regard to whether it is pointing towards the TouchSensor node's geometry or not. When the pointing device changes state from a position such that its bearing does not intersect any of the TouchSensor node's geometry to one in which it does intersect geometry, an *isOver* `TRUE` event is generated. When the pointing device moves from a position such that its bearing intersects geometry to one in which it no longer intersects the geometry, or some other geometry is obstructing the TouchSensor node's geometry, an *isOver* `FALSE` event is generated. These events are generated only when the pointing device has moved and changed 'over' state. Events are not generated if the geometry itself is animating and moving underneath the pointing device.

As the user moves the bearing over the TouchSensor node's geometry, the point of intersection (if any) between the bearing and the geometry is determined. Each movement of the pointing device, while *isOver* is `TRUE`, generates *hitPoint\_changed*, *hitNormal\_changed* and *hitTexCoord\_changed* events. *hitPoint\_changed* events contain the 3D point on the surface of the underlying geometry, given in the TouchSensor node's coordinate system. *hitNormal\_changed* events contain the surface normal vector at the *hitPoint*. *hitTexCoord\_changed* events contain the texture coordinates of that surface at the *hitPoint*. The values of *hitTexCoord\_changed* and *hitNormal\_changed* events are computed as appropriate for the associated shape.

If *isOver* is `TRUE`, the user may activate the pointing device to cause the TouchSensor node to generate *isActive* events (e.g., by pressing the primary mouse button). When the TouchSensor node generates an *isActive* `TRUE` event, it grabs all further motion events from the pointing device until it is released and generates an *isActive* `FALSE` event (other pointing device sensors will not generate events during this time). Motion of the pointing device while *isActive* is `TRUE` is termed a "drag" operation. If a 2D pointing device is in use, *isActive* events reflect the state of the primary button associated with the device (i.e., *isActive* is `TRUE` when the primary button is pressed and `FALSE` when it is released). If a 3D pointing device is in use, *isActive* events will typically reflect whether the pointing device is within (or in contact with) the TouchSensor node's geometry.

The field *touchTime* is generated when all three of the following conditions are true:

- a. The pointing device was pointing towards the geometry when it was initially activated (*isActive* is `TRUE`).
- b. The pointing device is currently pointing towards the geometry (*isOver* is `TRUE`).
- c. The pointing device is deactivated (*isActive* `FALSE` event is also generated).

More information about this behaviour is described in [20.2 Concepts](#).

## 20.5 Support levels

The Pointing Device Sensor component provides one level of support as specified in [Table 20.2](#).

**Table 20.2 — Pointing device sensor component support levels**

Level	Prerequisites	Nodes/Features	Support
<b>1</b>	Core 1 Grouping 1 Shape 1		
		<i>DragSensorNodeType</i> (abstract)	n/a
		<i>PointingDeviceSensorNodeType</i> (abstract)	n/a
		<i>TouchSensorNodeType</i> (abstract)	n/a
		CylinderSensor	All fields fully supported.
		PlaneSensor	All fields fully supported.
		SphereSensor	All fields fully supported.
		TouchSensor	All fields fully supported.



## Extensible 3D (X3D) Part 1: Architecture and base components

# 41 Volume rendering component



## 41.1 Introduction

### 41.1.1 Name

The name of this component is "VolumeRendering". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

### 41.1.2 Overview

This component provides the ability to specify and render volumetric data sets. [Table 41.1](#) provides links to the major topics in this clause.

**Table 41.1 — Topics**

- [41.1 Introduction](#)
  - [41.1.1 Name](#)
  - [41.1.2 Overview](#)
- [41.2 Concepts](#)
  - [41.2.1 Overview](#)
  - [41.2.2 Representing volumetric data](#)
    - [41.2.2.1 Registration and scaling](#)
    - [41.2.2.2 Data representation](#)
      - [41.2.2.2.1 3D texture definition](#)
      - [41.2.2.2.2 Vector and normal representation](#)
      - [41.2.2.2.3 Data optimization](#)
    - [41.2.2.3 Segmentation information](#)
    - [41.2.2.4 Tensor representation](#)
    - [41.2.2.5 Visual representation](#)
  - [41.2.3 Interaction with other nodes and components](#)
    - [41.2.3.1 Overview](#)
    - [41.2.3.2 Lighting](#)
    - [41.2.3.3 Geometry](#)
  - [41.2.4 Conformance](#)
    - [41.2.4.1 Dimensionality](#)
    - [41.2.4.2 Hardware requirements](#)
    - [41.2.4.3 Scene graph interaction](#)

- [41.3 Abstract types](#)
  - [41.3.1 X3DComposableVolumeRenderStyleNode](#)
  - [41.3.2 X3DVolumeDataNode](#)
  - [41.3.3 X3DVolumeRenderStyleNode](#)
- [41.4 Node reference](#)
  - [41.4.1 BlendedVolumeStyle](#)
  - [41.4.2 BoundaryEnhancementVolumeStyle](#)
  - [41.4.3 CartoonVolumeStyle](#)
  - [41.4.4 ComposedVolumeStyle](#)
  - [41.4.5 EdgeEnhancementVolumeStyle](#)
  - [41.4.6 IsoSurfaceVolumeData](#)
  - [41.4.7 OpacityMapVolumeStyle](#)
  - [41.4.8 ProjectionVolumeStyle](#)
  - [41.4.9 SegmentedVolumeData](#)
  - [41.4.10 ShadedVolumeStyle](#)
  - [41.4.11 SilhouetteEnhancementVolumeStyle](#)
  - [41.4.12 ToneMappedVolumeStyle](#)
  - [41.4.13 VolumeData](#)
- [41.5 Support levels](#)
- [Table 41.1 — Topics](#)
- [Table 41.2 — Mapping of texture colour components to 3D coordinates](#)
- [Table 41.3 — Weight function types](#)
- [Table 41.4 — Transfer function to weight mapping](#)
- [Table 41.5 — Transfer function mapping from texture type to texture coordinate](#)
- [Table 41.6 — Transfer function mapping from texture type to output colour](#)
- [Table 41.7 — Volume rendering component support levels](#)
- [Figure 41.1 — Torso in BlendedVolumeStyle](#)
- [Figure 41.2 — Default volume style on left and BoundaryEnhancementVolumeStyle on right](#)
- [Figure 41.3 — Default volume style on left and CartoonVolumeStyle on right](#)
- [Figure 41.4 — Default volume style on left and ComposedVolumeStyle on right](#)
- [Figure 41.5 — Default volume style on left and EdgeEnhancementVolumeStyle on right](#)
- [Figure 41.6 — IsoSurface volume data using CartoonVolumeStyle](#)
- [Figure 41.7 — Default volume style on left and OpacityMapVolumeStyle on right](#)
- [Figure 41.8 — Illustration of values selected when using MIP or LMIP volume rendering styles](#)
- [Figure 41.9 — Default volume style on left and MIP ProjectionVolumeStyle on right](#)
- [Figure 41.10 — Segmented volume data using OpacityMapVolumeStyle and ToneMappedVolumeStyle](#)
- [Figure 41.11 — Default volume style on left and ShadedVolumeStyle on right](#)
- [Figure 41.12 — Default volume style on left and SilhouetteEnhancementVolumeStyle on right](#)
- [Figure 41.13 — Default volume style on left and ToneMappedVolumeStyle on right](#)
- [Figure 41.14 — Volume data using default volume style](#)

## 41.2 Concepts

### 41.2.1 Overview



Volume rendering is an alternate form of visual data representation compared to the traditional polygonal form used in the rest of this part of ISO/IEC 19775. Whereas polygons represent a portion of an infinitely thin plane, volume data represents a three-dimensional portion of space. When polygonal data representing a volume in space is sliced, such as with a clipping plane, there is empty space. In the same situation, volumetric data shows the internals of that volume.

There are many different techniques for implementing rendering of volumetric data. This component does not define the technique used to render the data, only the type of visual output to be produced. In addition, it defines several different types of data representations for which the renderings may be applied. To implement some of the higher-complexity representations, the implementer may need to use a more complex rendering technique than the simpler representations (though this is not required). Each of the rendering nodes represents the visual output required, not the technique used to implement that visual output. Most of the rendering styles defined in this component are formally defined in [\[FOLEY\]](#).

## 41.2.2 Representing volumetric data

### 41.2.2.1 Coordinate system

Volumetric data consists of a set of aligned 2D textures. The coordinate system places the 2D textures in the volume such that each 2D texture lies in the XY-plane, with the depth increasing away from the viewer along the +Z axis.

NOTE This, effectively, inverts the 3D texture coordinates for the R axis direction, which defines them to have depth increasing along the -Z axis (see [Figure 33.1](#)).

The volume is centered around the local origin and is subject to the parent transformation hierarchy, including scales, shears and rotations.

#### 41.2.2.1 Registration and scaling

Volumetric data represents volume information that often comes from the real world or is computationally generated.

EXAMPLE Human body scans are from the real world while simulated stress analysis of an engine part is computationally generated.

The volumetric data is typically part of a larger environment space and thus needs to be located within that space so that volumes for different parts (*e.g.*, an arm and leg of a single human) may be presented in a spatially correct manner. Typically, volumes are not a unit cube in size. Thus, additional dimensional information accompanies the volume to indicate its true size in the local coordinate system.

#### 41.2.2.2 Data representation

##### 41.2.2.2.1 3D texture definition

Volume rendering requires the data be provided in a volumetric form. This component uses the 3D texturing component (see [33 Texturing3D component](#)) to represent the raw volume data, but without rendering that data directly onto polygonal surfaces. Volumetric rendering may make use of multiple 3D textures to generate a final visual form.

Data may be represented using between one and four colour components. How each colour

component is to be interpreted as part of the rendering is defined for each node. Some nodes may require a specific minimum number of components or define that anything more than a specific number are to be ignored. Providing extra data may not be helpful to the implementation. In cases where not enough components are provided (e.g., a surface normal texture only being defined with a one or two component colour image), the entire data source is ignored.

#### 41.2.2.2.2 Vector and normal representation

Some nodes make use of 3D textures to convey data other than colour.

EXAMPLE Normal or other vector information may be included.

For the purposes of representing 3D information, the 3D texture components shall be interpreted as defined by [Table 41.2](#).

**Table 41.2 — Mapping of texture colour components to 3D coordinates**

Color Component	3D Coordinate
Red	X
Green	Y
Blue	Z
Alpha	Ignored

If the texture provided for the field does not contain enough colour components for the data to be represented, it shall be ignored and the node's default behaviour used.

If a rendering style requires a surface normal value and is required to implicitly calculate one, the normal at a given voxel is the normalized gradient of the scalar field at that voxel location.

#### 41.2.2.2.3 Data optimization

An implementation is free to provide whatever data reduction techniques are appropriate during pre-processing prior to rendering. Within a specific volume data representation, the implementation may also perform its own optimization techniques.

EXAMPLE Automatic mipmapping may occur.

Volume visualization data sets are not required to be represented in sizes that are powers of two. Implementations may need to internally pad the texture sizes for passing to the underlying rendering engine, but user-provided content is not required to do this.

#### 41.2.2.3 Segmentation information

The volume data may optionally represent segmented data sets. Doing so requires representing the data in a slightly different manner than a standard volume data set. Therefore, a separate node is provided. Segmentation data takes the form of an additional volume of data where each voxel represents a segment ID value in addition to other values represented in each voxel. The segmentation information is used by the rendering process to

control how each voxel is to be rendered. It is not unusual to use segmentation information to render each segment identifier with a different style.

EXAMPLE Bone may be rendered using isosurfaces while skin may be rendered using tone shading.

#### 41.2.2.4 Tensor representation

This part of ISO/IEC 19775 does not explicitly handle or represent tensor data (*i.e.*, higher-order products of functions that are each applied to a set of variables). Nevertheless, tensor information may be rendered using the techniques in this International Standard even though no direct data is being transmitted. It is recommended that, if an application needs to know about the existence of tensor data, the metadata capabilities of this part of ISO/IEC 19775 also be used.

#### 41.2.2.5 Visual representation

Volumetric data is typically given as a 3D rectangular block of information. Turning that densely packed information into something meaningful where internal structures may be discernable is the job of the rendering process. However, there is not a single uniform approach to volume rendering. A technique that is good for exposing structures for medical visualization may be poor for fluid simulation visualization.

To allow for the production of different visual outputs, the Volume rendering component separates the scenegraph into two sets of responsibilities:

- a. nodes for representing the volume data, and
- b. nodes for rendering that volume data in different ways.

In this way, the same rendering process may be used for different sets of volume data where varying rendering styles may be used to highlight different structures within the one volume.

Many rendering techniques map volume data to a visual representation through the use of another texture known as a Transfer function. This secondary texture defines the colours to use, acting as a form of lookup table. Transfer functions can be defined in one, two, or three dimensions. A one-dimensional texture capability can be achieved through the use of a 2D texture that is only one pixel wide.

### 41.2.3 Interaction with other nodes and components

#### 41.2.3.1 Overview

Volumetric rendering requires a completely different implementation path from traditional polygonal rendering. The data represents not only surface information, but also colour and potentially lighting information as well. As such, volume rendering occupies the role in the renderable scenegraph of an X3DShapeNode rather than as individual geometry or appearance information.

#### 41.2.3.2 Lighting

Volumetric rendering is not required to follow the standard lighting equations specified in [17 Lighting component](#). Many techniques include the ability to self-light and self-shadow using information from the parent scene graph (*e.g.*, light scoping).

The volume data is rendered using one or more rendering styles. Each rendering style defines its own lighting equation that takes the colour and opacity value from the previously evaluated

style, modifies the lighting equation according to the local style rules, and generates an output colour and opacity value. The first rendering style that is applied to the voxel obtains the source values directly from the voxel data using the colour and/or opacity channels as needed. Typically, the first rendering style the used to render the volume data are transfer functions and the [OpacityMapVolumeStyle](#).

Many of these rendering styles involve non-photorealistic rendering effects. Each style presents its own lighting equation specifying how to get from the underlying voxel representation to the contributed output colour. The following are some common terms that are found in the lighting equations:

- $O_v$ : The initial opacity of the object prior to the use of this rendering style. If this is the first rendering style applied to the object, this is the value of the alpha component of the voxel being evaluated.
- $O_g$ : The output opacity of the object resulting from evaluating this rendering style.
- $C_v$ : The initial colour of the object prior to the use of this rendering style. If this is the first rendering style applied to the object, this is the value of the colour components of the voxel being applied.
- $C_g$ : The output colour of the object resulting from evaluating this rendering style.
- $\Delta f$ : The normalized value gradient of the voxel. This is the rate of change of the value relative to the values in neighbouring voxels.
- $\mathbf{V}$ : The vector from the viewer's position to the voxel being evaluated, in the local coordinate space of the volume data.
- $\mathbf{n}$ : The local surface normal. This may be provided by the user through another 3D texture that contains a surface normal for each voxel or else is internally calculated through algorithmic means.
- $\mathbf{L}_i$ : Light direction vector from light source  $i$ . Typically, this is part of a summation over all light sources affecting the volume.

When determining the view direction for any lighting or rendering calculations, the view direction is calculated from the user's current location in the world to the current voxel being processed. Lighting and rendering style calculations are assumed to be individually calculated for each voxel.

### 41.2.3.3 Geometry

The volumetric rendering nodes representing geometry are leaf nodes in the renderable tree. Volumetric nodes may exist as part of a shared scene graph with DEF/USE.

## 41.2.4 Conformance

### 41.2.4.1 Dimensionality

The minimum required voxel dimensions that shall be supported are 256x256x256.

### 41.2.4.2 Hardware requirements

There are no specific requirements for hardware acceleration of this component. In addition, this component does not define the specific implementation strategy to be used by a given rendering style. It is as equally valid to implement the code using simple multi-pass rendering as it is to use hardware shaders.

### 41.2.4.3 Scene graph interaction

For minimum conformance, sensor nodes that require interaction with the geometry (e.g., TouchSensor) shall provide intersection information based on the volume's bounds. An implementation may optionally provide real intersection information based on performing ray casting into the volume space and reporting the first non-transparent voxel hit.

Navigation and collision detection also require a minimal conformance requirement of using the bounds of the volume. In addition, the implementation may allow greater precision with non-opaque voxels in a similar manner to the sensor interactions.

## 41.3 Abstract types

### 41.3.1 X3DComposableVolumeRenderStyleNode

```
X3DComposableVolumeRenderStyleNode : X3DVolumeRenderStyleNode {
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

The *X3DComposableVolumeRenderStyleNode* abstract node type is the base type for all node types that allow rendering styles to be sequentially composed together to form a single renderable output. The output of one style may be used as the input of the next style. Composition in this manner is performed using the [ComposedVolumeStyle](#) node.

### 41.3.2 X3DVolumeDataNode

```
X3DVolumeDataNode : X3DChildNode, X3DBoundedObject {
  SFVec3f [in,out] dimensions 1 1 1 (0,∞)
  SFBool [in,out] bboxDisplay FALSE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFBool [in,out] visible TRUE
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}
```

The *X3DVolumeDataNode* abstract node type is the base type for all node types that describe volumetric data to be rendered. It sits at the same level as the polygonal *X3DShapeNode* (see [12.3.4 X3DShapeNode](#)) within the scene graph structure, but defines volumetric data rather than polygonal data.

The *dimensions* field specifies the dimensions of this geometry in the local coordinate space using standard X3D length base units. It is assumed the volume is centered around the local origin. If the *bboxSize* field is set, it typically has the same value as the *dimensions* field.

If one of the dimension values is zero, the *X3DVolumeDataNode* shall be rendered as a plane. If two of the dimension values are zero, the *X3DVolumeDataNode* shall be rendered as a line. If all three dimension values are zero, the *X3DVolumeDataNode* shall be rendered as a point.

### 41.3.3 X3DVolumeRenderStyleNode

```
X3DVolumeRenderStyleNode : X3DNode {
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

The *X3DVolumeRenderStyleNode* abstract node type is the base type for all node types that specify a specific visual rendering style to be used when rendering volume data.

The *enabled* field defines whether this rendering style is currently applied to the volume data. If the field is set to `FALSE`, the rendering shall not be applied. The result of rendering with the *enabled* field set to `FALSE` shall act as though no volume data is provided. Effectively, this allows

turning on and off volume rendering of specific parts of the volume without needing to add or remove style definitions from the volume data node.

## 41.4 Node reference

### 41.4.1 BlendedVolumeStyle

```

BlendedVolumeStyle : X3DComposableVolumeRenderStyleNode {
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFNode [in,out] renderStyle NULL [X3DComposableVolumeRenderStyleNode]
  SFNode [in,out] voxels NULL [X3DTexture3DNode]
  SFFloat [in,out] weightConstant1 0.5 [0,1]
  SFFloat [in,out] weightConstant2 0.5 [0,1]
  SFString [in,out] weightFunction1 "CONSTANT" ["CONSTANT", "ALPHA0", "ALPHA1", "TABLE",
    "ONE_MINUS_ALPHA0", "ONE_MINUS_ALPHA1"]
  SFString [in,out] weightFunction2 "CONSTANT" ["CONSTANT", "ALPHA0", "ALPHA1", "TABLE",
    "ONE_MINUS_ALPHA0", "ONE_MINUS_ALPHA1"]
  SFNode [in,out] weightTransferFunction1 NULL [X3DTexture2DNode]
  SFNode [in,out] weightTransferFunction2 NULL [X3DTexture2DNode]
}

```

The `BlendedVolumeStyle` combines the rendering of the parent volume data set and the rendering of a second specified volume data set into one by blending the values according to a weight function. The first data set is the data set that is specified by the parent `VolumeData` or `SegmentedVolumeData` node. The second data set and its render style is defined by the `voxels` and `renderStyle` fields specified by this `BlendedVolumeStyle` node. For the latter case, the value specified by the `renderStyle` field is applied to the voxels specified by the `voxels` field. The result is blended with the current state of voxels from the parent `VolumeData` or `SegmentedVolumeData` node. Those voxels are either the original parent voxels with default `OpacityMapVolumeStyle` applied or the result of any previous render styles having been applied by a `ComposedVolumeStyle` node.

The final colour is determined by:

$$C_g = \text{clamp}_{[0-1]}(C_v \times w_1 + C_{\text{blend}} \times w_2)$$

$$O_g = \text{clamp}_{[0-1]}(O_v \times w_1 + O_{\text{blend}} \times w_2)$$

where  $C_{\text{blend}}$  and  $O_{\text{blend}}$  is the color and alpha value of the second data set after the rendering style has been applied. The values of  $w_1$  and  $w_2$  depend on the `weightFunction1` and `weightFunction2` fields, respectively, as defined in [Table 41.3](#).

**Table 41.3 — Weight function types**

Value	Description of weightFunction1	Description of weightFunction2
"CONSTANT"	Use <i>weightConstant1</i> .	Use <i>weightConstant2</i> .
"ALPHA1"	Use $O_v$ .	Use $O_v$ .
"ALPHA2"	Use $O_{\text{blend}}$ .	Use $O_{\text{blend}}$ .
"ONE_MINUS_ALPHA1"	Use $1 - O_v$ .	Use $1 - O_v$ .
"ONE_MINUS_ALPHA2"	Use $1 - O_{\text{blend}}$ .	Use $1 - O_{\text{blend}}$ .
"TABLE"	Use the lookup value for texture coordinate $(O_v, O_{\text{blend}})$ in <i>weightTransferFunction1</i> and map	Use the lookup value for texture coordinate $(O_v, O_{\text{blend}})$ in <i>weightTransferFunction2</i> and map



	to weight value according to <a href="#">Table 41.4</a> or use $O_v$ if <i>weightTransferFunction1</i> is NULL.	to weight value according to <a href="#">Table 41.4</a> or use $O_v$ if <i>weightTransferFunction2</i> is NULL.
--	---	---

The *weightTransferFunction1* and *weightTransferFunction2* fields specify two-dimensional textures that are used to determine the weight values when the weight function is set to "TABLE". The output weight value depends on the number of components in the textures as specified in [Table 41.4](#).

**Table 41.4 — Transfer function to weight mapping**

Number of Texture Components	Texel Components	Weight
1	Luminance (L)	Luminance component (L)
2	Luminance Alpha (LA)	Luminance component (L)
3	RGB	Red component (R)
4	RGBA	Red component (R)

[Figure 41.1](#) depicts a human torso and part of a skull (OpacityMapRenderStyle) blended with a blue/yellow tone-mapped volume of the internal organs. The image shows how BlendedVolumeStyle allows two different volumes to be combined, each with its own render style.

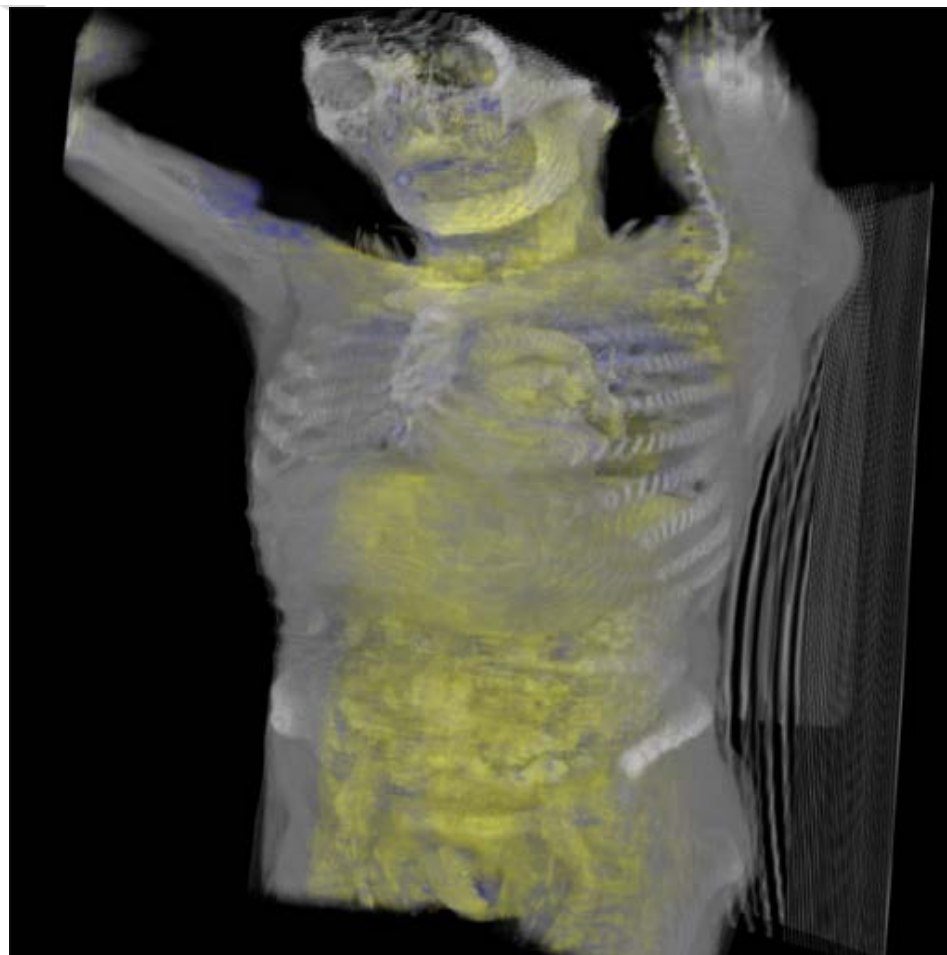


Figure 41.1 — Torso in BlendedVolumeStyle

## 41.4.2 BoundaryEnhancementVolumeStyle

```
BoundaryEnhancementVolumeStyle : X3DComposableVolumeRenderStyleNode {
  SFFloat [in,out] boundaryOpacity 0.9 [0,1]
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFFloat [in,out] opacityFactor 2 [0,∞)
  SFFloat [in,out] retainedOpacity 0.2 [0,1]
}
```

The `BoundaryEnhancementVolumeStyle` node provides boundary enhancement for the volume rendering style. In this rendering style, the colour rendered is based on the gradient magnitude. Faster-changing gradients (surface normals) are darker than slower-changing gradients. Thus, regions of different density are made more visible relative to parts that are of relatively constant density.

The `surfaceNormals` field is used to provide pre-calculated surface normal information for each voxel. If provided, this shall be used for all lighting calculations. If not provided, the implementation shall automatically generate surface normals using an implementation-specific method. If a value is provided, it shall be exactly the same voxel dimensions as the base volume data that it represents. If the dimensions are not identical, the browser shall generate a warning and automatically generate its own internal normals as though no value was provided for this field.

The output opacity for this rendering style is obtained by combining a fraction of the volume's original opacity with an enhancement based on the local boundary strength (*i.e.*, magnitude of the gradient between adjacent voxels). Colour components from the input are transferred unmodified to the output. The function used is:

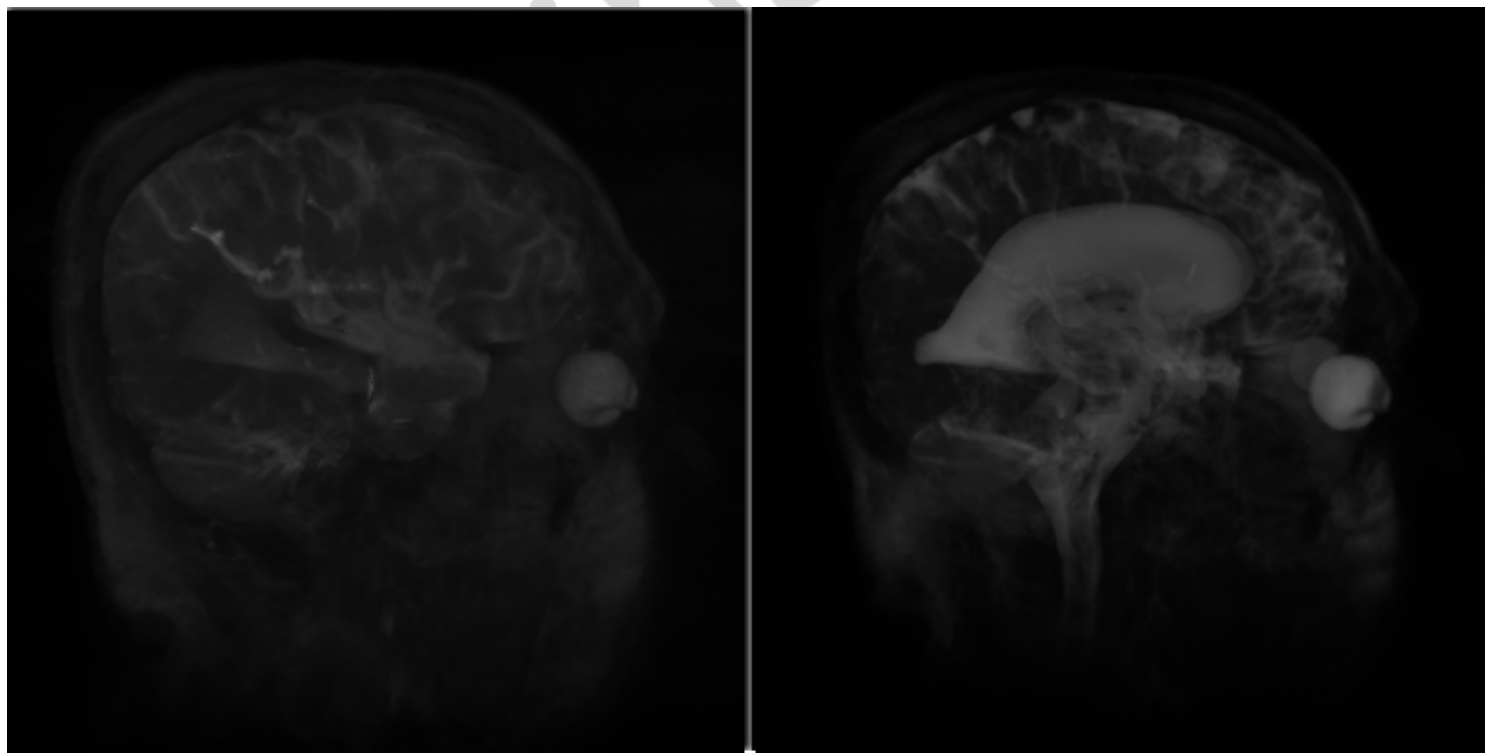


$$O_g = O_v (k_{gc} + k_{gs}(|\Delta f|)^{k_{ge}})$$

where

- the operator " $^$ " means "to the power"
- $O_g$  is the computed opacity of the voxel
- $O_v$  is the original opacity of the voxel
- $k_{gc}$  is the amount of initial opacity to mix into the output (*retainedOpacity*).
- $k_{gs}$  is the factored amount of the gradient enhancement to use (*boundaryOpacity*).
- $k_{ge}$  is the power function to control the slope of the opacity curve to highlight the set of data (*opacityFactor*).
- $|\Delta f|$  is the absolute value of the forward difference between the current and next voxel.

[Figure 41.2](#) shows a basic image of ventricles of the brain on the left and an image of ventricles of the brain using `BoundaryEnhancementVolumeStyle` on the right.



**Figure 41.2 — On the left, the ventricle with default render style and default field values. On the right, the ventricle using `BoundaryEnhancementVolumeStyle` with default values: `boundaryOpacity=0.9`, `opacityFactor=2`, and `retainedOpacity=0.2`.**

### 41.4.3 `CartoonVolumeStyle`

```

CartoonVolumeStyle : X3DComposableVolumeRenderStyleNode {
  SFInt32  [in,out] colorSteps  4  [1,64]
  SFBool   [in,out] enabled    TRUE
  SFNode   [in,out] metadata   NULL  [X3DMetadataObject]
  SFColorRGBA [in,out] orthogonalColor  1 1 1 1 [0,1]
  SFColorRGBA [in,out] parallelColor   0 0 0 1 [0,1]
  SFNode   [in,out] surfaceNormals  NULL  [X3DTexture3DNode]
}

```

The `CartoonVolumeStyle` generate a cartoon-style non-photorealistic rendering of the associated volumetric data. Cartoon rendering uses two colours that are rendered in a series of distinct flat-shaded sections based on the local surface normal's closeness to the average

normal with no gradients in between.

The *surfaceNormals* field contains a 3D texture with at least three component values. Each voxel in the texture represents the surface normal direction for the corresponding voxel in the base data source. This texture should be identical in dimensions to the source data. If not, the implementation may interpolate or average between adjacent voxels to determine the average normal at the voxel required. If this field is empty, the implementation shall automatically determine the surface normal using algorithmic means.

The *parallelColor* field specifies the colour to be used for surface normals that are orthogonal to the viewer's current location (where the plane of the surface itself is parallel to the user's view direction).

The *orthogonalColor* field specifies the colour to be used for surface normals that are parallel to the viewer's current location (the plane of the surface itself is orthogonal to the user's view direction). Surfaces that are backfacing are not rendered and shall have no colour calculated for them.

The *colorSteps* field indicates how many distinct colours are taken from the interpolated colours and used to render the object. If the value is 1, no colour interpolation takes place and only the orthogonal colour is used to render the surface. For any other value, the colours are interpolated between *parallelColor* and *orthogonalColor* in HSV colour space for the RGB components, and linearly for the alpha component.

To determine the colours to be used, the angles for the surface normal relative to the view position are used. The range  $[0, \pi/2]$  is divided by *colorSteps*. (The two ends of the spectrum are not interpolated in this way and shall use the specified field values). For each of the interpolated ranges, other than the two ends, the midpoint angle is determined and the interpolated colour value is computed using that point.

EXAMPLE Using the default field values for *CartoonVolumeStyle*, the following RGBA colour are computed:

- 1,1,1,1 for angles  $[0, \pi/8)$
- 0.625,0.625,0.625,1 for angles  $[\pi/8, \pi/4)$ ,
- 0.375,0.375,0.375,1 for angles  $[\pi/4, 3\pi/8)$ ,
- 0,0,0,1 for angles  $[3\pi/8, \pi/2]$

[Figure 41.3](#) shows a basic image of a backpack on the left and an image of the backpack using *CartoonVolumeStyle* on the right.



**Figure 41.3 — Default volume style on left and CartoonVolumeStyle on right**

#### 41.4.4 ComposedVolumeStyle

```
ComposedVolumeStyle : X3DComposableVolumeRenderStyleNode {
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFNode [in,out] renderStyle [] [X3DComposableVolumeRenderStyleNode]
}
```

The `ComposedVolumeStyle` node is a rendering style node that allows compositing multiple rendering styles together into a single rendering pass.

EXAMPLE `ComposedVolumeStyle` is used to render a simple volume with both edge and silhouette rendering styles.

The `renderStyle` field contains a list of contributing rendering style nodes or node references that can be applied to the object. The implementation shall apply each rendering style strictly in the order declared starting with the first rendering style in the `renderStyle` field.

[Figure 41.4](#) shows a basic image of a backpack on the left and an image of the backpack using `ComposedVolumeStyle` on the right that combines `EdgeEnhancementVolumeStyle` with `SilhouetteEnhancementVolumeStyle`.

King Draft



**Figure 41.4 — Default volume style on left and ComposedVolumeStyle on right**

### 41.4.5 EdgeEnhancementVolumeStyle

```
EdgeEnhancementVolumeStyle : X3DComposableVolumeRenderStyleNode {
  SFColorRGBA [in,out] edgeColor    0 0 0 1 [0,1]
  SFBool      [in,out] enabled      TRUE
  SFFloat     [in,out] gradientThreshold 0.4 [0,π]
  SFNode      [in,out] metadata     NULL [X3DMetadataObject]
  SFNode      [in,out] surfaceNormals NULL [X3DTexture3DNode]
}
```

The `EdgeEnhancementVolumeStyle` node specifies edge enhancement for the volume rendering style. Enhancement of the basic volume is provided by darkening voxels based on their orientation relative to the view direction. Perpendicular voxels are coloured according to the `edgeColor` while voxels parallel are not changed at all. A threshold can be set where the proportion of how close to parallel the direction needs to be before no colour changes are made.

The `gradientThreshold` field defines the minimum angle (in radians) away from the view direction vector for the surface normal before any enhancement is applied.

The `edgeColor` field defines the colour to be used to highlight the edges.

The `surfaceNormals` field contains a 3D texture with at least three component values. Each voxel in the texture represents the surface normal direction for the corresponding voxel in the base data source. This texture should be identical in dimensions to the source data. If not, the implementation may interpolate or average between adjacent voxels to determine the average normal at the voxel required. If the `surfaceNormals` field is empty, the implementation shall automatically determine the surface normal using algorithmic means.

The final colour is determined by:

$$C_g = C_v \text{ if } (|\mathbf{n} \cdot \mathbf{V}|) \geq \cos(\text{gradientThreshold});$$

$$C_v \times (|\mathbf{n} \cdot \mathbf{V}|) + \text{edgeColor} \times (1 - (|\mathbf{n} \cdot \mathbf{V}|)) \text{ otherwise.}$$

$$O = O$$

Figure 41.5 shows a basic image of a human brain on the left and an image of the human brain using `EdgeEnhancementVolumeStyle` on the right.

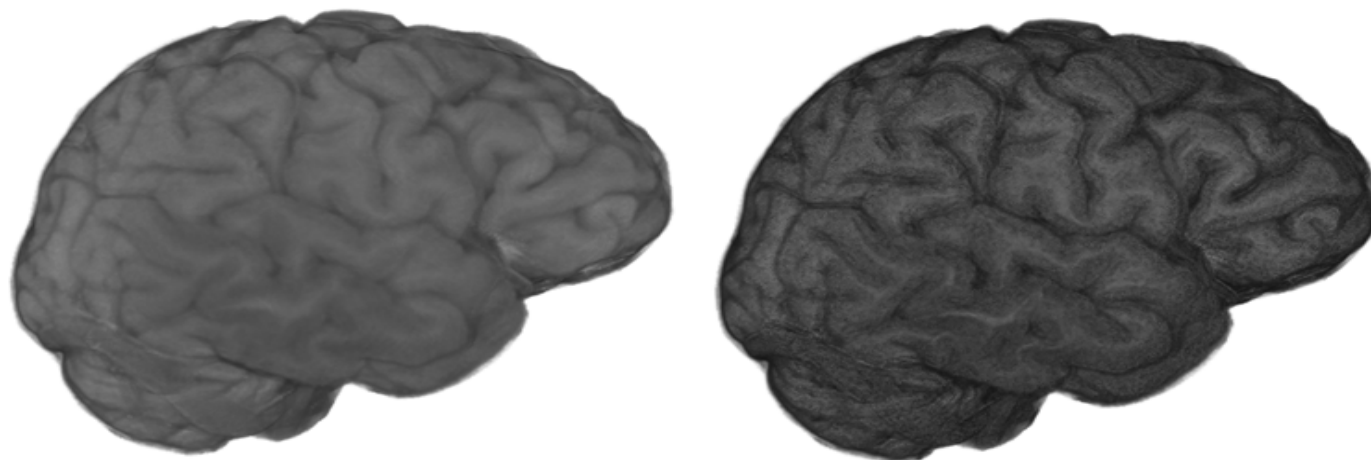


Figure 41.5 — Default volume style on left and `EdgeEnhancementVolumeStyle` on right

#### 41.4.6 IsoSurfaceVolumeData

```

IsoSurfaceVolumeData : X3DVolumeDataNode {
  SFFloat [in,out] contourStepSize 0 (-∞,∞)
  SFVec3f [in,out] dimensions 1 1 1 (0,∞)
  SFBool [in,out] bboxDisplay FALSE
  SFNode [in,out] gradients NULL [X3DTexture3DNode]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  MFNode [in,out] renderStyle [] [X3DVolumeRenderStyleNode]
  SFFloat [in,out] surfaceTolerance 0 [0,∞)
  MFFloat [in,out] surfaceValues [] (-∞,∞)
  SFNode [in,out] voxels NULL [X3DTexture3DNode]
  SFBool [in,out] visible TRUE
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}

```

The `IsoSurfaceVolumeData` node specifies one or more surfaces extracted from a voxel data set. A surface is defined as the boundary between regions in the volume where the voxel values are larger than a given value (the iso value) on one side of the boundary and smaller on the other side and the gradient magnitude is larger than *surfaceTolerance*. The *gradients* field may be used to provide explicit per-voxel gradient direction information for determining surface boundaries rather than having it implicitly calculated by the implementation.

This data representation has one of three possible modes of operation based on the values of the two fields *surfaceValues* and *contourStepSize*.

1. If *surfaceValues* has a single value defined, render the isosurface that corresponds to that value.
2. If the *surfaceValues* field has a single defined *contourStepSize* that is non-zero, also render all isosurfaces that are multiples of that step size from the initial surface value.

EXAMPLE With a surface value of 0.25 and a step size of 0.1, any additional isosurfaces at 0.05, 0.15, 0.35, 0.45, ... shall also be rendered. If *contourStepSize* is left at the default value of zero, only that single isovalue is rendered as a surface.



NOTE The *contourStepSize* is allowed to be negative so that stepping proceeds in a negative direction.

- If *surfaceValues* has more than a single value defined, the *contourStepSize* field is ignored and surfaces corresponding to the listed *surfaceValues* amounts are rendered.

For each isosurface extracted from the data set, a separate render style may be assigned using the *renderStyle* node. The rendering styles are taken from the *renderStyles* field corresponding to the index of the surface value defined. In the case where automatic contours are being extracted using the step size, the explicit surface value shall use the first declared render style. Then render styles are assigned starting from the smallest iso-value. In all cases, if there are insufficient render styles defined for the number of isosurfaces to be rendered, the last style shall be used for all surfaces that do not have an explicit style assigned.

$O_v$  is defined to be 1 for this volume data regardless of the number of components in the provided volume data texture.

[Figure 41.6](#) shows an IsoSurfaceVolume image of a skull using CartoonVolumeStyle.



Figure 41.6 — IsoSurface volume data using CartoonVolumeStyle

#### 41.4.7 OpacityMapVolumeStyle

```
OpacityMapVolumeStyle : X3DComposableVolumeRenderStyleNode {
  SFBool [in,out] enabled      TRUE
  SFNode [in,out] metadata    NULL [X3DMetadataObject]
  SFNode [in,out] transferFunction NULL [X3DTexture2DNode,X3DTexture3DNode]
}
```

The *OpacityMapVolumeStyle* specifies that the associated volumetric data is to be rendered using the opacity mapped to a transfer function texture. This is the default rendering style if no other *X3DComposableVolumeRenderStyleNode* is defined for the volume data.

The *transferFunction* field holds a single texture representation in either two or three dimensions that maps the voxel data values to a specific colour output. If no value is supplied for this field, the default implementation shall generate a 256x1 alpha-only image that blends from completely transparent at pixel 0 to fully opaque at pixel 255. The texture may be any

number of dimensions and any number of components. The voxel values are used as a lookup coordinates into the transfer function texture, where the texel value represents the output colour.

Components are mapped from the voxel data to the transfer function in a component-wise fashion. The first component of the voxel data is an index into the first dimension of the *transferFunction* texture (S). Similarly, T, R, and Q are indices into the second, third, and fourth dimensions of the *transferFunction* texture (see [Table 41.5](#)). If there are more components defined in the voxel data than there dimensions in the transfer function, the extra components are ignored. If there are more dimensions in the transfer function texture than the voxel data, the extra dimensions in the transfer function are ignored (effectively treating the voxel component data as a value of zero for the extra dimension). This mapping locates the texel value in the texture, which is then used as the output for this style.

**Table 41.5 — Transfer function mapping from texture type to texture coordinate**

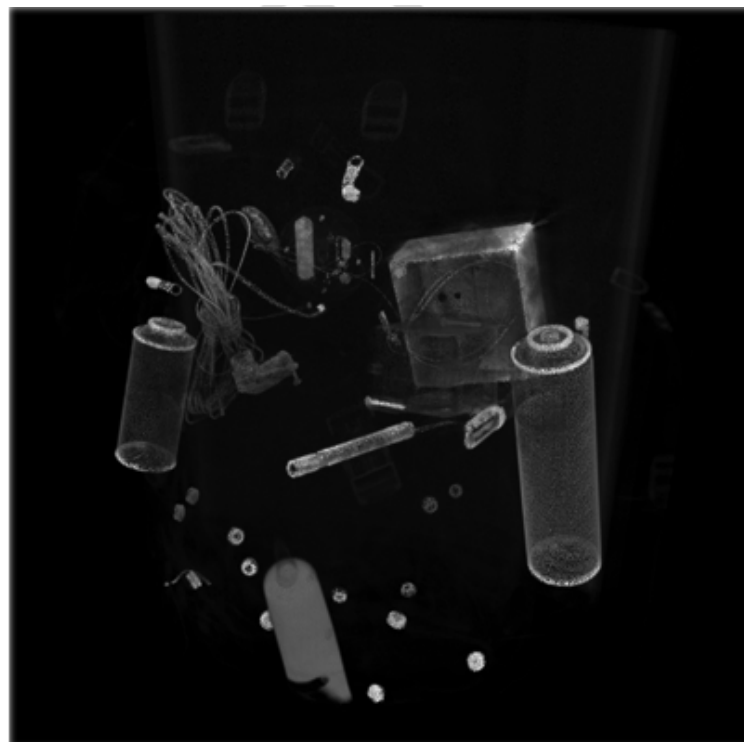
Number of Texture Components	Texture Components	Transfer Function Texture Coordinates
1	Luminance	S ← luminance (L)
2	Luminance Alpha	S, T ← luminance, alpha (LA)
3	RGB	S, T, R ← red, green, blue (RGB)
4	RGBA	S, T, R, Q ← red, green, blue, alpha (RGBA)

The colour value is treated like a normal texture with the colour mapping as defined in [Table 41.6](#).

**Table 41.6 — Transfer function mapping from texture type to output colour**

Texture Components	Red	Green	Blue	Alpha
Luminance (L)	L	L	L	1
Luminance Alpha (LA)	L	L	L	A
RGB	R	G	B	1
RGBA	R	G	B	A

[Figure 41.7](#) shows a basic image of a backpack on the left and an image of the backpack using `OpacityMapVolumeStyle` on the right.



**Figure 41.7 — Default volume style on left and OpacityMapVolumeStyle on right**

#### 41.4.8 ProjectionVolumeStyle

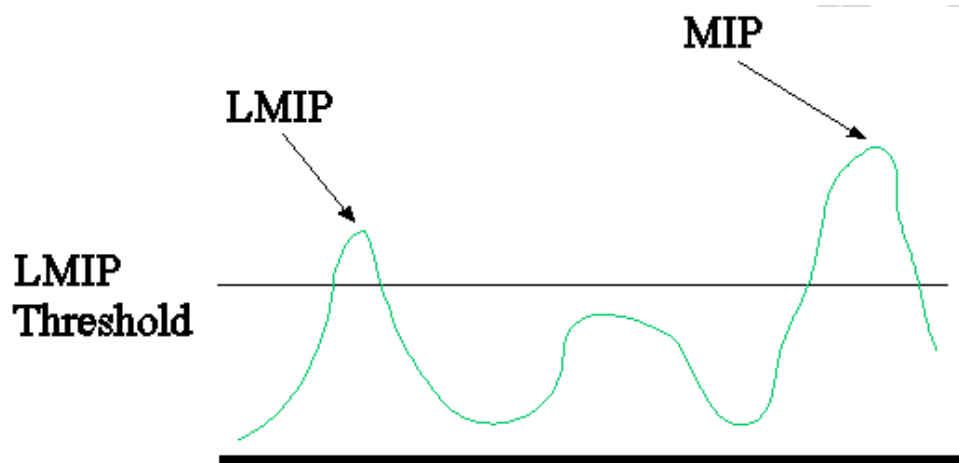
```
ProjectionVolumeStyle : X3DVolumeRenderStyleNode {
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFFloat [in,out] intensityThreshold 0 [0,1]
  SFString [in,put] type "MAX" ["MAX", "MIN", "AVERAGE"]
}
```

The ProjectionVolumeStyle volume style node uses the voxel data directly to generate output colour based on the values of voxel data along the viewing rays from the eye point.

If the value of type is "MAX", The Maximum Intensity Projection (MIP) algorithm is used to generate the output colour. This rendering style also includes the option to use the extended form of Local Maximum Intensity Projection (LMIP, see [\[LMIP\]](#)). The output colour is determined by projecting rays into the voxel data from the viewer location and finding the maximum voxel value found along that ray. If the *intensityThreshold* value is non-zero, rendering will use the first maximum value encountered that exceeds the threshold rather than the maximum found along the entire ray. [Figure 41.8](#) illustrates the difference in rendered value between LMIP and MIP.

ng Draft





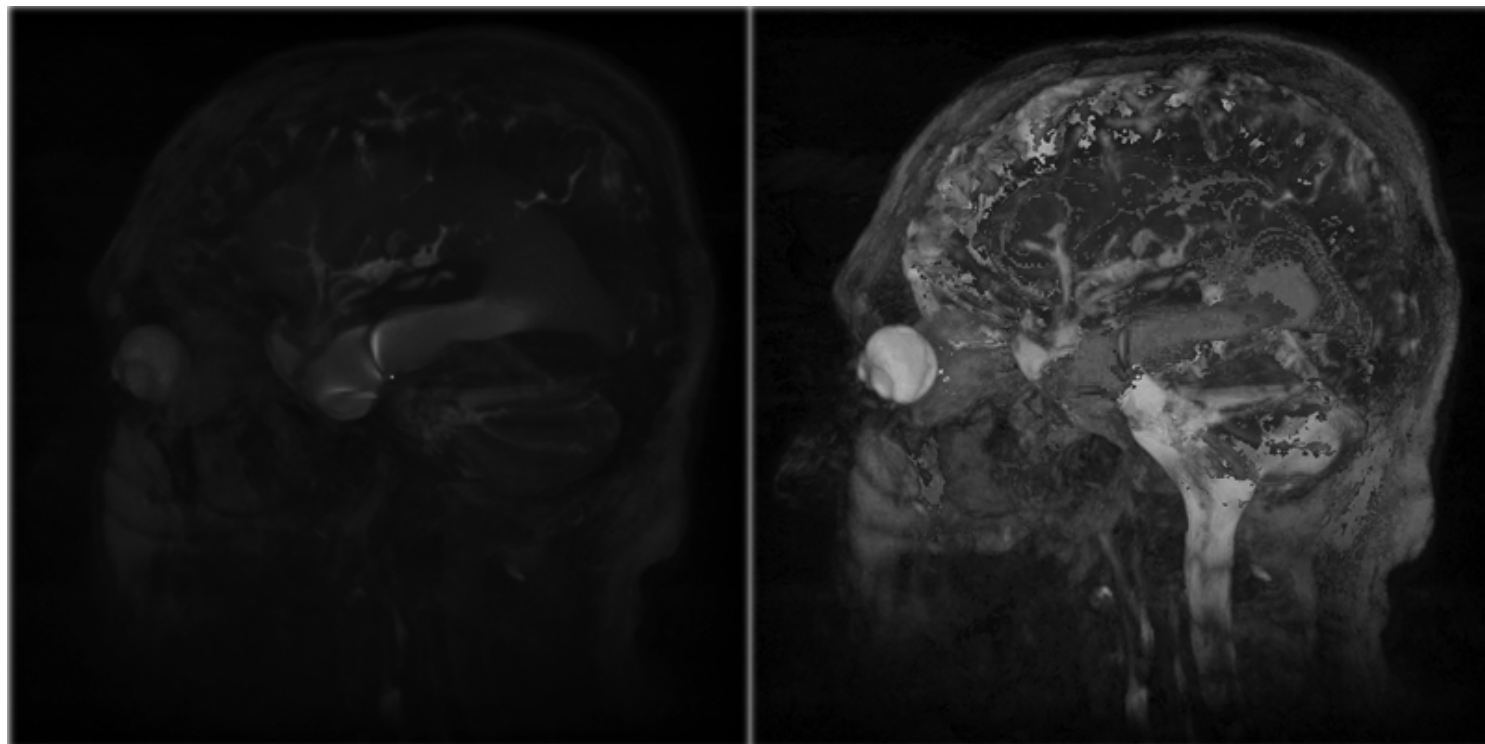
**Figure 41.8 — Illustration of values selected when using MIP or LMIP volume rendering styles**

If the value of *type* is "MIN", Minimum Intensity Projection is used. This works similar to Maximum Intensity Projection with the difference that the minimum voxel value along the ray is used.

If the value of *type* is "AVERAGE", Average Intensity Projection is used. In this case the average value of all voxels along the ray is used as the output colour. The *intensityThreshold* field is ignored. This is a simple approximation of X-Ray.

Since the output of this node is a set of intensity values, all colour components have the same value. The intensity is derived from the average of all colour components of the voxel data (though typical usage will only use single component textures). The Alpha channel is passed through as-is from the underlying data. If there is no alpha channel provided, a default alpha value of 1 is used.

[Figure 41.9](#) shows a basic image of ventricles of the brain on the left and an image of the ventricles of the brain using MIP ProjectionVolumeStyle on the right.



**Figure 41.9 — Default volume style on left and MIP ProjectionVolumeStyle on right**

### 41.4.9 SegmentedVolumeData

```

SegmentedVolumeData : X3DVolumeDataNode {
  SFVec3f [in,out] dimensions      1 1 1 (0,∞)
  SFBool [in,out] bboxDisplay     FALSE
  SFNode [in,out] metadata        NULL [X3DMetadataObject]
  MFNode [in,out] renderStyle     [] [X3DVolumeRenderStyleNode]
  MFBool [in,out] segmentEnabled  []
  SFNode [in,out] segmentIdentifiers NULL [X3DTexture3DNode]
  SFNode [in,out] voxels          NULL [X3DTexture3DNode]
  SFBool [in,out] visible         TRUE
  SFVec3f [] bboxCenter           0 0 0 (-∞,∞)
  SFVec3f [] bboxSize             -1 -1 -1 [0,∞) or -1 -1 -1
}

```

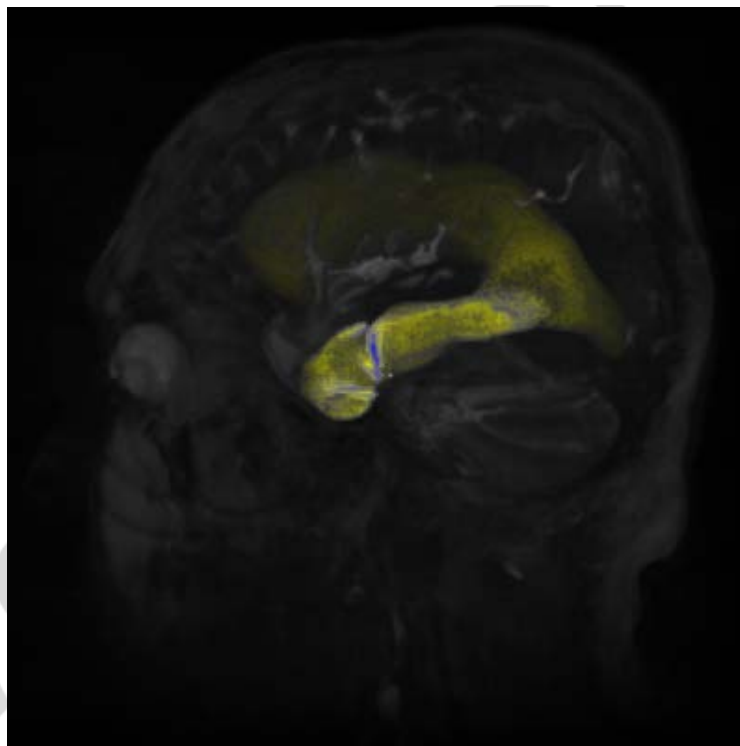
The `SegmentedVolumeData` node specifies a segmented volume data set that allows for representation of different rendering styles for each segment identifier.

The `renderStyle` field optionally describes a particular rendering style to be used. If this field has a non-zero number of values, the defined rendering style is to be applied to the object. If the object is segmented, the index of the segment shall look up the rendering style at the given index in this array of values and apply that style to data described by that segment identifier. If the `renderStyle` field is not specified, the implementation shall use an `OpacityMapVolumeStyle` node (see [41.4.7 OpacityMapVolumeStyle](#)) with default values.

The `voxels` field holds a 3D texture with the data for each voxel. For each voxel, there is a corresponding segment identifier supplied in the `segmentIdentifiers` field, which contains a single component texture. If the `segmentIdentifiers` texture is not identical in size to the main voxels, it shall be ignored. If it contains more than one colour component, only the initial component of the colour shall be used to define the segment identifier.

The `segmentEnabled` field specifies whether a segment is rendered or not. The indices of this array corresponds to the segment identifier. A value at index  $i$  of `FALSE` marks any data with the corresponding segment identifier to not be rendered. If a segment identifier is used that is greater than the length of the array, the value is assumed to be `TRUE`.

[Figure 41.10](#) shows a segmented volume image of ventricles of the brain using both `OpacityMapVolumeStyle` for some segments `ToneMappedVolumeStyle` for the highlighted segments.



**Figure 41.10 — Segmented volume data using `OpacityMapVolumeStyle` and `ToneMappedVolumeStyle`**

#### 41.4.10 `ShadedVolumeStyle`

```
ShadedVolumeStyle : X3DComposableVolumeRenderStyleNode {
  SFBool [in,out] enabled TRUE
  SFBool [in,out] lighting FALSE
  SFNode [in,out] material NULL [X3DMaterialNode]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFBool [in,out] shadows FALSE
  SFNode [in,out] surfaceNormals NULL [X3DTexture3DNode]
  SFString [] phaseFunction "Henyeey-Greenstein" ["Henyeey-Greenstein","NONE",...]
}
```

The `ShadedVolumeStyle` node applies the Blinn-Phong illumination model ([\[BLINN\]](#), [\[PHONG\]](#)) to volume rendering. This is similar to the model used for polygonal surfaces.

Colour and opacity is determined based on whether a value has been specified for the *material* field. If a *material* field value is provided, this voxel is considered to be lit using the lighting equations below. If no *material* field value is provided,  $O_c$  is used as diffuse color in the same lighting equations and the specular and ambient parts are ignored.

The *lighting* field controls whether the rendering should calculate and apply shading effects to the visual output. If lighting is enabled the lighting equation is defined as:

$$C_g = I_{F_{rgb}} \times (1 - f_0) + f_0 \times (C_{E_{rgb}} + \text{SUM}(on_i \times \text{attenuation}_i \times \text{spot}_i \times I_{L_{rgb}} \times (\text{ambient}_i + \text{diffuse}_i + \text{specular}_i)))$$

$$O_g = O_v \times O_m$$

where:

$$\text{attenuation}_i = 1 / \max(a_1 + a_2 \times d_L + a_3 \times d_L^2, 1)$$

$$\text{ambient}_i = I_{ia} \times C_{D_{rgb}} \times C_a$$

$$\text{diffuse}_i = I_i \times C_{D_{\text{rgb}}} \times (\mathbf{N} \cdot \mathbf{L})$$

$$\text{specular}_i = I_i \times C_{S_{\text{rgb}}} \times (\mathbf{N} \cdot ((\mathbf{L} + \mathbf{v}) / |\mathbf{L} + \mathbf{v}|))^{\text{shininess}} \times 128$$

and:

$\cdot$  = modified vector dot product:

if dot product  $< 0$ , then 0.0, otherwise, dot product

$a_1, a_2, a_3$  = light  $i$  *attenuation*

$d_v$  = distance from this voxel to viewer's position, in coordinate system of current fog node

$d_L$  = distance from light to voxel, in light's coordinate system

$f_0$  = fog interpolant, see [Table 17.5](#) for calculation

$I_{F_{\text{rgb}}}$  = currently bound fog's *color*

$I_{L_{\text{rgb}}}$  = light  $i$  *color*

$I_i$  = light  $i$  *intensity*

$I_{ia}$  = light  $i$  *ambientIntensity*

$\mathbf{L}$  = ([PointLight/SpotLight](#)) normalized vector from this voxel to light source  $i$  position

$\mathbf{L}$  = ([DirectionalLight](#)) -direction of light source  $i$

$\mathbf{N}$  = normalized normal vector at this voxel (interpolated from vertex normals specified by the *surfaceNormals* field or automatically calculated).

$O_m$  =  $(1 - X3DMaterialNode \text{ transparency})$  if material specified, 1 otherwise

$C_a$  = [X3DMaterialNode](#) *ambientIntensity* if material specified, 0 otherwise

$C_{D_{\text{rgb}}}$  = diffuse colour, from a node derived from *X3DMaterialNode* if specified,  $O_c$  otherwise

$C_{E_{\text{rgb}}}$  = *X3DMaterialNode emissiveColor* if material specified, RGB(0,0,0) otherwise

$C_{S_{\text{rgb}}}$  = *X3DMaterialNode specularColor* if material specified, RGB(0,0,0) otherwise

$on_i = 1$ , if light source  $i$  affects this voxel,

0, if light source  $i$  does not affect this voxel.

The following conditions indicate that light source  $i$  does not affect this voxel:

1. if the voxel is farther away than *radius* for *PointLight* or *SpotLight*;
2. if the volume is outside the enclosing [X3DGroupingNode](#) and/or if the *on* field is `FALSE`;
3. if the *lighting* field of this volume is `FALSE`.

*shininess* = *X3DMaterialNode shininess* if material specified, 0 otherwise

*spotAngle* =  $\arccosine(-\mathbf{L} \cdot \mathbf{spotDir}_i)$

*spot<sub>BW</sub>* = *SpotLight*  $i$  *beamWidth*

*spot<sub>CO</sub>* = *SpotLight*  $i$  *cutOffAngle*

*spot<sub>i</sub>* = spotlight factor, see [Table 17.4](#) for calculation

*spotDir<sub>i</sub>* = normalized *SpotLight*  $i$  *direction*

SUM: sum over all light sources  $i$

$\mathbf{v}$  = normalized vector from the voxel to viewer's position

If the *lighting* field is `FALSE`, the diffuse color is used without any shading effects.

$$C_g = C_{D_{\text{rgb}}}$$

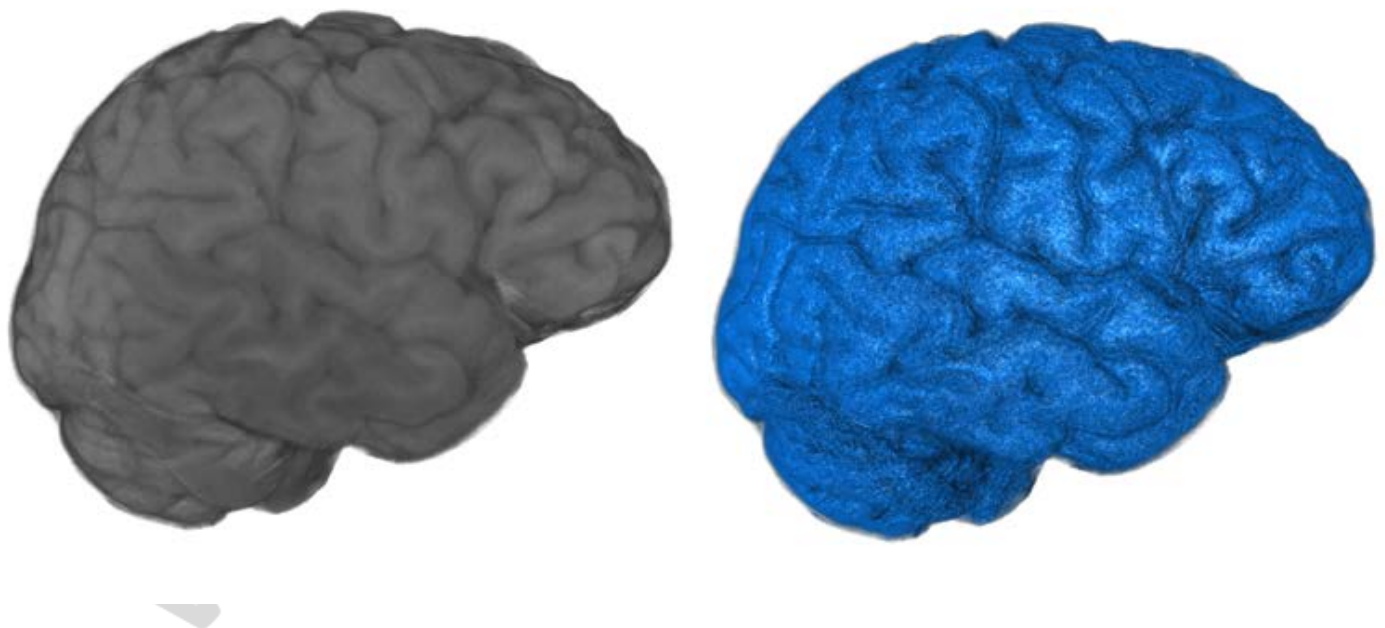
$$O_g = O_v \times O_m$$

The *surfaceNormals* field contains a 3D texture with at least three component values. Each voxel in the texture represents the surface normal direction for the corresponding voxel in the base data source to be used in the lighting equation. This texture should be identical in dimensions to the source data. If not, the implementation may interpolate or average between adjacent voxels to determine the average normal at the voxel required. If this field is empty, the implementation shall automatically determine the surface normal using algorithmic means.

The *shadows* field controls whether the rendering should calculate and apply shadows to the visual output (using global illumination model). A value of `FALSE` requires that no shadowing be applied. A value of `TRUE` requires that shadows be applied to the object. If the *lighting* field is set to `FALSE`, this field shall be ignored and no shadows generated.

The *phaseFunction* field is used to define the scattering model for use in an implementation using global illumination. The name defines the model type, based on standard algorithms externally defined to this specification. The valid values are "NONE" (which means no scattering) and "Henyey-Greenstein" which is the Henyey-Greenstein phase function defined in [HENYEY]. Browsers may choose to support other values. If a value is specified that is not supported by the browser, "Henyey-Greenstein" shall be used.

[Figure 41.11](#) shows a basic image of a human brain on the left and an image of the human brain using `ShadedVolumeStyle` on the right.



**Figure 41.11 — Default volume style on left and `ShadedVolumeStyle` on right**

#### 41.4.11 `SilhouetteEnhancementVolumeStyle`

```
SilhouetteEnhancementVolumeStyle : X3DComposableVolumeRenderStyleNode {
  SFBool [in,out] enabled          TRUE
  SFNode  [in,out] metadata        NULL [X3DMetadataObject]
  SFFloat [in,out] silhouetteBoundaryOpacity 0 [0,1]
  SFFloat [in,out] silhouetteRetainedOpacity 1 [0,1]
  SFFloat [in,out] silhouetteSharpness    0.5 [0,∞)
  SFNode  [in,out] surfaceNormals      NULL [X3DTexture3DNode]
}
```

The `SilhouetteEnhancementVolumeStyle` specifies that the associated volumetric data shall be rendered with silhouette enhancement. Enhancement of the basic volume is provided by darkening voxels based on their orientation relative to the view direction. This orientation is

determined by the *surfaceNormals* value corresponding to each voxel. Perpendicular voxels are coloured according to the *edgeColor* while parallel voxels are not changed at all. A threshold can be set where the proportion of how close to perpendicular the direction shall be before the values are made more opaque:

$$O_g = O_v \times (k_{sc} + k_{ss}(1 - |\mathbf{n} \cdot \mathbf{V}|) ^ k_{se})$$

where

- $O_g$  is the computed opacity of the voxel
- $O_v$  is the original opacity of the voxel
- $\mathbf{n}$  is the surface normal
- $\mathbf{V}$  is the view vector
- $k_{sc}$  controls the scaling of non-silhouette regions (*silhouetteRetainedOpacity*)
- $k_{ss}$  is the amount of the silhouette enhancement to use (*silhouetteBoundaryOpacity*)
- $k_{se}$  is a power function to control the sharpness of the silhouette. (*silhouetteSharpness*)

The *surfaceNormals* field contains a 3D texture with at least three component values. Each voxel in the texture represents the surface normal direction for the corresponding voxel in the base data source. This texture should be identical in dimensions to the source data. If not, the implementation may interpolate or average between adjacent voxels to determine the average normal at the voxel required. If the *surfaceNormals* field is empty, the implementation shall automatically determine the surface normal using algorithmic means.

[Figure 41.12](#) shows a basic image of a skull on the left and an image of the skull using *SilhouetteEnhancementVolumeStyle* on the right.

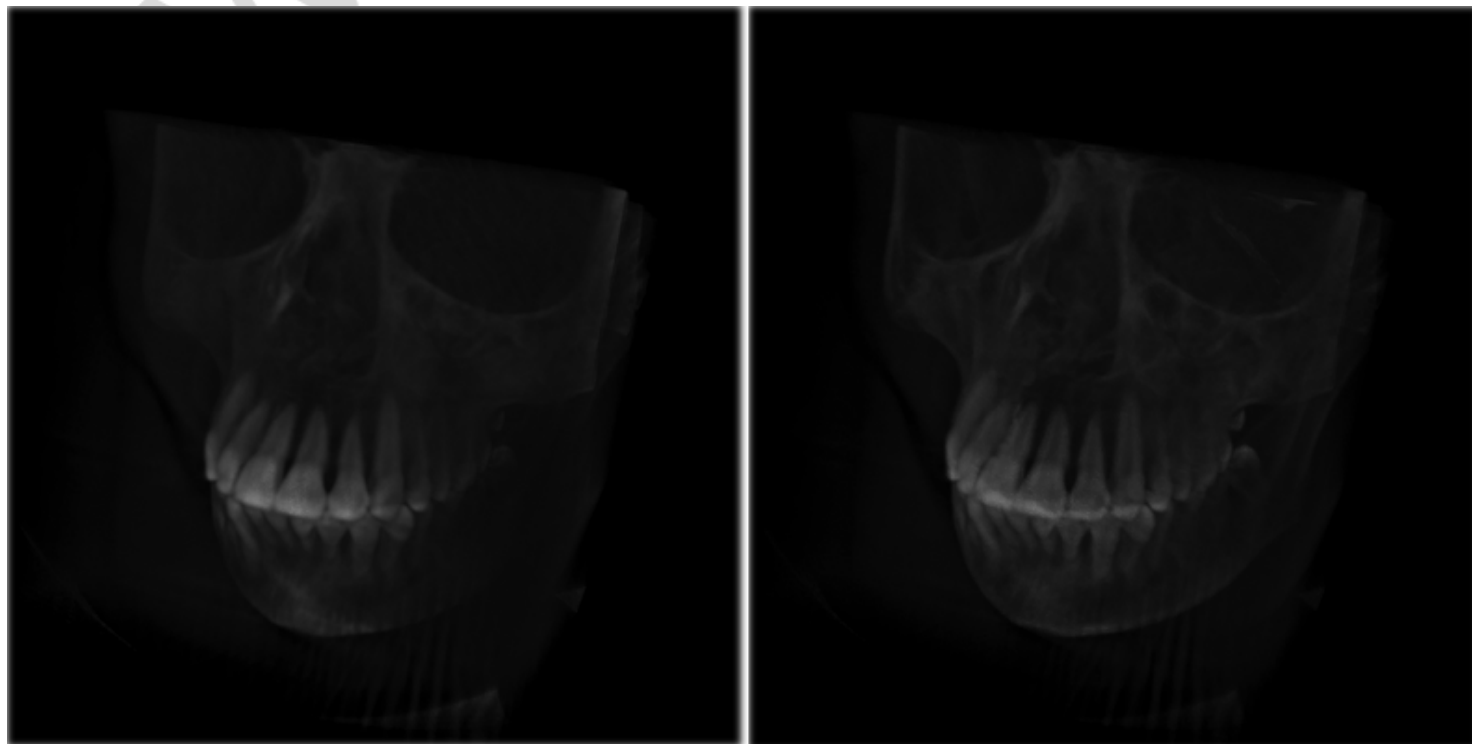


Figure 41.12 — Default volume style on left and *SilhouetteEnhancementVolumeStyle* on right

#### 41.4.12 *ToneMappedVolumeStyle*



```

ToneMappedVolumeStyle : X3DComposableVolumeRenderStyleNode {
  SFColorRGBA [in,out] coolColor 0 0 1 0 [0,1]
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFNode [in,out] surfaceNormals NULL [X3DTexture3DNode]
  SFColorRGBA [in,out] warmColor 1 1 0 0 [0,1]
}

```

The `ToneMappedVolumeStyle` node specifies that the associated volumetric data is to be rendered using the Gooch shading model of two-toned warm/cool colouring (see [\[GOOCH1\]](#), [\[GOOCH2\]](#)). Two colours are defined, a warm colour and a cool colour. The renderer shades between them based on the orientation of the voxel relative to the user. This is not the same as the basic isosurface shading and lighting. The following colour formula is used:

$$cc = (1 + \mathbf{L}_i \cdot \mathbf{n}) \times 0.5$$

$$C_g = \sum_{(\text{all } i)} (cc \times \text{warmColor} + (1 - cc) \times \text{coolColor})$$

where

- $\mathbf{L}_i$  is the vector to light source  $i$
- $\mathbf{n}$  is the surface normal
- $C_g$  is the resulting colour that is to be used to represent the voxel

The `warmColor` and `coolColor` fields specify the two colours to be used at the limits of the spectrum. The `warmColor` field is used for surfaces facing towards the light, while the `coolColor` is used for surfaces facing away from the light direction.

The `surfaceNormals` field contains a 3D texture with at least three component values. Each voxel in the texture represents the surface normal direction for the corresponding voxel in the base data source. This texture should be identical in dimensions to the source data. If not, the implementation may interpolate or average between adjacent voxels to determine the average normal at the voxel required. If the `surfaceNormals` field is empty, the implementation shall automatically determine the surface normal using algorithmic means.

The final output colour is determined by combining the interpolated colour value  $C_g$  with the opacity of the corresponding voxel. Colour components of the voxel are ignored.

[Figure 41.13](#) shows a basic image of ventricles of the brain on the left and an image of the ventricles of the brain using `ToneMappedVolumeStyle` on the right.

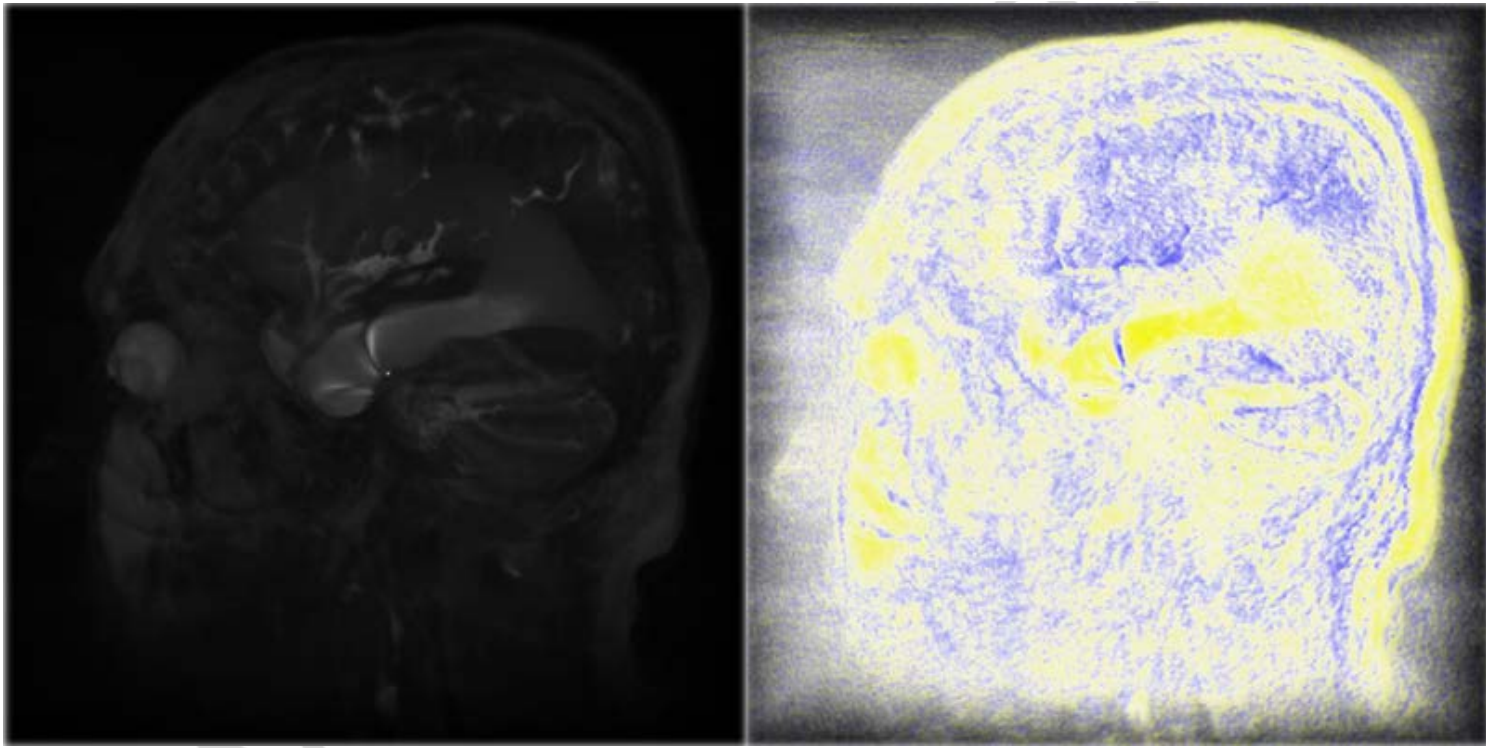


Figure 41.13 — Default volume style on left and ToneMappedVolumeStyle on right

### 41.4.13 VolumeData

```

VolumeData : X3DVolumeDataNode {
  SFVec3f [in,out] dimensions 1 1 1 (0,∞)
  SFBool [in,out] bboxDisplay FALSE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFNode [in,out] renderStyle NULL [X3DVolumeRenderStyleNode]
  SFNode [in,out] voxels NULL [X3DTexture3DNode]
  SFBool [in,out] visible TRUE
  SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
  SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}

```

The `VolumeData` node specifies a simple non-segmented volume data set that uses a single rendering style node for the complete volume.

The `renderStyle` field allows the user to specify a single specific rendering technique to be used on this volumetric object. If the `renderStyle` field is not specified, the implementation shall use an `OpacityMapVolumeStyle` node (see [41.4.7 OpacityMapVolumeStyle](#)) with default values.

The `voxels` field provides the raw voxel information to be used by the corresponding rendering styles. The value is any `X3DTexture3DNode` type and may have any number of colour components defined. The specific interpretation for the values at each voxel shall be defined by the value of the `renderStyle` field.

[Figure 41.14](#) shows a basic volume image of a backpack using the default rendering style.





Figure 41.14 — Volume data using default volume style

## 41.5 Support levels

The Volume Rendering component provides three levels of support as specified in [Table 41.7](#).

Table 41.7 — Volume rendering component support levels

Level	Prerequisites	Nodes/Features	Support
1	Core 1 Grouping 1 Shape 1 Rendering 1		
		<i>X3DComposableVolumeRenderStyleNode</i>	n/a
		<i>X3DVolumeRenderStyleNode</i>	n/a
		<i>X3DVolumeNode</i>	n/a
		OpacityMapVolumeStyle	Only 2D texture transfer functions need be supported. All other fields fully supported.
		VolumeData	All fields fully supported.
2	Core 1 Grouping 1 Shape 1		

	Rendering 1		
		All Level 1 nodes	All fields fully supported.
		BoundaryEnhancementVolumeStyle	All fields fully supported.
		ComposedVolumeStyle	<i>ordered</i> field is always treated as <code>FALSE</code> . All other fields fully supported.
		EdgeEnhancementVolumeStyle	All fields fully supported.
		IsoSurfaceVolumeData	All fields fully supported.
		OpacityMapVolumeStyle	All fields fully supported. 3D transfer functions shall be supported.
		ProjectionVolumeStyle	All fields fully supported
		SegmentedVolumeData	All fields fully supported.
		SilhouetteEnhancementVolumeStyle	All fields fully supported.
		ToneMappedVolumeStyle	All fields fully supported.
<b>3</b>	Core 1 Grouping 1 Shape 1 Rendering 1		
		All Level 2 nodes	All fields fully supported.
		BlendedVolumeStyle	All fields fully supported.
		CartoonVolumeStyle	All fields fully supported.
		CompositeVolumeStyle ComposedVolumeStyle	All fields fully supported.
		ShadedVolumeStyle	All fields fully supported except shadows. Shadows

			supported with at least Phong shading.
4	Core 1 Grouping 1 Shape 1 Rendering 1		
		All Level 3 nodes	All fields fully supported.
		ShadedVolumeStyle	All fields fully supported with at least Phong shading and Henyey-Greenstein phase function. Shadows fully supported.



Draft



# Extensible 3D (X3D)

## Part 1: Architecture and base components

### 21 Key device sensor component



#### 21.1 Introduction

##### 21.1.1 Name

The name of this component is "KeyDeviceSensor". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.3.4 Component statement](#)).

##### 21.1.2 Overview

This clause describes the Key Device Sensor component of this part of ISO/IEC 19775. This includes how individual keystrokes and a series of keystrokes are inserted into an X3D world. [Table 21.1](#) provides links to the major topics in this clause.

**Table 21.1 — Topics**

- [21.1 Introduction](#)
  - [21.1.1 Name](#)
  - [21.1.2 Overview](#)
- [21.2 Concepts](#)
- [21.3 Abstract types](#)
  - [21.3.1 X3DKeyDeviceSensorNode](#)
- [21.4 Node reference](#)
  - [21.4.1 KeySensor](#)
  - [21.4.2 StringSensor](#)
- [21.5 Support levels](#)
- [Table 21.1 — Topics](#)
- [Table 21.2 — Action key values](#)
- [Table 21.3 — Key Device Sensor component support levels](#)

#### 21.2 Concepts



The following node types are keydevice sensors:

- [KeySensor](#)
- [StringSensor](#)

KeySensors generate an event whenever the state of a key associated with the physical key device changes while the KeySensor is active. The identification of the key whose state has changed is returned by the event.

StringSensors generate an event whenever the termination string specified for the StringSensor is identified. The UTF-8 characters preceding the termination string are returned by the event. StringSensors also generate interim events whenever the string under construction changes. This allows prompting of the string during construction.

One keyboard-style key device is assumed to be available either as a physical device or through emulation whenever the sensor component is supported. For a key device to generate input to a key device sensor, the key device shall be active. Key devices are active when:

- The X3D world has focus in the supporting user interface; and
- The key device sensor has its *isActive* field set to `TRUE`.

The *isActive* event generated by a change of the state of the *isActive* field can be used for prompting.

A key device sensor is enabled when its *enabled* field is set to `TRUE`. This causes the *isActive* field of the keydevice sensor to be set to `TRUE`. Also, any other key device sensor which may be active will be sent an *enabled* event with value `FALSE`. Only one key device sensor may be active at a time

## 21.3 Abstract types

### 21.3.1 X3DKeyDeviceSensorNode

```
X3DKeyDeviceSensorNode : X3DSensorNode {
  SFBool   [in,out] enabled  TRUE
  SFNode   [in,out] metadata NULL [X3DMetadataObject]
  SFBool   [out]    isActive
}
```

This abstract node type is the base type for all sensor node types that operate using key devices.

## 21.4 Node reference

### 21.4.1 KeySensor

```
KeySensor : X3DKeyDeviceSensorNode {
  SFBool   [in,out] enabled      TRUE
  SFNode   [in,out] metadata    NULL [X3DMetadataObject]
  SFInt32  [out]    actionKeyPress
  SFInt32  [out]    actionKeyRelease
  SFBool   [out]    altKey
  SFBool   [out]    controlKey
  SFBool   [out]    isActive
  SFString [out]    keyPress
```

```

    SFString [out]    keyRelease
    SFBool    [out]    shiftKey
}

```

A `KeySensor` node generates events when the user presses keys on the keyboard. A `KeySensor` node can be enabled or disabled by sending it an *enabled* event with a value of `TRUE` or `FALSE`. If the `KeySensor` node is disabled, it does not track keyboard input or send events.

*keyPress* and *keyRelease* events are generated as keys which produce characters are pressed and released on the keyboard. The value of these events is a string of length 1 containing the single UTF-8 character associated with the key pressed. The set of UTF-8 characters that can be generated will vary between different keyboards and different implementations.

*actionKeyPress* and *actionKeyRelease* events are generated as 'action' keys are pressed and released on the keyboard. The value of these events are in [Table 21.2](#):

**Table 21.2 — Action key values**

KEY	VALUE	KEY	VALUE	KEY	VALUE
HOME	13	END	14	PGUP	15
PGDN	16	UP	17	DOWN	18
LEFT	19	RIGHT	20	F1-F12	1-12

*shiftKey*, *controlKey*, and *altKey* events are generated as each of the shift, control, and alt keys on the keyboard is respectively pressed and released. Their value is `TRUE` when the key is pressed and `FALSE` when the key is released.

When a key is pressed, the `KeySensor` sends an *isActive* event with value `TRUE`. Once the key is released, the `KeySensor` sends an *isActive* event with value `FALSE`.

The `KeySensor` is not affected by its position in the transformation hierarchy.

Recommended default key mappings for navigation are described in [Annex G Recommended navigation behaviours](#).

## 21.4.2 StringSensor

```

StringSensor : X3DKeyDeviceSensorNode {
  SFBool    [in,out] deletionAllowed TRUE
  SFBool    [in,out] enabled        TRUE
  SFNode    [in,out] metadata       NULL [X3DMetadataObject]
  SFString  [out]    enteredText
  SFString  [out]    finalText
  SFBool    [out]    isActive
}

```

A `StringSensor` node generates events as the user presses keys on the keyboard. A `StringSensor` node can be enabled or disabled by sending it an *enabled* event with a value of `TRUE` or `FALSE`. If the `StringSensor` node is disabled, it does not track keyboard input or send events.

*enteredText* events are generated as keys which produce characters are pressed on the keyboard. The value of this event is the UTF-8 string entered including the latest character struck. The set of UTF-8 characters that can be generated will vary between different keyboards and different implementations.

If a *deletionAllowed* has value `TRUE`, the previously entered character in the *enteredText* is removed when the browser-recognized value for deleting the preceding character of a string is entered. Typically, this value is defined by the local operating system. If *deletionAllowed* has value `FALSE`, characters may only be added to the string; deletion of characters shall not be allowed. Should the browser-recognized value for deleting the preceding character is entered, it shall be ignored.

The *finalText* event is generated whenever the browser-recognized value for terminating a string is entered. Typically, this value is defined by the local operating system. When this recognition occurs, the *finalText* field generates an event with value equal to that of *enteredText*. After the *finalText* field event has been generated, the *enteredText* field is set to the empty string but no event is generated.

When the user begins typing, the `StringSensor` sends an *isActive* event with value `TRUE`. When the string is terminated, the `StringSensor` sends an *isActive* event with value `FALSE`.

The `StringSensor` is not affected by its position in the transformation hierarchy.

## 21.5 Support levels

The Key Device Sensor component provides 2 levels of support as specified in [Table 21.3](#).

**Table 21.3 — Key device sensor component support levels**

Level	Prerequisites	Nodes/Features	Support
<b>1</b>	Core 1		
		<i>X3DKeyDeviceSensorNode</i> (abstract)	n/a
		<code>KeySensor</code>	All fields fully supported.
<b>2</b>	Core 1		
		All Level 1 Key Device Sensor nodes	All fields as supported in Level 1.
		<code>StringSensor</code>	All fields fully supported.







## Extensible 3D (X3D) Part 1: Architecture and base components

# 42 Projective Texture Mapping (PTM) Component



## 42.1 Introduction

### 42.1.1 Name

The name of this component is "ProjectiveTextureMapping". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

### 42.1.2 Overview

This clause describes the projective texture mapping component of this part of ISO/IEC 19775. This includes how projective texture maps are specified and how they are positioned on the 3D scene. [Table 42.1](#) provides links to the major topics in this clause.

**Table 42.1 — Topics**

- [42.1 Introduction](#)
  - [42.1.1 Name](#)
  - [42.1.2 Overview](#)
- [42.2 Concepts](#)
  - [42.2.1 Overview](#)
  - [42.2.2 Projective texture mapping concepts](#)
  - [42.2.3 Texture map image formats](#)
- [42.3 Abstract types](#)
  - [42.3.1 X3DTextureProjectorNode](#)
- [42.4 Node reference](#)
  - [42.4.1 TextureProjectorParallel](#)
  - [42.4.2 TextureProjectorPerspective](#)
- [42.5 Support levels](#)
- [Figure 42.1 — Concept of projective texture mapping](#)
- [Figure 42.2 — Application of projective texture mapping for reconstructing a 3D terrain](#)

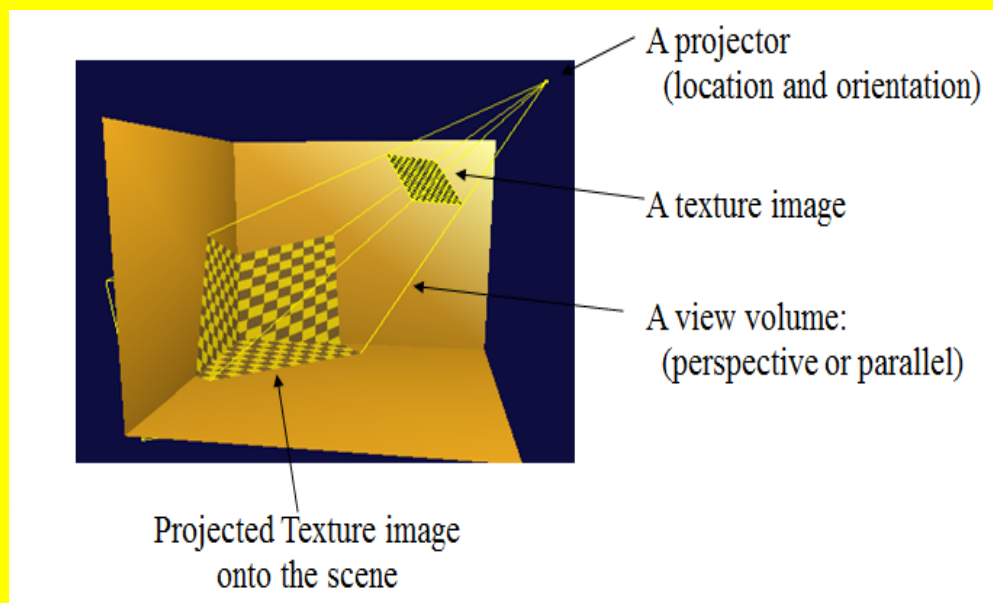
[Figure 42.3 — Application of projective texture mapping for reconstructing an endoscope 3D model](#)

- [Figure 42.4 — Description of 3D perspective texture mapping](#)
- [Figure 42.5 — Description of 3D parallel texture mapping](#)
- [Table 42.1 — Topics](#)
- [Table 42.2 — Support levels](#)

## 42.2 Concepts

### 42.2.1 Overview

This component provides additional texturing extensions to the basic capabilities defined in X3D. Generally, 2D and 3D texture mapping (see Clauses 18 and 33) has been used to enhance the quality of an image generated with a camera or to speed up the generation of an image with respect to a given scene including several geometric models. However, there are some constraints for mapping region and shape of textures over objects. As an extension of texture mapping, the texture image can be projected onto a 3D scene within the projection volume which is constructed from projection parameters such as a projection point, a projection direction and a projection aspect ratio. Figure 1 shows an example screen shot by applying a projective texture mapping to a 3D virtual scene. The texture mapping of this type is called a *projective texture mapping*.



**Figure 42.1 — Concept of projective texture mapping**

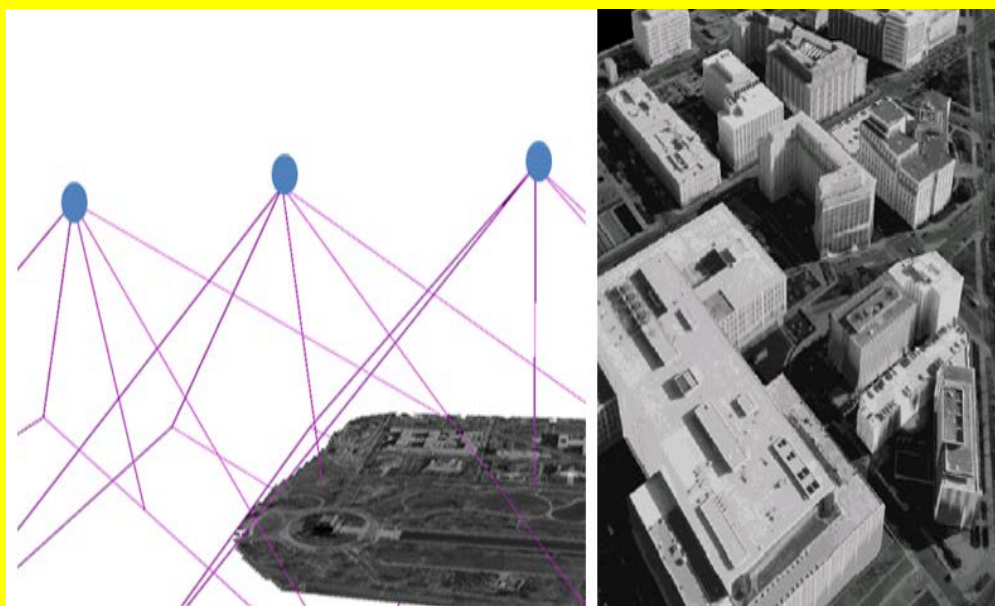
This projective texture mapping is essential for enhanced rendering effects in a 3D scene such as rendering of beam projection images, visualization of a terrain surface in GIS applications, and medical visualization.

### 42.2.2 Projective texture mapping concepts

The projective texture mapping allows a texture image to be projected onto a 3D virtual scene inside the projection volume visible from a specific position called a projection point. The projection volume is determined by projection parameters depending on two types: parallel and perspective projections. In a parallel projection, the parallel volume will become a parallel volume and in a perspective projection, the projection volume will become the shape of the frustum.

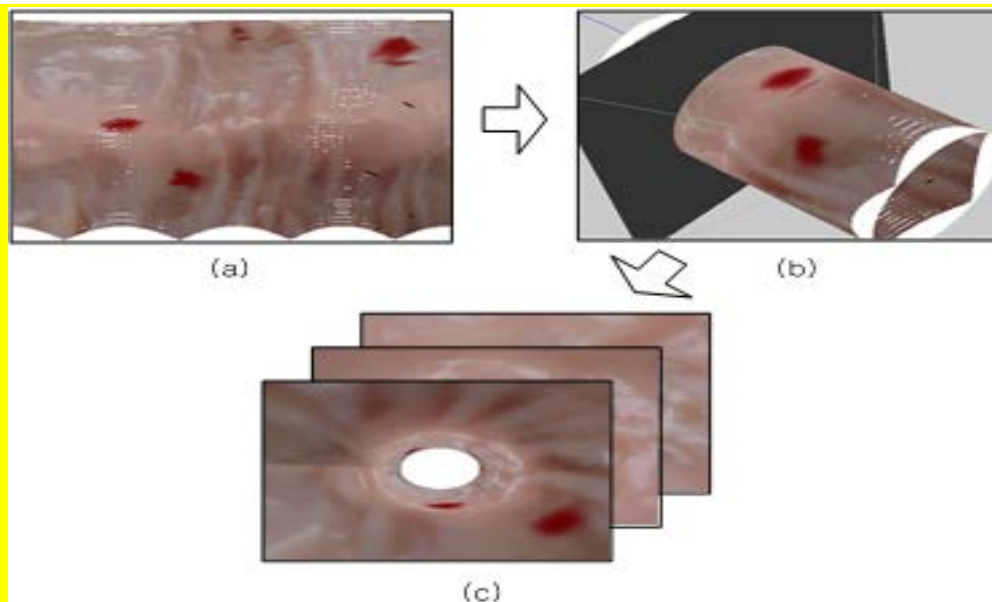
The parallel projection can be distinguished into orthographic and oblique according to the defined projection direction. If all the projection lines are orthogonal to the projection plane, the projection is called orthographic projection. All the projection lines in oblique projection will intersect the projection plane at an oblique angle. In order to describe the types of parallel projection, the projection point and projection direction will be given as a point and a vector, respectively. The projection volume in parallel projection can be defined as a parallelepiped.

The perspective projection can be defined as a field of view angle from a projection point, an aspect ratio of width and height, near and far planes. In a projective texture mapping, generally, single texture as well as several texture images can be projected onto a scene in a 3D virtual world. Furthermore, multiple projective texture mapping can be performed over a common scene with specific objectives such as photogrammetry or reconstruction of endoscope images. As shown Figure 2, assume that several images are provided, each of which is taken with a different camera. Construction of a terrain surface from those images can be performed by displaying overlapping images obtained after applying several projective textures to the surface model.



**Figure 42.2 — Application of perspective texture mapping for reconstructing a 3D terrain**

Figure 3 describes an example for reconstructing endoscope images over a cylinder by applying projective texture mapping. In a similar manner, each image is captured from an endoscope with perspective view information inside human body.



**Figure 42.3 — Application of perspective texture mapping for reconstructing an endoscope 3D model.**

### 42.2.3 Image formats for projective texture mapping

Node types specifying images for projective texture mapping may supply data with a number of color components between one and four. The valid types and interpretations of 3D textures are identical to that for 2D textures. The definition of texture formats is defined in [18.2.1 Texture map formats](#).

## 42.3 Abstract types

### 42.3.1 *X3DTextureProjectorNode*

```

X3DTextureProjectorNode : X3DChildNode {
  SFString [in,out] description ""
  SFVec3f [in,out] direction 0 0 1 (-∞,∞)
  SFFloat [in,out] farDistance 10
  SFBool [in,out] global TRUE
  SFVec3f [in,out] location 0 0 0 (-∞,∞)
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFFloat [in,out] nearDistance 1
  SFBool [in,out] on TRUE
  SFNode [in,out] texture NULL [X3DTexture2DNode]
  SFFloat [out] aspectRatio (0,∞)
}

```

This abstract node type is the base type for all node types that specify projective texture mapping.

The *description* field of this node tells the name of the projector, and makes the division of different projectors possible.

The *location* shows the position of the projector, and this implies projection point.

The *direction* is the way the projector is heading, and this implies to projection direction.

The *aspectRatio* is the aspect ratio of the width and length which refers to projection

spect ratio.

The *nearDistance* and *farDistance* is the minimum and maximum distance that is shown on the screen, respectively.

Each projective texture mapping type defines a *global* field that determines whether the projective texture mapping is global or scoped. Global projective texture mapping performs the texture mappings for all objects that fall within their volume of projective texture mapping influence. Scoped projective texture mapping only performs the texture mappings for objects that are in the same transformation hierarchy as the projective texture mapping; *i.e.*, only the children and descendants of its enclosing parent group are illuminated.

The *on* field specifies whether the projective texture mapping is performed or not. If *on* is `TRUE`, the projective texture mapping is performed for geometry objects in the scene. If *on* is `FALSE`, the projective texture mapping is not performed for any geometry in the scene.

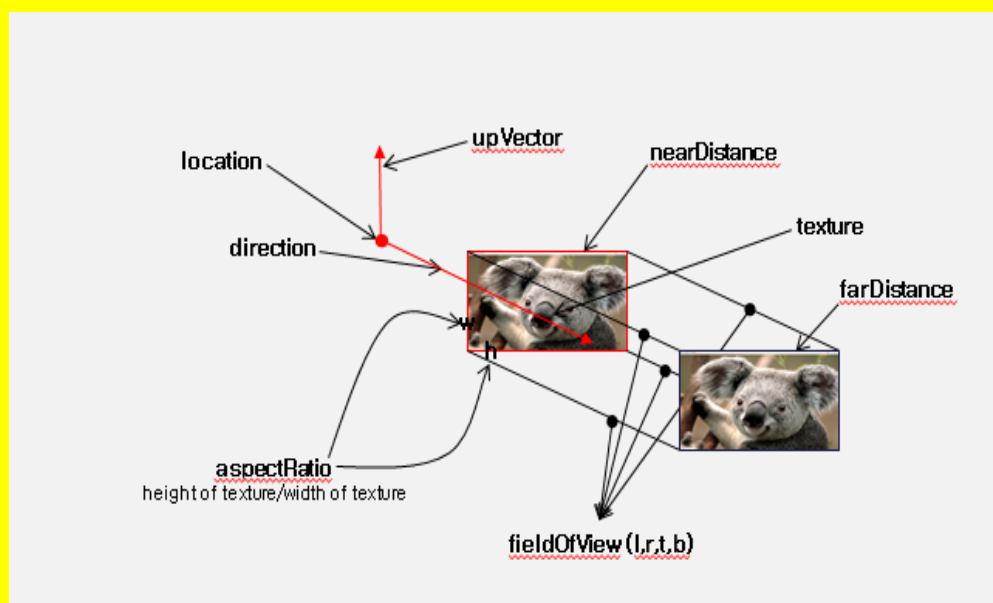
See [18 Texturing component](#) for a general description of the [X3DTexture2DNode](#) abstract type and interpretation of rendering for 2D images.

## 42.4 Node reference

### 42.4.1 TextureProjectorParallel

```
TextureProjectorParallel : X3DTextureProjectorNode {
  SFString [in,out] description ""
  SFVec3f [in,out] direction 0 0 1 (-∞,∞)
  SFFloat [in,out] farDistance 10
  SFVec4f [in,out] fieldOfView -1 -1 1 1 (-∞,∞)
  SFBool [in,out] global TRUE
  SFVec3f [in,out] location 0 0 0 (-∞,∞)
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFFloat [in,out] nearDistance 1
  SFBool [in,out] on TRUE
  SFNode [in,out] texture NULL [X3DTexture2DNode]
  SFFloat [out] aspectRatio (0,∞)
}
```

Parallel texture mapping is shown in the following figure.



## Figure 42.5 — Description of 3D parallel texture mapping

The *description* field of this node tells the name of the perspective projector, and makes the division of different perspective projectors possible.

The *location* shows the position of the perspective projector, and this implies perspective projection point.

The *direction* is the way the perspective projector is heading, and this implies to perspective projection direction.

The *fieldOfView* is the extent of the observable world that is parallelly seen on the display at any given moment. This value may change depending on the aspect ratio of the rendering resolution. The default value of this field is (-1 -1 1 1).

The *aspectRatio* is the aspect ratio of the width and length which refers to perspective projection spect ratio.

The *nearDistance* and *farDistance* is the minimum and maximum distance that is shown on the screen, respectively.

*global*, *on* and *texture* fields are the same as illustrated in the Abstract node.

TODO: convert to ClassicVRML syntax

```
<X3D profile="Interactive" version="3.3">
<Scene>

<TextureProjectorPerspective
  description='pt1' location='3 3 3' direction='-1 0 -1'
  fieldOfView='0.26' nearDistance='1' farDistance='10'
  upVector='0 1 0' global='true' on='true'>

  <ImageTexture url='C:/image/apple.jpg' repeatS='false' repeatT='false'/>
</TextureProjectorPerspective>

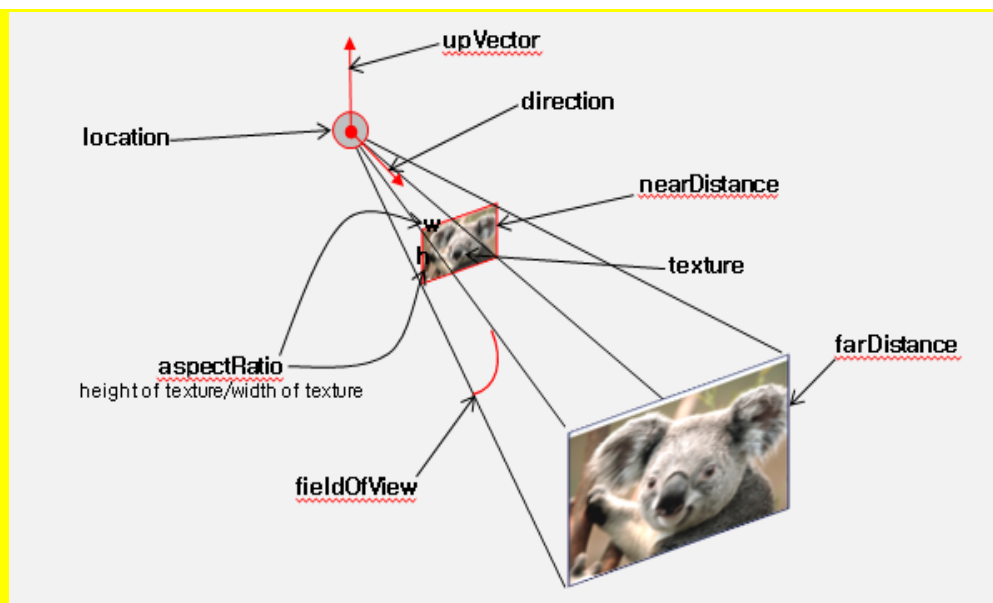
<Shape>
  <Appearance>
    <Material diffuseColor='0.5 0.5 0.5'/>
  </Appearance>

  <IndexedFaceSet solid='false' coordIndex="3 2 1 0 -1, 4 5 2 3 -1, 5 6 1 2 -1">
    <Coordinate point="1 0 1, -1 0 1, -1 0 -1, 1 0 -1, 1 1 -1, -1 1 -1, -1 1 1"/>
  </IndexedFaceSet>
</Shape>
</Scene>
</X3D>
```

### 42.4.2 TextureProjectorPerspective

```
TextureProjectorPerspective : X3DTextureProjectorNode {
  SFString [in,out] description ""
  SFVec3f [in,out] direction 0 0 1 (-∞,∞)
  SFFloat [in,out] farDistance 10
  SFFloat [in,out] fieldOfView π/4 (0,π)
  SFBool [in,out] global TRUE
  SFVec3f [in,out] location 0 0 0 (-∞,∞)
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFFloat [in,out] nearDistance 1
  SFBool [in,out] on TRUE
  SFNode [in,out] texture NULL [X3DTexture2DNode]
  SFVec3f [in,out] upVector 0 0 1
  SFFloat [out] aspectRatio (0,∞)
}
```

Perspective texture mapping is shown in the following figure.



**Figure 42.4 — Description of 3D perspective texture mapping**

The *description* field of this node tells the name of the perspective projector, and makes the division of different perspective projectors possible.

The *location* shows the position of the perspective projector, and this implies perspective projection point.

The *direction* is the way the perspective projector is heading, and this implies to perspective projection direction.

The *fieldOfView* is the extent of the observable world that is perspective seen on the display at any given moment. This value may change depending on the aspect ratio of the rendering resolution. The default value of this field is  $\pi/4$ .

The *aspectRatio* is the aspect ratio of the width and length which refers to perspective projection spect ratio.

The *upVector* describes the roll of the camera by saying which point is "up" in the camera's orientation. The default value of this field is (0 0 1).

The *nearDistance* and *farDistance* is the minimum and maximum distance that is shown on the screen, respectively.

*global*, *on* and *texture* fields are the same as illustrated in the Abstract node.

## 42.5 Support levels

The projective texture mapping component defines levels of support as specified in [Table 42.2](#).

**Table 42.2 — Projective texture mapping component support levels**

Level	Prerequisites	Nodes/Features	Support
-------	---------------	----------------	---------

	1	Core 1 Grouping 1 Shape 1 Rendering 1 Texturing 1		
			<i>X3DTextureProjectorNode</i>	n/a
			TextureProjectorPerspective	All fields fully supported.
	2	Core 1 Grouping 1 Shape 1 Rendering 1 Texturing 1		
			TextureProjectorParallel	All fields fully supported.







## Extensible 3D (X3D) Part 1: Architecture and base components

### 43 Annotation component



#### 43.1 Introduction

##### 43.1.1 Name

The name of this component is "Annotation". This name shall be used when referring to this component in the COMPONENT statement (see [7.2.5.4 Component statement](#)).

##### 43.1.2 Overview

This clause describes the Annotation component of this part of ISO/IEC 19775. This component specifies how to include additional information as part of an X3D file that is beyond basic metadata, and some or all of which may be displayed externally to the scene. [Table 43.1](#) provides links to the major topics in this clause.

**Table 43.1 — Topics**

- [43.1 Introduction](#)
  - [43.1.1 Name](#)
  - [43.1.2 Overview](#)
- [43.2 Concepts](#)
  - [43.2.1 Overview](#)
  - [43.2.2 Representing annotations](#)
    - [43.2.2.1 Parts of an annotation](#)
    - [43.2.2.2 Coordinate system](#)
- [43.3 Abstract types](#)
  - [43.3.1 X3DAnnotationNode](#)
- [43.4 Node reference](#)
  - [43.4.1 AnnotationLayer](#)
  - [43.4.2 AnnotationTarget](#)
  - [43.4.3 GroupAnnotation](#)
  - [43.4.4 IconAnnotation](#)
  - [43.4.5 TextAnnotation](#)
  - [43.4.6 URLAnnotation](#)

### [43.5 Support levels](#)

- [Table 43.1 — Topics](#)
- [Table 43.2 — Display policies](#)
- [Table 43.3 — Layout policies](#)
- [Table 43.4 — Annotation component support levels](#)

## 43.2 Concepts

### 43.2.1 Overview

Annotations occupy the information space between metadata provided by the core specification and the rendered visuals presented by the rest of the X3D specification. The goal is to provide the ability to include or give reference to large amounts of information that may be available in scene or, optionally at the browser implementer's choice, externally to the scene. Another term for annotations is "labelling" which implies purely in-scene information, where this specification seeks to provide both in-scene and out-of-scene information display capabilities.

### 43.2.2 Representing annotations

#### 43.2.2.1 Parts of an annotation

An annotation is comprised of three parts:

1. the object that is being annotated,
2. the information to be associated with that object, and
3. a visual connection between the two, such as a leadline.

The leadline and information may be optionally presented by the browser.

EXAMPLE A hide-away panel may be used to display annotations that are available for currently visible objects.

In addition, the user is provided with an ability to describe the preferred policy for the browser to show annotations that are not currently visible, such as when they are behind the current viewpoint.

#### 43.2.2.2 Coordinate system

The object that is being annotated will have the [AnnotationTarget](#) node associated with the containing grouping node. This node is not renderable, but provides a point in the local coordinate system that forms the end point of the annotation's leadline connecting that object.

Annotations themselves are not part of the renderable scene graph. They may, optionally, exist anywhere and are not effected by the parent transformation hierarchy, such as Switch or LOD nodes. The AnnotationTarget node is effected by parent transformation hierarchy changes, including those that disable parts of the visible scene graph. When the parent hierarchy hides the renderable parts, the target and any

associated leadlines are also hidden.

## 43.3 Abstract types

### 43.3.1 X3DAnnotationNode

```
X3DAnnotationNode : X3DChildNode {
  SFString [in,out] annotationGroupID ""
  SFString [in,out] displayPolicy "NEVER" ["POINTER_OVER", "POINTER_ACTIVATE",
                                         "ALWAYS", "WHEN_VISIBLE", "NEVER"]
  SFBool [in,out] enabled TRUE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This abstract node type is the base type for all node types that contain annotation information. The abstract type provides the facility to control the policy of when the annotation should be displayed to the user through the *displayPolicy* field. Additionally, a grouping label can be assigned to each annotation through the use of the *annotationGroupID* field.

Large files with many annotations can become unwieldy to view. Two options are available to control how and when annotations are to be made visible. The first option is to create a global filter list through the use of the *annotationGroupID* field. This is a text string that can be used to group sets of similar annotations together. It is recommended that plain, readable text be used for this field as a browser may present the values of these fields in a list of filter options. The second option controls when an individual annotation will show based on user actions. These provide basic shortcuts without the user needing to use combinations of touch sensors and scripting. Available basic behaviours are in [Table 43.2](#):

**Table 43.2 — Display policies**

POINTER_OVER	Show the annotation when the pointing device is over the targets that reference this node.
POINTER_ACTIVATE	Show this annotation when the pointing device has clicked on the target of this node. It will remain active until replaced by another annotation.
WHEN_VISIBLE	Show this annotation when it is visible.
ALWAYS	Always show this annotation.
NEVER	Never show this annotation.

The *enabled* field determines if the annotation is displayed. If *enabled* is `TRUE`, the annotation is displayed as described. If *enabled* is `FALSE`, no action is performed.

## 43.4 Node reference

### 43.4.1 AnnotationLayer

```
AnnotationLayer : X3DLayerNode {
  MFString [in,out] layoutPolicy "" ["circular", "edges", ...]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

```

SFBool [in,out] pickable TRUE
MFString [in,out] shownGroupID []
SFNode [in,out] viewport NULL [X3DViewportNode]
}

```

The `AnnotationLayer` node is custom layer version that provides automated layout of currently visible annotations and display of them within the currently running scene.

The layer shows annotations directly in the scene and lays them out around the target object's reference point according to one of the pre-defined layout policies that are defined in the `layoutPolicy` field. This field contains a list of policies in priority order based on what the browser implementation supports. Browsers may also define implementation-specific policies in addition to the required policies. The reference point is projected into the layer's space and used as the location to base the annotations around. The defined policies are specified in [Table 43.3](#):

**Table 43.3 — Layout policies**

CIRCULAR	Show the annotation when the pointing device is over the targets that reference this node.
DISPLAY_EDGE	Show this annotation when the pointing device has clicked on the target of this node. It will remain active until replaced by another annotation.

## 43.4.2 AnnotationTarget

```

AnnotationTarget : X3DChildNode {
  MFNode [in,out] annotations [] [X3DAnnotationNode]
  SFNode [in,out] leadLineStyle NULL [X3DLinePropertiesNode]
  SFNode [in,out] marker NULL [X3DShapeNode]
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFVec3f [in,out] referencePoint 0, 0, 0 (-∞,∞)
}

```

The `AnnotationTarget` node specifies the target with which annotations are associated. A target applies to all siblings of the parent grouping node. If the parent or target has more than one parent transformation hierarchy, each shall be rendered individually, including leader lines and markers to each visual object.

Each target may have zero or more annotation nodes associated with it as specified by the `annotations` field. All nodes referenced in this field are effected the position of this node's parent group in the world coordinates.

A grouping node may have more than one `AnnotationTarget` specified, each with different sets of annotations.

Annotations are visually connected to the target through the use of a lead line if a node is defined for the `leadLineStyle` field. If no style is defined, a lead line is not shown and the annotation is shown with no connecting line. When a `LineProperties` node is provided and line ends are defined, the source end will be the target and the destination end will be the annotation.

## 43.4.3 GroupAnnotation

```

GroupAnnotation : X3DGroupingNode, X3DAnnotationNode {
  MFNode [in] addChildren [X3DChildNode]
}

```

```

MFNode [in] removeChildren [X3DChildNode]
SFString [in,out] annotationGroupID ""
MFNode [in,out] children [] [X3DChildNode]
SFString [in,out] displayPolicy "NEVER" ["POINTER_OVER", "POINTER_ACTIVATE",
"ALWAYS", "WHEN_VISIBLE", "NEVER"]
SFBool [in,out] enabled TRUE
SFNode [in,out] metadata NULL [X3DMetadataObject]
SFVec3f [] bboxCenter 0 0 0 (-∞,∞)
SFVec3f [] bboxSize -1 -1 -1 [0,∞) or -1 -1 -1
}

```

The GroupAnnotation node specifies annotation that is specified in the form of an X3D grouping node.

The annotation to be displayed is specified in the *children* field. The coordinate system established for this group is one in which the origin is at the end of the offset sequence with the XY-plane parallel with the screen plane. From this coordinate system the nodes in the *children* field may apply further transformations. See [4.3.5, Transformation hierarchy](#), and [4.3.6, Standard units and coordinate system](#), for a description of coordinate systems and transformations.

[10.2.1, Grouping and children node types](#), provides a description of the *children*, *addChildren*, and *removeChildren* fields.

The *bboxCenter* and *bboxSize* fields specify a bounding box that encloses the children of the GroupAnnotation node. This is a hint that may be used for optimization purposes. The results are undefined if the specified bounding box is smaller than the actual bounding box of the children at any time. A default *bboxSize* value, (-1, -1, -1), implies that the bounding box is not specified and, if needed, shall be calculated by the browser. The bounding box shall be large enough at all times to enclose the union of the group's children's bounding boxes; it shall not include any transformations performed by the group itself (*i.e.*, the bounding box is defined in the local coordinate system of the children). The results are undefined if the specified bounding box is smaller than the true bounding box of the group. A description of the *bboxCenter* and *bboxSize* fields is provided in [10.2.2 Bounding boxes](#).

#### 43.4.4 IconAnnotation

```

IconAnnotation : X3DAnnotationNode, X3DURLObject {
SFString [in,out] annotationGroupID ""
SFString [in,out] displayPolicy "NEVER" ["POINTER_OVER", "POINTER_ACTIVATE",
"ALWAYS", "WHEN_VISIBLE", "NEVER"]
SFBool [in,out] enabled TRUE
SFNode [in,out] metadata NULL [X3DMetadataObject]
MFString [in,out] url []
}

```

The IconAnnotation node specifies annotation that is iconic in form using an image specified by the URL field.

The icon to be displayed is read from location specified by the *url* field. When the *url* field contains no satisfiable values, the browser implementation shall substitute a default icon in its place. Browsers shall support the JPEG (see [2.\[JPEG\]](#)) and PNG (see [ISO/IEC 15948](#)) image file formats. Browsers may support other image file formats. Details on the *url* field can be found in [9.2.1 URLs](#).

#### 43.4.5 TextAnnotation

```

TextAnnotation : X3DAnnotationNode {
SFString [in,out] annotationGroupID ""
SFString [in,out] contentType "text/plain"
SFString [in,out] displayPolicy "NEVER" ["POINTER_OVER", "POINTER_ACTIVATE",

```

```

"ALWAYS", "WHEN_VISIBLE", "NEVER"]
SFBool [in,out] enabled TRUE
SFNode [in,out] metadata NULL [X3DMetadataObject]
SFString [in,out] text ""
}

```

The TextAnnotation node specifies an annotation that contains in-lined formatted text. The text may be one of several formats based on the defined MIME type in the *contentType* field. All browsers shall support the default plain text content type, and may support other content types (e.g., HTML).

### 43.4.6 URLAnnotation

```

URLAnnotation : X3DAnnotationNode {
SFString [in,out] annotationGroupID ""
SFString [in,out] displayPolicy "NEVER" ["POINTER_OVER", "POINTER_ACTIVATE",
"ALWAYS", "WHEN_VISIBLE", "NEVER"]
SFBool [in,out] enabled TRUE
SFNode [in,out] metadata NULL [X3DMetadataObject]
MFString [in,out] url []
}

```

The URLAnnotation node specifies an annotation that defines its content in another file.

The location of the other file is defined by the *url* field. This annotation is not required to be immediately loaded. A browser may choose to load the URL or just display the URL for the user to select and load externally.

## 43.5 Support levels

The Volume Rendering component provides two levels of support as specified in [Table 43.4](#).

**Table 43.4 — Annotation component support levels**

Level	Prerequisites	Nodes/Features	Support
1	Core 1 Grouping 1 Shape 1 Rendering 1 Networking 1		
		<i>X3DAnnotationNode</i>	n/a
		AnnotationTarget	All fields fully supported.
		IconAnnotation	All fields fully supported.
		TextAnnotation	All fields fully supported.
		URLAnnotation	All fields fully supported.
2	Core 1 Grouping 1 Shape 1 Rendering 1 Networking 1,		

	Layering 1		
		All Level 1 nodes	All fields fully supported.
		AnnotationLayer	All fields fully supported.
		GroupAnnotation	All fields fully supported.



Draft



## Extensible 3D (X3D) Part 1: Architecture and base components

### 4 Concepts

Editors note: multiple sections in the Concepts clause will receive additions and modifications to describe how X3D models are included and interact with external surfaces such as HTML5/DOM Web-page presentations. Current focus is on open-source implementation and evaluation using [X3DOM](#) and [X\\_ITE](#).

---



#### 4.1 General

##### 4.1.1 Topics in this clause

This clause describes the X3D core concepts, including how X3D scenes are authored and played back, the run-time semantics of the X3D scene, modularization through components and profiles, conformance via support levels, data encoding semantics, programmatic access, and networking considerations.

[Table 4.1](#) provides links to the major topics in this clause.

**Table 4.1 — Topics**

- [4.1 General](#)
  - [4.1.1 Topics in this clause](#)
  - [4.1.2 Overview](#)
  - [4.1.3 Conventions used](#)
- [4.2 Authoring and playback](#)
  - [4.2.1 X3D browsers](#)
  - [4.2.2 X3D generators](#)
  - [4.2.3 X3D loaders](#)
- [4.3 The scene graph](#)
  - [4.3.1 Overview](#)
  - [4.3.2 Root nodes](#)
  - [4.3.3 Scene graph hierarchy](#)
  - [4.3.4 Descendant and ancestor nodes](#)
  - [4.3.5 Transformation hierarchy](#)
  - [4.3.6 Standard units and coordinate system](#)



- [4.3.7 Behaviour graph](#)
- [4.4 Run-time environment](#)
  - [4.4.1 Overview](#)
  - [4.4.2 Object model](#)
    - [4.4.2.1 Overview](#)
    - [4.4.2.2 Field semantics](#)
    - [4.4.2.3 Interface hierarchy](#)
    - [4.4.2.4 Modifying objects](#)
      - [4.4.2.4.1 Routes](#)
      - [4.4.2.4.2 Modifying objects via programmatic access](#)
    - [4.4.2.5 Object life cycle](#)
  - [4.4.3 DEF/USE semantics](#)
  - [4.4.4 Prototype semantics](#)
    - [4.4.4.1 Introduction](#)
    - [4.4.4.2 PROTO interface declaration semantics](#)
    - [4.4.4.3 PROTO definition semantics](#)
    - [4.4.4.4 Prototype scoping rules](#)
  - [4.4.5 External prototype semantics](#)
    - [4.4.5.1 Introduction](#)
    - [4.4.5.2 EXTERNPROTO interface semantics](#)
    - [4.4.5.3 EXTERNPROTO URL semantics](#)
  - [4.4.6 Import/Export semantics](#)
  - [4.4.7 Run-time name scope](#)
  - [4.4.8 Event model](#)
    - [4.4.8.1 Events](#)
    - [4.4.8.2 Routes](#)
    - [4.4.8.3 Execution model](#)
    - [4.4.8.4 Loops](#)
    - [4.4.8.5 Fan-in and fan-out](#)
    - [4.4.8.6 Internal/external event passing](#)
- [4.5 Components](#)
  - [4.5.1 Overview](#)
  - [4.5.2 Defining components](#)
  - [4.5.3 Base components](#)
- [4.6 Profiles](#)
  - [4.6.1 Overview](#)
  - [4.6.2 Defining profiles](#)
  - [4.6.3 Relationship between profiles and components](#)
- [4.7 Support levels](#)
- [4.8 Data encodings](#)
- [4.9 Scene access interface \(SAI\)](#)
- [4.10 Component and profile registration](#)
- [Figure 4.1 — X3D Architecture](#)
- [Figure 4.2 — Interface hierarchy](#)
- [Figure 4.3 — Conceptual execution model](#)

- [Table 4.1 — Topics](#)
- [Table 4.2 — Standard units](#)
- [Table 4.3 — Derived units](#)
- [Table 4.4 — Rules for mapping PROTOTYPE declarations to node instances](#)
- [Table 4.5 — Example support level table](#)

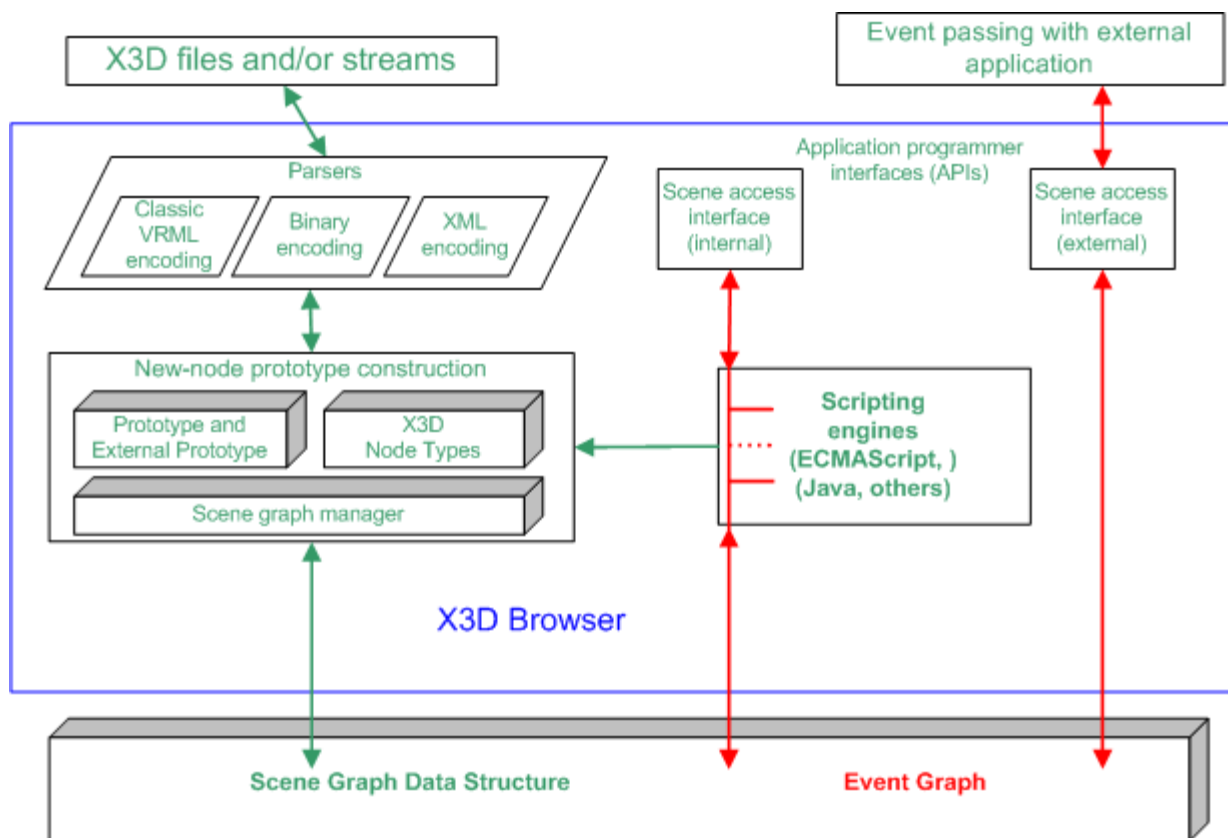
## 4.1.2 Overview

Conceptually, each X3D application is a 3D time-based space that contains graphic and aural objects that can be loaded over a network and dynamically modified through a variety of mechanisms. The semantics of X3D describe an abstract functional behaviour of time-based, interactive 3D, multimedia information. X3D does not define physical devices or any other implementation-dependent concepts (*e.g.*, screen resolution and input devices). X3D is intended for a wide variety of devices and applications, and provides wide latitude in interpretation and implementation of the functionality. For example, X3D does not assume the existence of a mouse or 2D display device.

Each X3D application:

- a. implicitly establishes a world coordinate space for all objects defined, as well as all objects included by the application;
- b. explicitly defines and composes a set of 2D, 3D and multimedia objects;
- c. can specify hyperlinks to other files and applications;
- d. can define object behaviours;
- e. can connect to external modules or applications via programming and scripting languages.

The X3D system architecture is shown in [Figure 4.1](#).



**Figure 4.1 — X3D architecture**

The abstract structure of the sequence of statements that form an X3D world is specified in [7.2.5 Abstract X3D structure](#).

### 4.1.3 Conventions used

The following conventions are used throughout this part of ISO/IEC 19775:

*Italics* are used for field names, and are also used when new terms are introduced and equation variables are referenced.

A `fixed-space` font is used for URL addresses and source code examples.

Node type names are appropriately capitalized (*e.g.*, "The Billboard node is a grouping node..."). However, the concept of the node is often referred to in lower case in order to refer to the semantics of the node, not the node itself (*e.g.*, "To rotate the billboard...").

The form "0xhh" expresses a byte as a hexadecimal number representing the bit configuration for that byte.

Throughout this part of ISO/IEC 19775, references to International Standards cite the number of the standard and hyperlinks to the reference in [2 Normative references](#). References to portions of this International Standard consist of the clause or subclause number followed by the title of the clause or subclause. The text consisting of the number and title is hyperlinked to the referenced material. References to external documents that are not International Standards are denoted using the "x.[ABCD]" notation, where "x" denotes in which clause the reference is described and "[ABCD]" is

an abbreviation of the reference title. For the Bibliography, the "x." is omitted.

In addition, the first reference to a node or node type in a subclause will be hyperlinked to the definition of that node or node type.

EXAMPLE "2.[ABCD]" refers to a reference described in [2 Normative references](#) and [ABCD] refers to a reference described in the [Bibliography](#).

## 4.2 Authoring and playback

### 4.2.1 X3D browsers

The interpretation, execution, and presentation of X3D files occurs using a mechanism known as a *browser*, which displays the shapes and sounds in the scene graph. This presentation is known as a *virtual world* and is navigated in the browser by a human or mechanical entity, known as a *user*. The world is displayed as if experienced from a particular location; that position and orientation in the world is known as the *viewer*. The browser may provide navigation paradigms (such as walking or flying) that enable the user to move the viewer through the virtual world.

In addition to navigation, the browser provides a limited mechanism allowing the user to interact with the world through sensor nodes in the scene graph hierarchy. Sensors respond to user interaction with geometric objects in the world, the movement of the user through the world, or the passage of time. Additionally, the X3D Scene Access Interface (SAI) defined in [Part 2 of this International Standard](#) provides mechanisms for getting user input, and for getting and setting the current viewpoint. To provide navigation capabilities, a viewer may use the SAI to provide the user with the ability to navigate. Additionally, authors may use scripting or programming languages with bindings to the SAI to implement their own navigation algorithms. Other profiles may specify navigation capabilities as a requirement of the viewer; implementations of such viewers will typically do so by making use of the SAI.

The visual presentation of geometric objects in an X3D world follows a conceptual model designed to resemble the physical characteristics of light. The X3D lighting model describes how appearance properties and lights in the world are combined to produce displayed colours (see [17 Lighting component](#) for details).

### 4.2.2 X3D generators

A *generator* is a human or computerized creator of X3D files. It is the responsibility of the generator to ensure the correctness of the X3D file and the availability of supporting assets (*e.g.*, images, audio clips, other X3D files) referenced therein. It is also the responsibility of the generator to insure that the functionality represented in the X3D file is correctly stated in the profile, component and level information in the header statement of the file.

### 4.2.3 X3D loaders

A *loader* is a program responsible for loading X3D content but does not apply any run-time execution to the content. Geometry is presented as though time has not run,

although the loader is free to load textures and other remotely defined content. A time zero loader is typically found in modelling tools that intend to construct or modify existing X3D content without evaluating the run-time aspects of the specification.

A second form of loader may load files and allow run-time execution of content, but it does so as part of a larger user interface and 3D graphics rendering engine. Such loaders might be used to load individual models such as trees in a game environment, but the run-time evaluation of the X3D content is dependent on the external application, and is not self contained in the same fashion as an X3D browser.

## 4.3 The scene graph

### 4.3.1 Overview

The basic unit of the X3D run-time environment is the *scene graph*. This structure contains all the objects in the system and their relationships. Relationships are contained along several axes of the scene graph. The *transformation hierarchy* describes the spatial relationship of rendering objects. The *behavior graph* describes the connections between fields and the flow of events through the system. Each scene graph may also interact with external surfaces such as HTML5/DOM Web-page presentations.

### 4.3.2 Root nodes

An X3D file contains zero or more root nodes. The root nodes for an X3D file are those nodes defined by the node statements or USE statements that are not contained in other node or PROTO statements. Root nodes shall be children nodes as specified in [10 Grouping component](#) or the LayerSet node as specified in [35.4.2 LayerSet](#).

X3D4 goals related to HTML5/DOM:

- Usage of either X3D or X3DCanvas capabilities for style, other HTML attributes
- url (or src) field to simply refer to an X3D model to load (see current X\_ITE approach)
- specifying that multiple encodings are allowed
- whether or not multiple distinct scenes can be loaded at once, or require separate declarations

### 4.3.3 Scene graph hierarchy

An X3D scene graph is a directed acyclic graph. Nodes can contain specific fields with one or more children nodes which participate in the hierarchy. These may, in turn, contain nodes (or instances of nodes). This hierarchy of nodes is called the *scene graph*. Each arc in the graph from A to B means that node A has a field whose value directly contains node B. See [\[FOLEY\]](#) for details on hierarchical scene graphs.

### 4.3.4 Descendant and ancestor nodes

The *descendants* of a node are all of the nodes in its fields, as well as all of those nodes'

descendants. The *ancestors* of a node are all of the nodes that have the node as a descendant.

### 4.3.5 Transformation hierarchy

The transformation hierarchy includes all of the root nodes and root node descendants that are considered to have one or more particular locations in the virtual world. X3D includes the notion of *local coordinate systems*, defined in terms of transformations from ancestor coordinate systems. The coordinate system in which the root nodes are displayed is called the *world coordinate system*.

An X3D browser's task is to present an X3D file to the user; it does this by presenting the transformation hierarchy to the user. The transformation hierarchy describes the directly perceptible parts of the virtual world.

Some nodes, such as sensors and environmental nodes, are in the scene graph but not affected by the transformation hierarchy. These include [CoordinateInterpolator](#), [Script](#), [TimeSensor](#), and [WorldInfo](#).

Some nodes, such as [Switch](#) or [LOD](#), contain a list of children, of which at most one is traversed during rendering. However, for the purposes of computing scene position, all children of these nodes are considered to be part of the transformation hierarchy, whether they are traversed during rendering or not. For instance, a [Viewpoint](#) node which is a child of a Switch whose `whichChoice` field is set to -1 (indicating that none of its children should be traversed during rendering) still uses the local coordinate space of the Switch to determine its position in the scene.

The transformation hierarchy shall be a directed acyclic graph; a node in the transformation hierarchy that is its own ancestor is considered invalid and shall be ignored. The following is an example of a node in the scene graph that is its own ancestor:

```
DEF T Transform {
  children {
    shape { ... }
    USE T
  }
}
```

### 4.3.6 Standard units and coordinate system

ISO/IEC 19775 defines the initial base unit of measure of the world coordinate system to be metres. However, the world coordinate units may be modified by specifying a different length unit using the UNIT statement. All other coordinate systems are then built from transformations based upon the specified world coordinate system. Other measurements used in this International Standard have their own initial base units.

[Table 4.2](#) lists the initial base units for ISO/IEC 19775, including the reference for each unit in [ISO 80000](#).

**Table 4.2 — Standard units**

Category	Initial base unit	Reference
----------	-------------------	-----------

angle	radian	ISO 80000-3:2006 item 3-5.a
force	newton	ISO 80000-4:2006 item 4-9.a and item 4-9.1
length	metre	ISO 80000-3:2006 item 3-1.a
mass	kilogram	ISO 80000-4:2006 item 4-1.a

The initial base units for the entire hierarchy of an X3D world may be changed to another default base unit by using one or more UNIT statements as specified in [7 Core component](#). In this International Standard, the initial base units of measure are assumed. Any ranges specified in initial base units apply to their equivalent limits in the specified default base unit. The browser shall convert the default base unit to initial base units as necessary for correct processing.

The base unit of time is seconds and cannot be changed.

Additional units, called *derived units* are used in this International Standard. A derived unit depends on the current base units. The value for a derived unit can be calculated using the appropriate formula from Table 4.3:

**Table 4.3 — Derived units**

Category	Initial base unit	Reference
acceleration	length/second <sup>2</sup>	ISO 80000-3:2006 item 3-9.a
angular_velocity angular_rate	angle/second	ISO 80000-3:2006 item 3-10.a
area	length <sup>2</sup>	ISO 80000-3:2006 item 3-3.a
speed	length/second	ISO 80000-3:2006 item 3-8.a and item 3-8.1
volume	length <sup>3</sup>	ISO 80000-3:2006 item 3-4.a

The standard colour space used by this International Standard is RGB where each colour component has the range [0.,1.].

ISO/IEC 19775 uses a Cartesian, right-handed, three-dimensional coordinate system. By default, the viewer is on the Z-axis looking down the -Z-axis toward the origin with +X to the right and +Y straight up. A modelling transformation (see the [Transform](#) node definition in [10 Grouping component](#) and the [Billboard](#) node definition in [23 Navigation component](#)) or viewing transformation (see the [X3DViewpointNode](#) node type definition in [23 Navigation component](#)) can be used to alter this default projection.

### 4.3.7 Behaviour graph



The event model of X3D allows the declaration of connections between fields (routes) and a model for the propagation of events along those connections. The behavior graph is the collection of these field connections. It can be changed dynamically by rerouting, adding or breaking connections. Events are injected into the system and propagate through the behavior graph in a well defined order.

Fields can only be routed to other fields with the same data type, unless a component supports an extension to this rule.

## 4.4 Run-time environment

### 4.4.1 Overview

The X3D run-time environment maintains the current state of the scene graph, renders the scene as needed, receives input from a variety of sources (*Sensors*) and performs changes to the scene graph in response to instructions from the behavioral system. The X3D run-time environment manages the life cycle of objects, including built-in and user-defined objects and programmatic scripts. The run-time environment coordinates the processing of *Events*, the primary means of generating behaviors in X3D. The run-time environment also manages interoperation between the X3D browser and host application for file delivery, hyperlinking, page integration and external programmatic access.

The run-time environment manages objects. X3D supports several types of *built-in objects* that contain generally useful functionality in the run-time environment. There are built-in objects to represent data structures such as an *SFVec3f* 3D vector value, nodes such as geometry (e.g., [Cylinder](#)), and ROUTEs between nodes. Each node contains zero or more *fields* that define storage for data values, and/or zero or more *events* for sending messages to/from the object. Nodes are instantiated by declaring them in a file or by using procedural code at run-time. The author may create new node types using the prototyping mechanism (see [4.4.4 Prototype semantics](#)). These nodes become part of the run-time environment and behave exactly like built-in nodes. New nodes can be created declaratively by including a prototype declaration in a file, by including an external prototype referencing a prototype declaration in a separate location, or by using a native prototype declaration provided by the run-time environment itself. PROTOs may only be used to create other nodes, not fields or routes.

*Events* are the primary means of generating behaviors in the X3D run-time environment. Events are used throughout X3D: driving time-based animations; handling object picking; detecting user movement and collision; changing the scene graph hierarchy. The run-time environment manages the propagation of events through the system and order of evaluation according to a well-defined set of rules.

An author of X3D content can control the creation and management of scenes, rendering and behavior, and loading of media assets. The loading and incorporation of authored extensions, which can be written in X3D or an external language, can also be controlled. The ability to make content-defined extensions is provided in profiles that support the Prototyping mechanism.



## 4.4.2 Object model

### 4.4.2.1 Overview

The X3D system is made up of abstract individual entities called *objects*. This part of ISO/IEC 19775 defines a functional specification for each object type but does not dictate implementation. A compliant implementation of an object shall behave according to its functional specification as provided in [5 Field type reference](#), clauses 7 through 40 describing components, [Part 2 of ISO/IEC 19775](#) or additional parts of this standard that define object, field or node types. An X3D author arranges objects in the scene graph using one of the declarative X3D encodings described in [ISO/IEC 19776](#) or other future encoding formats, or at run time using built-in scripting (if the supported profile provides it) or some other form of programmatic access to the scene graph (see [Part 2 of ISO/IEC 19775](#)).

Objects representing lightweight concepts such as data storage and operations on data of that type are called *fields* and are derived from the [X3DField](#) object. Objects representing more complete spatial or temporal processing concepts are called *nodes* and are derived from the [X3DNode](#) object. Nodes contain one or more fields that hold data values or send or receive events for that node.

Some nodes implement additional functionality by inheritance of *interfaces* that represent common properties or functionality, such as bounding boxes for visual objects and grouping nodes or a specification that a particular object represents metadata. In addition, X3D defines object types for accessing scene graph information not stored in fields or nodes, such as ROUTEs, PROTO declarations, Component/Profile information and world metadata.

A field may contain either a single value of the given type or an array of such types. Throughout this document, a field type containing a single value is said to be of the given type and is prefixed by the characters *SF* (e.g., field *a* is of type *SFVec3f*), while a field containing an array has its type prefixed by the characters *MF* (e.g., field *b* is of type *MFVec3f*). A field may contain a reference to one or more nodes by using the *SFNode* and *MFNode* field types.

Each object has the following common characteristics:

- a. **A type name.** Examples include *SFVec3f*, *MFColor*, *SFFloat*, [Group](#), [Background](#), or [SpotLight](#).
- b. **An implementation.** The implementation of each object defines how it reacts to changes in its property values, what other property values it alters as a result of these changes, and how it effects the state of the run-time environment. This part of ISO/IEC 19775 defines the functional semantics of built-in nodes (*i.e.*, nodes with implementations that are provided by the X3D browser).

An object derived from *X3DNode* has the following additional characteristics:

- d. **Zero or more field values.** Field values are stored in the X3D file along with the nodes or fields, and encode the state of the virtual world.
- e. **Zero or more events that it can receive and send.** Each node may receive events to its fields which will result in some change to the node's state. Each node

may also generate events from its fields to report changes in the node's state. Events generated from one node can be connected to fields of other nodes to propagate these changes. This is done using the ROUTE statement in the file or through an SAI service reference.

- f. **A name.** Nodes can be named using either the DEF statement in the file or at run-time through an SAI service call. This is used by other statements to reference a specific instantiation of a node. It is also be used to locate a specific named node within the scene hierarchy.

Node implementations can come from two sources, built-in nodes and prototypes. Built-in nodes are nodes that are available to the author as specified by the applicable profile and/or component declarations. Different components define different sets of built-in nodes.

Additionally, X3D supports content extensions using prototypes. Prototypes are objects that the author creates using PROTO or EXTERNPROTO statements. These objects are written in the same declarative notation used to describe nodes in the scene graph. They add new object types to the system which are only available for the lifetime of the session into which they are loaded. Some profiles may not include support of these extension capabilities. The semantics of prototypes are discussed in [4.4.4, Prototype semantics](#), and [4.4.5, External prototype semantics](#).

Both prototypes and built-in nodes are available for instantiation using similar mechanisms. An object can be instantiated declaratively or at run-time using the SAI services specified in [Part 2 of ISO/IEC 19775](#). All prototypes inherit from the base node type [X3DPrototypeInstance](#).

#### 4.4.2.2 Field semantics

Fields define the persistent state of nodes, and values which nodes may send or receive in the form of events. X3D supports four types of access to a node's fields:

- a. *initializeOnly* access, which allows content to supply an initial value for the field but does not allow subsequent changes to its value;
- b. *inputOnly* access, which means that the node may receive an event to change the value of its field, but does not allow the field's value to be read;
- c. *outputOnly* access, which means that the node may send an event when its value changes, but does not allow the field's value to be written; and
- d. *inputOutput* access, which allows full access to the field: content may supply an initial value for the field, the node may receive an event to change the value of its field, and the node may send an event when its value changes.

An inputOutput field can receive events like an inputOnly field, can generate events like an outputOnly field, and can be stored in X3D files like an initializeOnly field. An inputOutput field named *zzz* can be referred to as '*set\_zzz*' and treated as an inputOnly, and can be referred to as '*zzz\_changed*' and treated as an outputOnly field. Within ISO/IEC 19775, fields with inputOutput access or inputOnly access are collectively referred to as *input* fields, fields with inputOutput access or outputOnly access are collectively referred to as *output* fields, and the events these fields receive and send are called *input events* and *output events*, respectively.

The initial value of an inputOutput field is its value in the X3D file, or the default value for the node in which it is contained, if a value is not specified. When an inputOutput field receives an event it shall generate an event with the same value and timestamp. The following sources, in precedence order, shall be used to determine the initial value of the inputOutput field:

- e. the user-defined value in the instantiation (if one is specified);
- f. the default value for that field as specified in the node or prototype definition.

The recommendations for naming initializeOnly fields, inputOutput fields, outputOnly fields, and inputOnly fields for built-in nodes are as follows:

- g. All names containing multiple words start with a lower case letter, and the first letter of all subsequent words is capitalized (*e.g.*, *addChildren*), with the exception of *set\_* and *\_changed*, as described below.
- h. It is recommended that all inputOnly fields have the prefix "*set\_*", with the exception of the *addChildren* and *removeChildren* fields.
- i. Certain inputOnly fields and outputOnly fields of type SFTime do not use the "*set\_*" prefix or "*\_changed*" suffix.
- j. It is recommended that all other outputOnly fields have the suffix "*\_changed*" appended, with the exception of outputOnly fields of type SFBool.

### 4.4.2.3 Interface hierarchy

Most object types derive some of their interfaces and functionality from other object types in the system. These are known as its *supertypes*, and an object is said to be *derived* from these supertypes. Likewise, these supertypes may derive their capabilities from other object types, forming a chain all the way to a small number of base types from which all the others are ultimately derived. The graph describing the relationship between all object types in the system is called the *interface hierarchy*. In this part of ISO/IEC 19775, the object hierarchy specifies conceptual relationships between objects but does not necessarily dictate actual implementation.

[Figure 4.2](#) depicts the object hierarchy for object types defined in this part of ISO/IEC 19775 for all versions. A specification of which object types are available for which versions may be found in [Annex L Version content](#).

NOTE Not all object types are supported in certain component levels, profiles or versions; refer to the individual component and profile specifications in this part of ISO/IEC 19775 for details.

```
X3DField +------ X3DArrayField +-
          +- SFBool                    +- MFBool
          +- SFColor                   +- MFColor
          +- SFColorRGBA                +- MFColorRGBA
          +- SFDouble                   +- MFDouble
          +- SFFloat                     +- MFFloat
          +- SFImage                     +- MFImage
          +- SFInt32                     +- MFInt32
          +- SFMatrix3d                  +- MFMatrix3d
          +- SFMatrix3f                  +- MFMatrix3f
          +- SFMatrix4d                  +- MFMatrix4d
          +- SFMatrix4f                  +- MFMatrix4f
          +- SFNode                       +- MFNode
          +- SFRotation                  +- MFRotation
          +- SFString                     +- MFString
          +- SFTime                       +- MFTime
          +- SFVec2d                      +- MFVec2d
          +- SFVec2f                      +- MFVec2f
          +- SFVec3d                      +- MFVec3d
          +- SFVec3f                      +- MFVec3f
```

+- SFVec4d +- MFVec4d  
 +- SFVec4f +- MFVec4f

X3DBoundedObject  
 X3DFogObject  
 X3DPickableObject  
 X3DProgrammableShaderObject  
 X3DMetadataObject  
 X3DUrlObject

X3DNode  
 +- Contact  
 +- Contour2D  
 +- EaseInEaseOut  
 +- GeoOrigin **(deprecated)**  
 +- LayerSet  
 +- MetadataBoolean (X3DMetadataObject)\*  
 +- MetadataDouble (X3DMetadataObject)\*  
 +- MetadataFloat (X3DMetadataObject)\*  
 +- MetadataInteger (X3DMetadataObject)\*  
 +- MetadataSet (X3DMetadataObject)\*  
 +- MetadataString (X3DMetadataObject)\*  
 +- NurbsTextureCoordinate  
 +- RigidBody  
 +- ShaderPart (X3DUrlObject)\*  
 +- ShaderProgram (X3DUrlObject, X3DProgrammableShaderObject)\*  
 +- TextureProperties

-- X3DAppearanceNode -- Appearance

-- X3DAppearanceChildNode -- **AcousticProperties**  
 +- FillProperties  
 +- LineProperties  
 +- **PointProperties**

+- X3DMaterialNode -- X3DOneSidedMaterialNode -- Material  
 +- PhysicalMaterial  
 +- UnlitMaterial  
 +- TwoSidedMaterial (deprecated)

X3DMaterialNode +- Material  
 +- TwoSidedMaterial

X3DProgrammableShaderObject)\*  
 +- X3DShaderNode -- ComposedShader (X3DProgrammableShaderObject)\*  
 +- PackagedShader (X3DUrlObject,  
 +- ProgramShader

-- X3DTextureNode -- MultiTexture

+- X3DSingleTextureNode -- X3DEnvironmentTextureNode -

+- ComposedCubeMapTexture

+- GeneratedCubeMapTexture

+- ImageCubeMapTexture (X3DUrlObject)\*

ImageTexture (X3DUrlObject)\* +- X3DTexture2DNode --

MovieTexture (X3DSoundSourceNode, X3DUrlObject)\* +-

PixelTexture +-

ComposedTexture3D +- X3DTexture3DNode --

ImageTexture3D (X3DUrlObject)\* +-

PixelTexture3D +-

+- X3DEnvironmentTextureNode

+- ComposedCubeMapTexture

+- GeneratedCubeMapTexture

+- ImageCubeMapTexture (X3DUrlObject)\*

ImageTexture (X3DUrlObject)\* +- X3DTexture2DNode +-

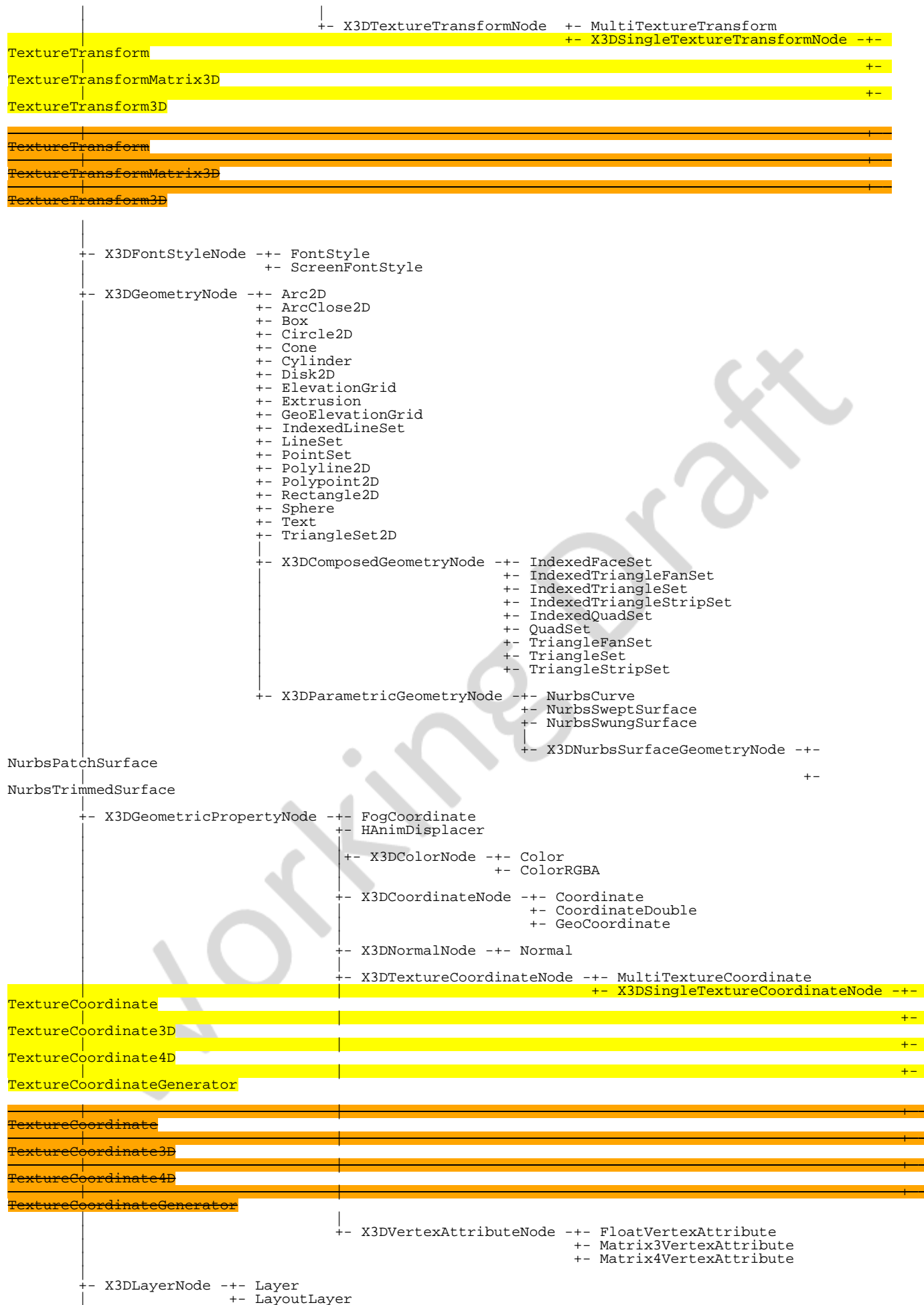
MovieTexture (X3DSoundSourceNode, X3DUrlObject)\* +-

PixelTexture +-

ComposedTexture3D +- X3DTexture3DNode +-

ImageTexture3D (X3DUrlObject)\* +-

PixelTexture3D +-



```

+- X3DNBodyCollisionSpaceNode (X3DBoundedObject)* +- CollisionSpace
+- X3DNurbsControlCurveNode +- ContourPolyline2D
  +- NurbsCurve2D
+- X3DParticleEmitterNode +- ConeEmitter
  +- ExplosionEmitter
  +- PointEmitter
  +- PolylineEmitter
  +- SurfaceEmitter
  +- VolumeEmitter
+- X3DParticlePhysicsModelNode +- BoundedPhysicsModel
  +- ForcePhysicsModel
  +- WindPhysicsModel
+- X3DProtoInstance
+- X3DRigidJointNode +- BallJoint
  +- DoubleAxisHingeJoint
  +- MotorJoint
  +- SingleAxisHingeJoint
  +- SliderJoint
  +- UniversalJoint
+- X3DVolumeRenderStyleNode +- ProjectionVolumeStyle
  +- X3DComposableVolumeRenderStyle +- BlendedVolumeStyle
    +- BoundaryEnhancementVolumeStyle
    +- CartoonVolumeStyle
    +- ComposedVolumeStyle
    +- EdgeEnhancementVolumeStyle
    +- OpacityMapVolumeStyle
    +- ProjectionVolumeStyle
    +- ShadedVolumeStyle
    +- SilhouetteEnhancementVolumeStyle
    +- ToneMappedVolumeStyle
+- X3DChildNode +- BooleanFilter
  +- BooleanToggle
  +- ClipPlane
  +- CollisionCollection
  +- DISEntityManager
  +- GeoLOD (X3DBoundedObject)*
  +- HAnimHumanoid (X3DBoundedObject)*
  +- HAnimMotion
  +- Inline (X3DUrlObject, X3DBoundedObject)*
  +- LocalFog (X3DFogObject)*
  +- NurbsOrientationInterpolator
  +- NurbsPositionInterpolator
  +- NurbsSet (X3DBoundedObject)*
  +- NurbsSurfaceInterpolator
  +- RigidBodyCollection
  +- StaticGroup (X3DBoundedObject)*
  +- ViewpointGroup
+- X3DBindableNode +- Fog (X3DFogObject)*
  +- GeoViewpoint
  +- NavigationInfo
  +- ListenerPoint
  +- X3DBackgroundNode +- Background
    +- TextureBackground
  +- X3DViewpointNode +- GeoViewpoint
    +- OrthoViewpoint
    +- Viewpoint
    +- ViewpointGroup
+- X3DFollowerNode +- X3DChaserNode +- ColorChaser
  +- CoordinateChaser
  +- OrientationChaser
  +- PositionChaser
  +- PositionChaser2D
  +- ScalarChaser
  +- TexCoordChaser2D
  +- X3DDamperNode +- ColorDamper
    +- CoordinateDamper
    +- OrientationDamper
    +- PositionDamper
    +- PositionDamper2D
    +- ScalarDamper
    +- TexCoordDamper
+- X3DGroupingNode (X3DBoundedObject)* +- Anchor
  +- Billboard
  +- CADAssembly
(X3DProductStructureChildNode)*
  +- CADLayer
  +- CADPart (X3DProductStructureChildNode)*
  +- Collision (X3DSensorNode)*
  +- EspduTransform (X3DSensorNode)*
  +- GeoLocation
  +- GeoTransform
  +- Group

```



```

+- HAnimJoint
+- HAnimSegment
+- HAnimSite
+- LayoutGroup
+- LOD
+- PickableGroup (X3DPickableObject)*
+- ScreenGroup
+- Switch
+- Transform
+- X3DViewportNode --+ Viewport

+- X3DInfoNode --+ DISEntityTypeMapping
  +- GeoMetadata
  +- WorldInfo

+- X3DInterpolatorNode --+ ColorInterpolator
  +- CoordinateInterpolator
  +- CoordinateInterpolator2D
  +- GeoPositionInterpolator
  +- NormalInterpolator
  +- OrientationInterpolator
  +- PositionInterpolator
  +- PositionInterpolator2D
  +- ScalarInterpolator
  +- SplinePositionInterpolator
  +- SplinePositionInterpolator2D
  +- SplineScalarInterpolator
  +- SquadOrientationInterpolator

+- X3DLayoutNode --+ Layout

+- X3DLightNode --+ DirectionalLight
  +- PointLight
  +- SpotLight

+- X3DNBodyCollidableNode (X3DBoundedObject)* --+ CollidableOffset
  +- CollidableShape

+- X3DProductStructureChildNode --+ CADAssembly (X3DGroupingNode)*
  +- CADFace (X3DBoundedObject)*
  +- CADPart (X3DGroupingNode)*

+- X3DTextureProjectorNode --+ TextureProjectorPerspective
  +- TextureProjectorParallel

+- X3DScriptNode (X3DUrlObject)* --+ Script

+- X3DSensorNode --+ Collision (X3DGroupingNode)*
  +- CollisionSensor
  +- EspduTransform (X3DGroupingNode)*
  +- ReceiverPdu (X3DBoundedObject)*
  +- SignalPdu (X3DBoundedObject)*
  +- TimeSensor (X3DTimeDependentNode)*
  +- TransmitterPdu (X3DBoundedObject)*
  +- X3DEnvironmentalSensorNode --+ GeoProximitySensor
    +- ProximitySensor
    +- TransformSensor
    +- VisibilitySensor
  +- X3DKeyDeviceSensorNode --+ KeySensor
    +- StringSensor
  +- X3DNetworkSensorNode --+ LoadSensor
  +- X3DPickSensorNode --+ LinePickSensor
    +- PointPickSensor
    +- PrimitivePickSensor
    +- VolumePickSensor
  +- X3DPointingDeviceSensorNode --+ X3DDragSensorNode --+
    |
    |
    |
    +- X3DTouchSensorNode --+
      +-

CylinderSensor
PlaneSensor
SphereSensor
GeoTouchSensor
TouchSensor

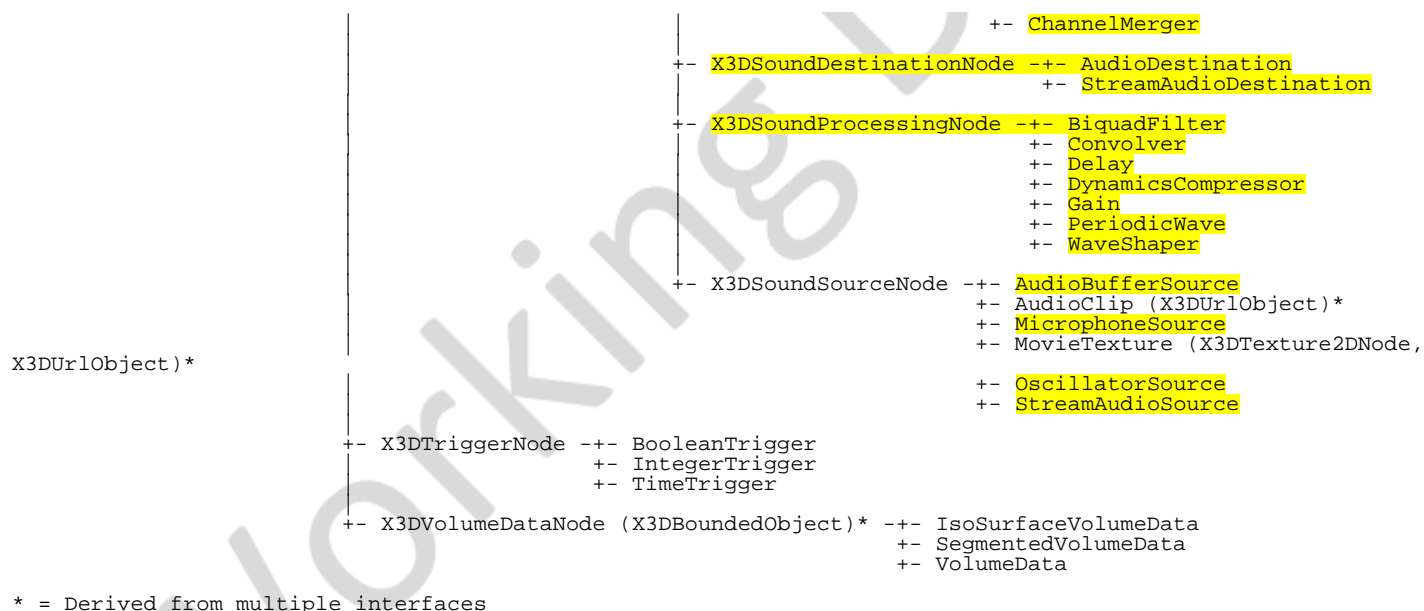
+- X3DSequencerNode --+ BooleanSequencer
  +- IntegerSequencer

+- X3DShapeNode (X3DBoundedObject)* --+ ParticleSystem
  +- Shape

+- X3DSoundNode --+ Sound
  +- SpatialSound

+- X3DTimeDependentNode --+ TimeSensor (X3DSensorNode)*
  +- X3DSoundAnalysisNode --+ Analyser
  +- X3DSoundChannelNode --+ ChannelSplitter

```



**Figure 4.2 — Interface hierarchy**

The object hierarchy defines both *abstract* interfaces and *concrete* node types. Abstract interfaces define functionality that is inherited by other interfaces and/or nodes, but are never instantiated in the scene graph as objects. Concrete node types derive from one or more abstract interfaces and may be instantiated in the scene graph. Thus, the live scene graph consists only of instances of concrete node types. Components defined in this part of ISO/IEC 19775 are required to implement the functionality of abstract interfaces only insofar as that functionality is made available via one of the derived concrete node types. [Part 2 of ISO/IEC 19775](#) defines the means by which applications may access the functionality provided in both abstract interfaces and concrete nodes via programmatic means.

The two main types of object from which most others are derived are [X3DNode](#) and [X3DField](#). Nodes are the objects used in the declarative language to form the scene graph, while fields are contained within nodes and hold the data items for nodes. Some field objects contain simple data values like integers or arrays of strings. Other field objects contain references to nodes. It is this ability of [X3DNode](#) to contain [X3DField](#), and [X3DField](#) to contain references to [X3DNode](#), that makes it possible for X3D to form scene graph hierarchies.

#### EXAMPLE

```

Transform { translation 1 2 3
  children [
    Shape {
      geometry Box { }
    }
    Group {
      children [ ... ]
    }
  ]
}
    
```

In the above example, the Transform contains a simple field, `translation`, which contains a vector of 3 numbers. It also contains a `children` field which may contain an array of other nodes. In this case it has two, a Shape and a Group. The Shape and the Group both contain fields which may have other objects as well.

Derivation makes it possible to strongly type all objects. In the above example, the `children` field is constrained to contain a list of objects derived from an object type



called [X3DChildNode](#). Both [Shape](#) and [Group](#) are derived (indirectly) from this object and can therefore be placed in the children field. The geometry field of Shape, on the other hand, can only contain a single node derived from [X3DGeometryNode](#). [Box](#) is derived from this object and can therefore be placed in the geometry field. But Box is not derived from [X3DChildNode](#), so it cannot be placed in the children field. Likewise, Group is not derived from [X3DGeometryNode](#) and can therefore not be placed in the geometry field.

The above example exhibits another quality of derivation. [Transform](#) is derived from [X3DGroupingNode](#) and therefore inherits its children field. This makes the specification of Transform simpler because it does not need to describe the functionality of the children field. Because it is derived from [X3DGroupingNode](#), the author knows it contains a children field which behaves like the one in Group which is also derived from [X3DGroupingNode](#).

## 4.4.2.4 Modifying objects

### 4.4.2.4.1 Routes

There are several ways to modify the fields of an object. Using one of the X3D file formats, an author can declare a set of nodes, the initial state of their fields, and interconnections between the fields called *Routes*. X3D uses an event propagation, or *dataflow* model to change the values of fields at run-time. As part of its abstract specification, the behavior of a node in response to events sent to its fields, and the conditions under which its fields send events out, is described.

EXAMPLE It is possible to create a scene with run-time behavior using only this event propagation model:

```
DEF TS TimeSensor {
  loop TRUE
  cycleInterval 5
}
DEF I PositionInterpolator {
  key [ 0 0.5 1 ]
  keyValue [ 0 -1 0, 0 1 0, 0 -1 0 ]
}
DEF T Transform {
  children [
    Shape {
      geometry Box { }
    }
  ]
}
ROUTE ts.fraction_changed TO I.set_fraction
ROUTE I.value_changed TO T.set_translation
```

This example bounces a box up and down repeatedly over a five-second interval. The TimeSensor object is defined to send an event continuously out of its `fraction` field. This event sends a floating point value which varies from 0 to 1 over a 5 second interval, as specified by the `cycleInterval`. Its `loop` field tells it to do so repeatedly. This fraction value is sent to the `fraction` field of a PositionInterpolator. This object is defined to send an event out of its `value` field whenever it receives an event on its fraction field. The value is determined by the `key` and `keyValue` fields. In this case it sends a vector whose y value varies between -1 and +1 and back again over the interval. This value is sent to the `translation` field of the Transform node. This node is defined to set the position of its children according to the value of `translation`. [4.4.8.2 Routes](#) contains more information on routing.

### 4.4.2.4.2 Modifying objects via programmatic access

The routing mechanism is simple, but is limited to changing field values of nodes, and only changes that are designed into a given node set. For greater flexibility, some

profiles provide programmatic access to objects in the system. This allows field values to be set and read, and functions to be called. Mechanisms are also provided to allow PROTO objects to be found, which in turn allows objects of that type to be instantiated.

There are two types of programmatic access in X3D: External access (EXAMPLE access from a containing HTML page or embedding native application) and Internal scripts using any of the supported scripting languages.

Programmatic access to objects is provided via *interfaces* to those objects. The interface of an object (its set of data and function properties) is specified, and is also referred to as the *object type*. An object type that represents a node is also referred to as a *node type*. Object types may be either abstract or concrete. Abstract object types are not instantiable. Instead, they are used to derive other object types or to indicate that a field may contain a node of any of the derivative node types. Concrete node types are those derived from abstract node types and are instantiable. A compliant implementation of an object's interface shall support the interface specifications as defined in [Part 2 of ISO/IEC 19775](#).

See [4.9, Application programmer interfaces](#) for additional information.

#### 4.4.2.5 Object life cycle

Nodes have a life cycle: they are created, used and eventually destroyed. A node is considered live if one or more of the following is true:

- a. The node is a root node in the scene.
- b. The node is referenced by a field of a live node.
- c. There is a reference from a live script to the node.
- d. There is an external programmatic reference to the node.

Rules b and c are applied recursively to cover the entire live scene graph.

Nodes instanced from a file are created implicitly by the browser upon encountering a node instance or upon instancing a prototype's scene graph. Nodes may also be instanced programmatically; in this case there are additional discrete steps in the node's life cycle. Refer to [Part 2 of ISO/IEC 19775](#) for more details.

#### 4.4.3 DEF/USE semantics

Node names are limited in scope to a single X3D file, prototype definition, or string submitted to either CreateX3DFromString, CreateX3DFromStream, or CreateX3DFromURL browser service or a construction for SFNodes within a script. The USE statement does not create a copy of the node. Instead, the same node is inserted into the scene graph a second time, resulting in the node having multiple parents (see [4.3.5 Transformation hierarchy](#), for restrictions on self-referential nodes).

Node names shall be unique in the context within which the associated DEF keyword occurs.

TODO: describe how, when an external environment exists,

- Abstract definition of how events can be exchanged between external environment and scene graph.
- Syntax for multiple encoding/language bindings may be defined in related specifications, e.g. updates to 19777-1 JavaScript, 19776-1 XML Encoding, and (eventually) 19776-5 JSON.
- For example, HTML5/DOM id attribute can be used for performing event callbacks using JavaScript, and thus has a similar role to DEF when events are ROUTEd.
- Editors discussion: examples should not go into an annex, will need to go into other file encodings and language bindings.
- Pending eventual ISO submission and review of those specifications, we will need example usage and some specification details publicly available to support implementation efforts.

## 4.4.4 Prototype semantics

### 4.4.4.1 Introduction

The PROTO statement defines a new node type in terms of already defined (built-in or prototyped) node types. Once defined, prototyped node types may be instantiated in the scene graph exactly like the built-in node types.

Node type names shall be unique in each X3D file. The results are undefined if a prototype is given the same name as a built-in node type or a previously defined prototype in the same scope.

### 4.4.4.2 PROTO interface declaration semantics

The prototype interface defines the fields and field access types for the new node type. The interface declaration includes the types, names and default values (for initializeOnly and inputOutput fields) for the prototype's fields.

The interface declaration may contain inputOutput field declarations, which are a convenient way of defining an initializeOnly field, inputOnly field, and outputOnly field at the same time. If an inputOutput field named *zzz* is declared, it is equivalent to separately declaring an initializeOnly field named *zzz*, an inputOnly field named *set\_zzz*, and an outputOnly field named *zzz\_changed*.

Each prototype instance can be considered to be a complete copy of the prototype, with its own field values and copy of the prototype definition. A prototyped node type is instantiated using standard node syntax. For example, the following prototype (which has an empty interface declaration):

```
PROTO Cube [ ] { Box { } }
```

may be instantiated as follows:

```
Shape { geometry Cube { } }
```

It is recommended that user-defined field names defined in PROTO interface declarations statements follow the naming conventions described in [4.4.2.2 Field semantics](#).

If an `outputOnly` field in the prototype declaration is associated with an `inputOutput` field in the prototype definition, the initial value of the associated `outputOnly` field shall be the initial value of the `inputOutput` field. If the `outputOnly` field is associated with multiple `inputOutput` fields, the results are undefined.

#### 4.4.4.3 PROTO definition semantics

A prototype definition consists of one or more nodes, nested PROTO statements, and ROUTE statements. The first node type determines how instantiations of the prototype can be used in an X3D file. An instantiation is created by filling in the parameters of the prototype declaration and inserting copies of the first node (and its scene graph) wherever the prototype instantiation occurs.

**EXAMPLE** If the first node in the prototype definition is a Material node, instantiations of the prototype can be used wherever a Material node can be used. Any other nodes and accompanying scene graphs are not part of the transformation hierarchy, but may be referenced by ROUTE statements or Script nodes in the prototype definition.

Nodes in the prototype definition may have their fields associated with the fields of the prototype interface declaration by using IS statements in the body of the node. When prototype instances are read from an X3D file, field values for the fields of the prototype interface may be given. If given, the field values are used for all nodes in the prototype definition that have IS statements for those fields. Similarly, when an input field of a prototype instance is sent an event, the event is delivered to all nodes that have IS statements for that field. When a node in a prototype instance generates an output event that has an IS statement, the event is sent to any input fields connected (via ROUTE) to the prototype instance's output field.

IS statements may appear inside the prototype definition wherever fields may appear. IS statements shall refer to fields defined in the prototype declaration. Results are undefined if an IS statement refers to a non-existent declaration. Results are undefined if the type of the field being associated by the IS statement does not match the type declared in the prototype's interface declaration. For example, it is illegal to associate an `SFColor` with an `SFVec3f`. It is also illegal to associate an `SFColor` with an `MFCColor` or *vice versa*.

Results are undefined if an IS statement:

- `inputOnly` field is associated with a `initializeOnly` field or an `outputOnly` field;
- `outputOnly` field is associated with a `initializeOnly` field or `inputOnly` field;
- `initializeOnly` field is associated with an `inputOnly` field or `outputOnly` field.

An `inputOutput` field in the prototype interface may be associated only with an `inputOutput` field in the prototype definition, but an `inputOutput` field in the prototype definition may be associated with either an `inputOutput` field, `inputOnly` field, or `outputOnly` field in the prototype interface. When associating an `inputOutput` field in a prototype definition with an `inputOnly` field or `outputOnly` field in the prototype declaration, it is valid to use either the shorthand `inputOutput` field name (e.g., *translation*) or the explicit field name (e.g., *set\_translation* or *translation\_changed*). [Table 4.4](#) defines the rules for mapping between the access types of fields in a prototype declarations and the access types for fields in its primary scene graph's nodes (*yes* denotes a legal mapping, *no* denotes an error).

**Table 4.4 — Rules for mapping PROTOTYPE declarations to node instances**

	Prototype declaration			
	<u>inputOutput</u>	<u>initializeOnly</u>	<u>inputOnly</u>	<u>outputOnly</u>
<b>Prototype definition</b>				
<u>inputOutput</u>	yes	yes	yes	yes
<u>intializeOnly</u>	no	yes	no	no
<u>inputOnly</u>	no	no	yes	no
<u>outputOnly</u>	no	no	no	yes

Results are undefined if a field of a node in the prototype definition is associated with more than one field in the prototype's interface (*i.e.*, multiple IS statements for a field in a node in the prototype definition), but multiple IS statements for the fields in the prototype interface declaration is valid. Results are undefined if a field of a node in a prototype definition is both defined with initial values (*i.e.*, field statement) and associated by an IS statement with a field in the prototype's interface. If a prototype interface has an outputOnly field  $E$  associated with multiple outputOnly fields in the prototype definition  $ED_i$ , the value of  $E$  is the value of the field that generated the event with the greatest timestamp. If two or more of the outputOnly fields generated events with identical timestamps, results are undefined.

#### 4.4.4.4 Prototype scoping rules

Prototype definitions appearing inside a prototype definition (*i.e.*, nested) are local to the enclosing prototype. IS statements inside a nested prototype's implementation may refer to the prototype declarations of the innermost prototype.

A PROTO statement establishes a DEF/USE name scope separate from the rest of the scene and separate from any nested PROTO statements. Nodes given a name by a DEF construct inside the prototype may not be referenced in a USE construct outside of the prototype's scope. Nodes given a name by a DEF construct outside the prototype scope may not be referenced in a USE construct inside the prototype scope.

A prototype may be instantiated in a file anywhere after the completion of the prototype definition. A prototype may not be instantiated inside its own implementation (*i.e.*, recursive prototypes are illegal).

#### 4.4.5 External prototype semantics

##### 4.4.5.1 Introduction

The EXTERNPROTO statement defines a new node type. It is equivalent to the PROTO statement, with two exceptions. First, the implementation of the node type is stored externally, either in an X3D file containing an appropriate PROTO statement or using some other implementation-dependent mechanism. Second, default values for fields are

not given since the implementation will define appropriate defaults.

#### 4.4.5.2 EXTERNPROTO interface semantics

The semantics of the EXTERNPROTO are exactly the same as for a PROTO statement, except that default field values are not specified locally. In addition, events sent to an instance of an externally prototyped node may be ignored until the implementation of the node is found.

Until the definition has been loaded, the browser shall determine the initial value of inputOutput fields using the following rules (in order of precedence):

- a. the user-defined value in the instantiation (if one is specified);
- b. the default value for that field type.

For outputOnly fields, the initial value on startup will be the default value for that field type. During the loading of an EXTERNPROTO, if an initial value of an outputOnly field is found, that value is applied to the field and no event is generated.

The names and types of the fields of the interface declaration shall be a subset of those defined in the implementation. Declaring a field with a non-matching name is an error, as is declaring a field with a matching name but a different type.

It is recommended that user-defined field names defined in EXTERNPROTO interface statements follow the naming conventions described in [4.4.2.2 Field semantics](#).

#### 4.4.5.3 EXTERNPROTO URL semantics

The string or strings specified after the interface declaration give the location of the prototype's implementation. If multiple strings are specified, the browser searches in the order of preference. For more information on URLs, see [9 Networking component](#).

If a URL in an EXTERNPROTO statement refers to an X3D file, the first PROTO statement found in the X3D file (excluding EXTERNPROTOS) is used to define the external prototype's definition. The name of that prototype does not need to match the name given in the EXTERNPROTO statement. Results are undefined if a URL in an EXTERNPROTO statement refers to a non-X3D file

To enable the creation of libraries of reusable PROTO definitions, browsers shall recognize EXTERNPROTO URLs that end with "#name" to mean the PROTO statement for "name" in the given X3D file. For example, a library of standard materials might be stored in an X3D file called "materials.x3dv" that looks like:

```
#X3D V3.0 utf8
PROTO Gold [] { Material { ... } }
PROTO Silver [] { Material { ... } }
...etc.
```

A material from this library **could** **might** be used as follows:

```
#X3D V3.0 utf8
EXTERNPROTO GoldFromLibrary [] "http://.../materials.x3dv#Gold"
...
Shape {
  appearance Appearance { material GoldFromLibrary {} }
  geometry ...
}
...
```



## 4.4.6 Import/Export semantics

The IMPORT feature allows authors to incorporate content defined within Inline nodes or created programmatically into the namespace of the containing file for the purposes of event routing. In contrast with external prototyping (see [4.4.5 External prototype semantics](#)), which allows access to individual fields of nodes defined as prototypes in external files, IMPORT provides access to all the fields of an externally defined node with a single statement (see [9.2.5 IMPORT statement](#)).

Importing nodes from an Inlined file is accomplished with two statements: IMPORT and EXPORT. The IMPORT statement is used in the containing file to define which nodes of an Inline are to be incorporated into the containing file's namespace. The EXPORT statement is used in the file being Inlined, to control access over which nodes within a file are visible to other files (see [9.2.6 EXPORT statement](#)). EXPORT statements are not allowed in prototype declarations.

## 4.4.7 Run-time name scope

Each X3D browser defines a run-time name scope that contains all of the root nodes currently contained by the scene graph and all of the descendant nodes of the root nodes, with the exception of nodes hidden inside another name scope. Prototypes establish a name scope and therefore nodes inside prototype instances are hidden from the parent name scope.

Each Inline node or prototype instance also defines a run-time name scope, consisting of all of the root nodes of the file referred to by the Inline node or all of the root nodes of the prototype definition, restricted as above. Other nodes or extension mechanism may be introduced which specify their own name scope.

The IMPORT feature allows nodes defined within files referenced from [Inline](#) nodes to be incorporated into the run-time name scope of the containing scene graph. Once an IMPORT statement has been encountered, the new name may be used exactly like any other node name for the purposes of routing or programmatic access (*i.e.*, may be used in ROUTE statements and accessed as a field from the Scene Access Interface). Names imported from an Inline shall be explicitly declared as exportable within the content of the inlined file, using the EXPORT statement; only names exported using the EXPORT statement are available to be imported into other run-time name scopes. The optional AS keyword allows a unique name to be assigned to the imported node in order to avoid name conflicts in the containing scene graph's run-time name scope.

Nodes created dynamically (using the X3D Scene Access Interface) are not part of any name scope, until they are added to the scene graph, at which point they become part of the same name scope of their parent node(s). A node may be part of more than one run-time name scope. A node shall be removed from a name scope when it is removed from the scene graph.

## 4.4.8 Event model

### 4.4.8.1 Events

*Events* are the primary means of generating behaviors in the X3D run-time environment. Events are used throughout X3D: driving time-based animations; handling object picking; detecting user movement and collision; changing the scene graph hierarchy. The run-time environment manages the propagation of events through the system according to a well-defined set of rules.

Nodes define input fields (*i.e.*, fields with `inputOutput` or `inputOnly` access) that trigger behavior. When a given event occurs, the node receives notification and can potentially change internal state and the value of one or more of its fields. Nodes also define output fields (*i.e.*, fields with `inputOutput` or `outputOnly` access) that are sent upon signal state changes or other occurrences within the node. Events sent to input fields and events sent by output fields are referred to collectively in ISO/IEC 19775 as *Events*.

TODO: determine whether we need to further elaborate this definition when considering external events.

#### 4.4.8.2 Routes

*Routes* allows an author to declaratively connect the output events of a node to input events of other nodes, providing a way to implement complex behaviors without imperative programming. When a routed output event is fired, the corresponding destination input event receives notification and can process a response to that change. This processing can change the state of the node, generate additional events, or change the structure of the scene graph. Routes may be created declaratively in an X3D file or programmatically via an SAI call.

Routes are not nodes. The ROUTE statement is a construct for establishing event paths between specified fields of nodes. ROUTE statements may either appear at the top level of an X3D file or inside a node wherever fields may appear. It can appear after its source or destination node and placing a ROUTE statement within a node does not associate it with that node in any way. A ROUTE statement does follow the name scoping rules as described in [4.4.7 Run-time name scope](#).

The type of the destination field shall be the same as the source type, unless a component or support level permits an extension to this rule.

Redundant routing is ignored. If an X3D file repeats a routing path, the second and subsequent identical routes are ignored. This also applies for routes created dynamically using the X3D SAI.

Nodes created through the X3D prototyping mechanism give authors an opportunity to create custom processing of incoming events. Events coming into a prototyped node through an interface field can be routed to internal nodes for processing, or routed to other interface fields for propagation outside the node. An author can also add programmatic processing logic to an interface field using the internal scripting support of the Script node.

#### 4.4.8.3 Execution model

Once a sensor or Script has generated an *initial event*, the event is propagated from the field producing the event along any ROUTEs to other nodes. These other nodes may respond by generating additional events, continuing until all routes have been



honoured. This process is called an *event cascade*. All events generated during a given event cascade are assigned the same timestamp as the initial event, since all are considered to happen instantaneously.

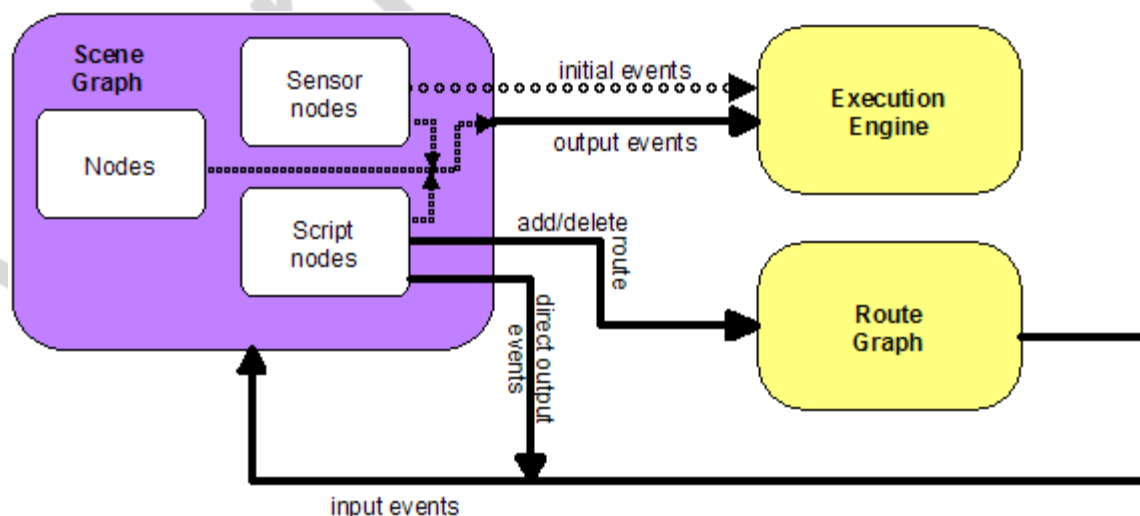
Some sensors generate multiple events simultaneously. Similarly, it is possible that asynchronously generated events **could** **might** arrive at the identical time as one or more sensor generated event. In these cases, all events generated are part of the same initial event cascade and each event has the same timestamp. The order in which the events are applied is not considered significant. Conforming X3D worlds shall be able to accommodate simultaneous events in arbitrary order.

After all events of the initial event cascade are honored, post-event processing performs actions stimulated by the event cascade. The browser shall perform the following sequence of actions during a single timestamp:

- Update camera based on currently bound Viewpoint's position and orientation.
- Evaluate input from sensors.
- Evaluate routes.
- If any events were generated from steps b and c, go to step b and continue.
- If particle system evaluation is to take place, evaluate the particle systems here.
- If physics model evaluation is to take place, evaluate the physics model.

For profiles that support [Script](#) nodes and the Scene Access Interface, the above order may have several intermediate steps. Details are described in [29 Scripting](#) and [2\[I.19775-2\]](#).

[Figure 4.3](#) provides a conceptual illustration of the execution model.



**Figure 4.3 — Conceptual execution model**

Nodes that contain output events shall produce at most one event per field per timestamp. If a field is connected to another field via a ROUTE, an implementation shall send only one event per ROUTE per timestamp. This also applies to scripts where the rules for determining the appropriate action for sending output events are defined in [29 Scripting component](#).

#### 4.4.8.4 Loops

Event cascades may contain *loops* where an event  $E$  is routed to a node that generates an event that eventually results in  $E$  being generated again. See [4.4.8.3 Execution model](#), for the loop breaking rule that limits each event to one event per timestamp. This rule shall also be used to break loops created by cyclic dependencies between different sensor nodes.

#### 4.4.8.5 Fan-in and fan-out

*Fan-in* occurs when two or more routes have the same destination field. All events are considered to have been received simultaneously; therefore, the order in which they are processed is not considered relevant.

*Fan-out* occurs when one field is the source for more than one route. This results in sending any event generated by the field along all routes. All events are considered to have been sent simultaneously; therefore, the order in which they are processed is not considered relevant.

#### 4.4.8.6 Internal/external event passing

TODO: describe how, when an external environment exists,

- Abstract definition of when events are be exchanged between external environment and scene graph.
- Essentially the external presentation event loop must complete each render/interaction cycle before passing events to a contained X3D scene, and
- Event loop for a contained X3D scene must complete each render/interaction cycle before passing events to an external presentation.

## 4.5 Components

### 4.5.1 Overview

An X3D component is a set of related functionality consisting of various X3D objects and services as described below.

Components are specified in this standard or may be defined elsewhere. This standard specifies a set of requirements which shall be satisfied for a component to be considered an X3D component. Components may be organized into support levels as provided by the component specification. The support levels are assigned an integer identifier starting with level 1 as the simplest support level. Higher numbered support levels (if specified) should incorporate all of the functionality of lower numbered support levels. Thus, the support levels support a hierarchy of functionality.

New components may be defined either through creation of a new part to this International Standard or through registration. Functionality may be added to an already defined component by amending the appropriate part of this International Standard or through registration. Such new functionality shall be in the form of one or more new levels that augment the functionality already provided. Levels already

defined shall not be subdivided. Each such addition shall satisfy the requirements for component definition stated above.

## 4.5.2 Defining components

The following are the requirements for defining components:

- a. All node objects within a component shall be derived, either directly or indirectly, from the [X3DNode](#) class.
- b. All field objects within a component shall be derived from the [X3DField](#) or [X3DArrayField](#) classes.
- c. The names for nodes and fields shall follow the naming semantics set forth in this standard including those for scoping.

Several components are defined in this standard as shown in the [Component index](#). These components are defined in their respective parts of this International Standard. In all cases, the X3D extension mechanism may be used to add new levels to the components or may be used to define separate new components.

Each component definition is comprised of:

- d. a name for the component suitable for use in the COMPONENT statement;
- e. one or more levels starting with Level 1;
- f. a list of prerequisites for the component (each prerequisite consisting of a statement of which level in which other component is required for support of the component being defined);
- g. a conceptual description of the functionality being provided;
- h. a definition of nodes being provided with an indication of in which level each node is; and
- i. a statement of conformance for the component.

## 4.5.3 Base components

Components are specified in this standard or may be defined elsewhere. See the [Component index](#) for a list of the components of X3D which have been formally accepted by the governing body.

Each component is presented by describing the functionality to be supported. This is followed by the specification of the abstract nodes of the component, if any. Following the abstract node specifications, the concrete nodes of the component are specified. Finally, the support levels are specified.

The support levels are specified in a table in which the first column presents the number of each support level. The second column specifies the prerequisite components that are required by the particular support level for the component being specified. Each new level is presented with its prerequisites in a separate row of the table. Subsequent rows until the next new level are used to specify node support for that level. The third column specifies the nodes and other features of the component that are to be supported, in whole or in part, by the indicated support level. The fourth column specifies any constraints on the particular feature or node for the indicated

support level. For each support level  $i+1$ , all features of the previous support level shall also be supported.

In the second column, each prerequisite for a support level is listed by a component name and a support level within that component. These table entries indicate that, for the browser to claim support for that level of the component, the browser implementation shall also support the component and support level(s) listed as a prerequisite. If there are no prerequisites, the word "None" is specified.

In the third column, abstract nodes introduced at that support level are listed first followed by the concrete nodes introduced at that support level.

In the fourth column, a listing of "n/a" means "not applicable". When it is indicated that a field is "optionally supported", an X3D browser is not required to support that field. If all fields of a node are to be entirely supported, the phrase "Full support" is used.

[Table 4.5](#) is an example of the format for a support level table:

**Table 4.5 — Example support level table**

Level	Prerequisites	Nodes/Features	Support
<b>1</b>	Core 1 Networking 2		
		<i>X3DTimeDependentNode</i> (abstract)	n/a
		Node1Name	fieldi optionally supported.
		Node2Name	All fields fully supported.
<b>2</b>			
		Level 1 nodes	All fields as supported by Level 1.
		NodeName	All fields fully supported.

Any new components defined by amendment or in new parts of this International Standard shall specify their functionality using the same format.

## 4.6 Profiles

### 4.6.1 Overview

ISO/IEC 19775 supports the concept of profiles. A profile is a named collection of functionality and requirements that shall be supported in order for an implementation to conform to that profile. Profiles are defined as a set of components and levels of each component as well as the minimum support criteria for all of the objects contained

within that set.

This part of ISO/IEC 19775 defines seven profiles satisfying varying sets of requirements:

- a. Core profile (see [Annex A](#))
- b. Interchange profile (see [Annex B](#))
- c. Interactive profile (see [Annex C](#))
- d. MPEG-4 interactive profile (see [Annex D](#))
- e. Immersive profile (see [Annex E](#))
- f. Full profile (see [Annex F](#))
- g. CADInterchange profile (see [Annex H](#))

Each set of requirements is directed at supporting the needs of a particular constituency. Not all constituencies may be satisfied by the functionality represented by these profiles. Therefore, this part of ISO/IEC 19775 allows for defining additional profiles either through amendment to this part of this International Standard or by registration.

A system that conforms to a given profile supports the full set of objects and capabilities defined for that profile.

## 4.6.2 Defining profiles

A profile definition consists of the following:

- a. a name for the profile suitable for use in the PROFILE statement;
- b. an introduction defining the purpose for the profile;
- c. a list of the components and levels within those components which comprise the profile;
- d. a statement of conformance criteria for the profile;
- e. a table containing the node type set supported by the profile stating the X3D File Limit and Minimum Browser Support for each node type;
- f. a table of other limitations for the profile; and
- g. any other information specific to the profile.

## 4.6.3 Relationship between profiles and components

A profile consists of a collection of components at given support levels. A user may also supplement the predefined set of components for a given profile by specifying extra component statements (see [7.2.5.4 COMPONENT statement](#)). If the user supplies additional component declarations in addition to the components and levels defined as part of the profile, the resultant components supported shall be the union of all components and levels requested. That is, a user cannot force a lower level of component conformance onto a profile by explicitly declaring the component with a lower level of support than that defined by the profile.

A profile definition shall be internally consistent. If a profile contains components that list prerequisites that are not covered by the component levels declared for that profile,

the prerequisites shall not be automatically made available. Authors wishing to use these missing prerequisites shall explicitly declare the component and level required through the use of the COMPONENT statement.

## 4.7 Support levels

The X3D specification may be supported at varying *Levels*, or qualities of service. Any X3D component may designate a level of service by using a numbering scheme in which higher-numbered levels denote increasing qualities of service. A higher level of service may indicate any of the following:

- a. The presence (or absence) of features;
- b. Improved support for a particular feature;
- c. More rigorously defined semantics; or
- d. More stringent conformance requirements.

Note that service levels between different features do not necessarily correspond. For example, a profile may contain one component supported at level 2 and another at level 1. Any profile may combine components defined at different service levels, provided that the features interoperate properly, the behavior is deterministic (within practical limits) and the conformance requirements for that profile and components are well-defined.

## 4.8 Data encodings

The X3D run-time architecture is independent of the data encoding format. X3D content and applications can be authored in a variety of encodings, including textual (XML and Classic VRML encodings) and binary, either compressed or uncompressed. ISO/IEC 19775 contains an abstract encoding specification that defines the structure of the X3D scene: hierarchical relationships among objects, initial values for objects, and dataflow connections between objects. All concrete data encodings for X3D shall conform to this abstract specification.

Browsers and generators may support any or all of the standard encoding formats, depending on their application needs and the conformance requirements of a specific component or profile.

X3D encodings are fully specified in the parts of [ISO/IEC 19776](#).

## 4.9 Scene access interface (SAI)

X3D provides a set of application programmer interfaces (APIs), called the Scene Access Interface (SAI), that defines run-time access to the scene. Using the SAI a developer may create and destroy nodes, send events to nodes, create connections between nodes (*routes*), read or set field values in nodes, traverse the scene graph, and control the operations of the browser. Programmatic access may be *internal* (*i.e.*, used to create customized elements within the scene graph) or *external* (*i.e.*, connecting to program elements outside the scene such as in a host application such as a web browser). Internal access is supported via a special node called a [Script](#) node.



Script nodes allow developers to connect programming language functions and object classes to the scene graph. Fields of a script are automatically mapped to properties and methods of the object associated with that script. Script node code may generate events which are propagated back to the scene graph by the run-time environment. External access is supported through integration between the X3D run-time system and a variety of programming language run-time libraries.

The X3D SAI is specified as a set of language-independent services and bindings to several programming and scripting languages. A complete specification of the X3D SAI services and the component model interfaces may be found in [2. \[19775-2\]](#). The language bindings for the services defined in ISO/IEC 19775-2 are specified in [2. \[19777\]](#). Internal programmatic access is enabled through the Script node, described in [29 Scripting component](#).

TODO: determine whether we need to further elaborate this definition when considering external environments.

## 4.10 Component and profile registration

This part of ISO/IEC 19775 allows new concepts to be defined by registration of components, new levels within components, and profiles. Registration shall not be used to modify any existing component, level of a component, or profile. New functionality is registered using the established procedures of the [ISO International Register of Items<sup>1\)</sup>](#). These procedures require the proposer to supply all information for a new registered item except for the level number. The level number (if applicable) is assigned and managed by the ISO International Registration Authority for Graphical Items. Registration shall be according to the procedures in [ISO/IEC 9973](#).

<sup>1)</sup>Contact information for the ISO-designated Registration Authority for Items registered under the ISO/IEC 9973 procedures is available at the ISO Maintenance Agencies and Registration Authorities web site: [http://www.iso.org/iso/standards\\_development/maintenance\\_agencies.htm](http://www.iso.org/iso/standards_development/maintenance_agencies.htm).



