An Effective Method for Specifying Projective Texture Mapping in X3D

In-Kwon Kim, Add (Brutzman, Puk), Jong-Sung Ha, Nakhoon Baek, Kwan-Hee Yoo

*Dept. of Digital Information Convergence, Chungbuk National University
bDept. of Computer Science, Chungbuk National University
cDept. of Game and Contents, Wooksuk University
eSchool of Computer Sci. and Eng., Kyungpook National University

Elsevier use only: Received date here; revised date here; accepted date here

Abstract

This paper presents the specification and implementation of a new functionality called projective texture mapping (PTM) in the Extensible 3D (X3D) that is the ISO standard for defining 3D interactive web-contents. PTM is a method projecting an image, also known as the texture image, onto a scene as if by a slide projector, which is useful for enhancing image quality in a variety of lighting techniques. Even though there have been attempts to specify PTM in X3D, users need to iteratively specify images and coordinates of textures on each object in a 3D scene. As a standard functionality of X3D, we suggest effective PTM methods to provide perspective and parallel projective textures, and the implementation results of PTM rendering in an open source X3D viewer called FreeWRL are shown.

Keywords: Projective texture mapping; X3D; scene graph

1. Introduction

The Extensible 3D (X3D) [1,2] is a royalty-free ISO standard file format for representing interactive 3D contents in the world-wide-web, based on the Extensible Markup Language (XML). This paper is concerned with developing a new texture mapping functionality called Projective Texture Mapping (PTM) in X3D. PTM is a specialized texture mapping method, which was first introduced by Everitt [3]. PTM actually allows a texture image to be projected onto objects as if projected by a slide projector. This technique is useful for various applications such as photo formation [4,5] as well as for enhancing image quality in rendering [6,7,8]. X3D represents the 3D virtual world with the scene graph that is a general data structure commonly used in the area of 3D computer graphics and Web environment. A scene graph is the ordered collection of fundamental components called node in a graph or tree structure. A node may have many children, with the effect of the parent node applied to all its children nodes.

Even though X3D includes various nodes for texturing features, none of them has supported PTM yet. Kamburelis [9] suggested a PTM method by using the technique of shadow mapping of Everitt [6,7] with additive PTM information. Kim et. al [10] also tried to support PTM with different methods defining an independent node for PTM information for Appearance node. These methods are inconvenient and exhaustive because users have to repeatedly define nodes for PTM information for each of the objects in a scene. In other words, PTM is processed in local for each object in the scene.

In order to resolve the inconvenience, this paper suggests an effective method in order to globally process
PTM with respect to all objects at once in a scene. And we also define two types of PTM nodes: perspective and parallel PTM nodes, still strictly obeying the existing hierarchy of X3D node structures. The two types of proposed PTM nodes have been implemented in X3D viewer of FreeWRL [11] by using OpenGL shader languages [12]. The implementation details and those results are presented in Section 4.

We will start with previous works in Section 2. The details of the PTM node design were presented in Section 3. Experimental results are followed in Section 4. Finally conclusions and future work are followed.

2. Related Works

In this section, we show a set of works directly related to PTM methods. Concepts and theoretical details are presented.

2.1. PTM Concept

PTM is a technique to project a texture image onto the surfaces of objects within the projection volume with a view from a certain spot called 3D scene projection point, as illustrated in Fig. 1. The projection volume is determined by projection parameters such projection point, direction, and aspect ratio, which was described in the process of PTM by Everitt [3].

![Fig. 1. Projective Texture Mapping[10]](image)

In the classical texture mapping techniques, 2D texture coordinate values with the range from 0 to 1 are mapped onto the vertices of 3D models. In order to understand PTM concept, we consider the object linear method in which a texture is fixed in an object space. The method computes the texture coordinate, \( T^i = (s, t, r, q) \) for a vertex, \( V^i = (x_0, y_0, z_0, w_0) \), by multiplying a concatenated matrix \( T_0 \) into the vertex, \( V^i \), as shown in the following Equations:

\[
\begin{bmatrix}
    s \\
    t \\
    r \\
    q
\end{bmatrix} = \begin{bmatrix}
    1/2 & 0 & 0 & 1/2 \\
    0 & 1/2 & 0 & 1/2 \\
    0 & 0 & 1 & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
    x_0 \\
    y_0 \\
    z_0 \\
    w_0
\end{bmatrix}
\]

with

\[
T_0 = \begin{bmatrix}
    1/2 & 0 & 0 & 1/2 \\
    0 & 1/2 & 0 & 1/2 \\
    0 & 0 & 1 & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]

where \( P_p \), \( V_p \), and \( M \) are transformation matrices for projection, camera view, and model, respectively.

However, the concept does not reflect one of projective texture mapping which allows a texture image to be projected onto the scene as if by a slide projector. Since the texture to be projected is fixed in an eye space with viewpoint, the method called eye linear, is used to the texture coordinates on an object in PTM. First, object coordinates are transformed into the eye space by the following equation:

\[
\begin{bmatrix}
    x_e \\
    y_e \\
    z_e \\
    w_e
\end{bmatrix} = \begin{bmatrix}
    C_C \\ C_Cr_C \\ C_CView \\ C_CMatrix
\end{bmatrix} \begin{bmatrix}
    x_0 \\
    y_0 \\
    z_0 \\
    w_0
\end{bmatrix}
\]

Then, a matrix called the eye linear generator is multiplied as follows:

\[
\begin{bmatrix}
    x_e \\
    y_e \\
    z_e \\
    w_e
\end{bmatrix} = \begin{bmatrix}
    X_e \\
    Y_e \\
    Z_e \\
    W_e
\end{bmatrix} \begin{bmatrix}
    I_C \ \ I_C \ \ I_C \ \ I_C
\end{bmatrix}
\]

The eye linear generator matrix \( T_s \) can be approximated as the following Equation:

\[
T_s = \begin{bmatrix}
    1/2 & 0 & 0 & 1/2 \\
    0 & 1/2 & 0 & 1/2 \\
    0 & 0 & 1 & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]

based on a projector instead of a camera in order to represent the projector's properties.

2.2. Previous PTM extensions with local effects

Kamburelis obtained PTM coordinates by extending the existing X3DLightNode with new fields describing the minimum and maximum distance of the projection and the up-vector of the light, as in Table 1 [9]. The obtained coordinates are used in the PTM of ProjectedTextureCoordinate node inheriting the existing X3DTextureCoordinateNode as in Table 2.

Since this method has the structure dividing the nodes into ProjectedTextureCoordinate and ImageTexture, PTM necessarily requires the interaction among the nodes of X3DLight, ProjectedTextureCoordinate, and ImageTexture.
Furthermore, users have to iteratively define the two nodes of `ProjectedTextureCoordinate` and `ImageTexture` for each of objects in a virtual scene.

Table 1. New fields in `X3DLightNode`.

<table>
<thead>
<tr>
<th>Field</th>
<th>Typ</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectionNear</td>
<td>SFFloat</td>
<td>0</td>
</tr>
<tr>
<td>projectionFar</td>
<td>SFFloat</td>
<td>0</td>
</tr>
<tr>
<td>up</td>
<td>SFBool</td>
<td>0 0 0</td>
</tr>
<tr>
<td>defaultShadowMap</td>
<td>SFBool</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Table 2. `ProjectedTextureCoordinate` Node

```
ProjectedTextureCoordinate : X3DTextureCoordinateNode {
  SFNode [in,out] projector NULL # [SpotLight, DirectionalLight, X3DViewpointNode]
}
```

Recently, Kim et al. [10] tried to resolve the limitations of Kamburelis’s method by defining a new node `ProjectiveTexture` as in Table 3. This method generates an independent projector node `PerspectiveProjector` as in Table 4, and divides the two nodes into `X3DLight` and `PerspectiveProjector` to obtain the PTM coordinates, while Kamburelis’s method exploits the node `X3DLight`.

Still the inconvenient problem has been also remained that users have to iteratively define two other nodes of `ProjectiveTexture` and `Appearance` for each of objects in a scene.

Table 3. `ProjectedTextureCoordinate` Node

```
ProjectedTextureCoordinate : X3DTextureNode {
  SFNode [in,out] metadata Null [X3DMetadataObject]
  SFString [in,out] projectorName ""
  SFBool  [in,out] value true
  MFString [in,out] url ""
}
```

Table 4. `PerspectiveProjector` Node

```
PerspectiveProjector : X3DChildNode {
  SFNode [in,out] metadata Null [X3DMetadataObject]
  SFString [in,out] description ""
  SFVec3f [in,out] centerOfProjection 0 0 0 (-∞,∞)
  SFVec3f [in,out] direction 0 0 1 (-∞,∞)
  SFFloat [in,out] fieldOfView 45
  SFFloat [in,out] aspectRatio 1
  SFFloat [in,out] nearFar 1 10
  SFVec3f [in,out] upVector 0 1 0
}
```

3. A New PTM Extension with Global Effects

Similar to the `X3DLight` node in the X3D node structure hierarchy, a PTM node needs to be defined without any violation of the original hierarchy in order to globally effect all objects in a scene at once. First, we define a new node `TextureProjectorNode` inheriting `ChildNode` as like the `LightNode` obtaining the global effect by inheriting the `ChildNode`, as illustrated in Fig. 2. Next, two other new nodes `TextureProjectorPerspective` and `TextureProjectorParallel` are defined by inheriting `TextureProjectorNode` that have the effects of perspective and parallel projection, respectively.

Table 5, 6, and 7 describe each definition of three node `TextureProjectorNode`, `TextureProjectorPerspective`, and `TextureProjectorParallel`, in which the last columns describe the default value of each field. `TextureProjectorNode` inherits `X3DChildNode` of the original X3D hierarchy, and again other two nodes `TextureProjectorPerspective` and `TextureProjectorParallel` inherits `TextureProjectorNode`.

In Table 5, the field “description” is a value identifying projective texture projectors arranged in 3D space. The projectors are located in the position of “location” with the viewing direction of “direction”. The field “aspectRatio” of the projectors cannot be given by users, but can be obtained from other fields. The fields of “nearDistance” and “nearDistance” represent the minimum and maximum distances that can be projected, respectively. The fields of “global” and “on” are defined to represents the characteristics of children nodes; if “global” is true and “on”
is false, the projector node effects only its children nodes, but it effects all objects within the projected volume if “global” is false and “on” is true. The 2D image to be projected is described in the field “texture”.

Table 5. X3DTextureProjectorNode

| X3DTextureProjectorNode:X3DChildNode {                     |
| SFNode [in,out] metadata Null                            |
| SFString [in,out] description ""                          |
| SFVec3f [in,out] location 0 0 0 (-∞,∞)                   |
| SFVec3f [in,out] direction 0 0 1 (-∞,∞)                  |
| SFFloat [in,out] aspectRatio                              |
| SFFloat [in,out] nearDistance 1                           |
| SFFloat [in,out] farDistance 10                           |
| SFBool [in,out] global                                    |
| SFBool [in,out] on                                       |
| SFNode [in,out] texture                                   |
| [X3DTexture2DNode]                                       |

In Table 6, two more fields are added in order to describe the properties of perspective projection; “fieldOfView” is the view angle of the projector, and “upVector” is the angle rotating along Z-axis.

Table 6. X3DTextureProjectorPerspective Node

| X3DTextureProjectorPerspective:X3DTextureProjectorNode { |
| SFFloat [in,out] fieldOfView π/4 (0,π)                   |
| SFFloat [in,out] aspectRatio                             |
|

Table 7, only a simple field is added in order to describe the properties of parallel projection; “fieldOfView” is the X-, Y-coordinate values of the screen.

Table 7. X3DTextureProjectorParallel Node

| X3DTextureProjectorParallel:X3DTextureProjectorNode {    |
| SFFloat [in,out] fieldOfView (-1,1,-1) (-∞,∞)          |
|

4. Implementation and Experiments

We implemented the rendering of newly defined PTM nodes with OpenGL shader language (GLSL) in FreeWRL.2015 that is an open source X3D viewer. The system environments for implementing and testing the projective texture mapping of X3D is described in Table 8.

Table 8. System Environments

<table>
<thead>
<tr>
<th>Software</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler</td>
<td>MS Visual Studio 2008</td>
</tr>
<tr>
<td>3D Graphics API</td>
<td>OpenGL Shader Language</td>
</tr>
<tr>
<td>X3D Viewer</td>
<td>FreeWRL (C Language)</td>
</tr>
<tr>
<td>Hardware</td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td>Intel(R) Core(TM) i7-4702MQ CPU @ 2.20GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>12GB</td>
</tr>
<tr>
<td>GPU</td>
<td>NVIDIA GeForce GT 740M</td>
</tr>
</tbody>
</table>

Fig. 3 illustrates the pipeline architecture of FreeWRL, in which, each node of X3D script files are parsed to produce a tree hierarchical structure, and then rendered by shader programs. As illustrated in Appendix A, FreeWRL can be extended with PERL programming to parse the new nodes: X3DTextureProjectorPerspective and X3DTextureProjectorParallel.

Table 9 is the implementation of OpenGL shader programs for rendering of the projective texture mapping in X3D. The uniform variable “projTexGenMatCam0” is the eye linear generator matrix generated in the parsing of nodes. The computed texture coordinate in the varying variable “projTexCoord” is passed to the next stage.
condition “projTexCoord.q > 0.0” checks the projection from the reverse of viewing direction.

Table 9. Shader Programs for Projective Texture Mapping

<table>
<thead>
<tr>
<th><strong>Vertex Shader</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>uniform mat4 projTexGenMatCam0;</td>
</tr>
<tr>
<td>uniform mat4 viewMat;</td>
</tr>
<tr>
<td>varying vec4 projTexCoord;</td>
</tr>
<tr>
<td>void vertProjCalTexCoordinate(void) {</td>
</tr>
<tr>
<td>mat invViewMat = invers(viewMat);</td>
</tr>
<tr>
<td>vec4 posEye = gl_ModelviewMatrix *gl_Vertex;</td>
</tr>
<tr>
<td>vec4 posWorld = invViewMat *pos_Eye;</td>
</tr>
<tr>
<td>projTexCoord = projTexGenMatCam0 *posWorld;</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Fragment Shader</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>varying vec4 projTexCoord;</td>
</tr>
<tr>
<td>vec4 projMapColor_forCam1;</td>
</tr>
<tr>
<td>void fragProjCalTexCoord(void) {</td>
</tr>
<tr>
<td>if (projTexCoord.q &gt; 0.0) {</td>
</tr>
<tr>
<td>projMapColor_forCam1</td>
</tr>
<tr>
<td>= texture2Dproj(fw_Texture_unit0, projTexCoord);</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

Table 10-13 shows the examples of X3D scripts using our PTM nodes, and their rendered results are shown in Fig. 4-7, respectively.

Table 10. Perspective Project of an Apple Image into a Plane

```
<X3D profile="Interactive" version="3.3">
  <Scene>
    <TextureProjectorPerspective description='pt1' location='3 3 3' direction='-1 0 -1' fieldOfView='15' 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>
```

Table 11. Perspective Project of an Apple Image into a Box

```
<X3D profile="Interactive" version="3.3">
  <Scene>
    <TextureProjectorPerspective description='pt1' location='3 3 3' direction='-1 -1 -1' fieldOfView='15' nearDistance='1' farDistance='10' upVector='0 1 0' global='true' on='true'>
      <ImageTexture url='C:/image/apple.jpg' repeatS='false' repeatT='false' />
    </TextureProjectorPerspective>
    <Shape>
      <Material diffuseColor='0.5 0.5 0.5'/>
    </Appearance>
    <Box/>
  </Scene>
</X3D>
```
Table 12. Perspective Project of an Apple Image into a Sphere and a Plane

```x3d
<Scene>
  <TextureProjectorPerspective
description='pt1' location='3 3 3' direction='-1 0 -1'
fieldOfView='15' 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 -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>
  <Transform translation='0,0.25,0'>
    <Shape>
      <Appearance>
        <Material diffuseColor='0.5 0.5 0.5'/>
      </Appearance>
      <Sphere radius='0.5'/>
    </Shape>
    <Transform>
      <Shape>
        <Appearance>
          <Material diffuseColor='0.5 0.5 0.5'/>
        </Appearance>
        <Sphere radius='0.5'/>
      </Shape>
    </Transform>
  </Scene>
</X3D>
```

Fig. 5. Rendered Result of Table 11

Table 13. Perspective Project of an Apple Image into two Planes

```x3d
<Scene>
  <TextureProjectorPerspective
description='pt1' location='3 3 3' direction='-1 0 -1'
fieldOfView='15' 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 -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>
  <Transform translation='0,0.25,0'>
    <Shape>
      <Appearance>
        <Material diffuseColor='0.5 0.5 0.5'/>
      </Appearance>
      <Transform translation='0,0.25,0'>
        <Shape>
          <Appearance>
            <Material diffuseColor='0.5 0.5 0.5'/>
          </Appearance>
          <Sphere radius='0.5'/>
        </Shape>
      </Transform>
    </Shape>
  </Transform>
  <Shape>
    <Appearance>
      <Material diffuseColor='0.5 0.5 0.5'/>
    </Appearance>
    <Sphere radius='0.5'/>
  </Shape>
</Scene>
</X3D>
```

Fig. 6. Rendered Result of Table 12
5. Conclusion

In this paper, we provide two new X3D nodes, to support projective texture mapping (PTM), in the web contents. These PTM nodes can be naturally integrated into the exiting hierarchy of X3D, through extending the parser capability of X3D viewers such as of FreeWRL 2015 and implementing the rendering shader programs. Since these PTM nodes globally effect all objects in the scene at once, we could improve the efficiency in the previous solutions of [9] and [10], through separately creating two nodes per each appearance node of multiple objects.

Acknowledgements

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2014R1A1A2055379) and by Woosuk University

References


Appendix A. The Parser written in PERL language

A.1. TextureProjectorPerspective node

```perl
TextureProjectorPerspective => new VRML::NodeType("TextureProjectorPerspective","{ "metadata => [SFNode, NULL, inputOutput, "SPEC_X3D30" | "SPEC_X3D31" | "SPEC_X3D32" | "SPEC_X3D33"],
description => [SFString, "", inputOutput, "SPEC_VRML" | "SPEC_X3D30" | "SPEC_X3D31" | "SPEC_X3D32" | "SPEC_X3D33"],
location=> [SFVec3f, [0, 0, 1], inputOutput, "SPEC_VRML" | "SPEC_X3D30" | "SPEC_X3D31" | "SPEC_X3D32" | "SPEC_X3D33"],
direction => [SFVec3f, [0, 0, 1], inputOutput, "SPEC_VRML" | "SPEC_X3D30" | "SPEC_X3D31" | "SPEC_X3D32" | "SPEC_X3D33"],
upVector => [SFVec3f, [0, 1, 0], inputOutput, "SPEC_VRML" | "SPEC_X3D30" | "SPEC_X3D31" | "SPEC_X3D32" | "SPEC_X3D33"],
fieldOfView => [SFFloat, 45, inputOutput, "SPEC_VRML" | "SPEC_X3D30" | "SPEC_X3D31" | "SPEC_X3D32" | "SPEC_X3D33"],
aspectRatio => [SFFloat, 1, inputOutput, "SPEC_VRML" | "SPEC_X3D30" | "SPEC_X3D31" | "SPEC_X3D32" | "SPEC_X3D33"],
nearDistance => [SFFloat, 1, inputOutput, "SPEC_VRML" | "SPEC_X3D30" | "SPEC_X3D31" | "SPEC_X3D32" | "SPEC_X3D33"],
farDistance => [SFFloat, 10, inputOutput, "SPEC_VRML" | "SPEC_X3D30" | "SPEC_X3D31" | "SPEC_X3D32" | "SPEC_X3D33"],
global => [SFBool, FALSE, inputOutput, "SPEC_X3D30" | "SPEC_X3D31" | "SPEC_X3D32" | "SPEC_X3D33"],
on => [SFBool, FALSE, inputOutput, "SPEC_X3D30" | "SPEC_X3D31" | "SPEC_X3D32" | "SPEC_X3D33"],
}
";
```

Fig. 7. Rendered Result of Table 13
A.2. TextureProjectorParallel node

```
TextureProjectorParallel => new VRML::NodeType("TextureProjectorParallel",{
    metadata => [SFNode, NULL, inputOutput, "(SPEC_X3D30 | SPEC_X3D31 | SPEC_X3D32 | SPEC_X3D33)"],
    description => [SFString, "", inputOutput, "(SPEC_VRML | SPEC_X3D30 | SPEC_X3D31 | SPEC_X3D32 | SPEC_X3D33)"],
    location => [SFVec3f, [0, 0, 1], inputOutput, "(SPEC_VRML | SPEC_X3D30 | SPEC_X3D31 | SPEC_X3D32 | SPEC_X3D33)"],
    direction => [SFVec3f, [0, 0, 1], inputOutput, "(SPEC_VRML | SPEC_X3D30 | SPEC_X3D31 | SPEC_X3D32 | SPEC_X3D33)"],
    upVector => [SFVec3f, [0, 1, 0], inputOutput, "(SPEC_VRML | SPEC_X3D30 | SPEC_X3D31 | SPEC_X3D32 | SPEC_X3D33)"],
    fieldOfView => [SFColorRGBA, [-1, 1, -1, 1], inputOutput, "(SPEC_VRML | SPEC_X3D30 | SPEC_X3D31 | SPEC_X3D32 | SPEC_X3D33)"],
    aspectRatio => [SFFloat, 1, inputOutput, "(SPEC_VRML | SPEC_X3D30 | SPEC_X3D31 | SPEC_X3D32 | SPEC_X3D33)"],
    nearDistance => [SFFloat, 1, inputOutput, "(SPEC_VRML | SPEC_X3D30 | SPEC_X3D31 | SPEC_X3D32 | SPEC_X3D33)"],
    farDistance => [SFFloat, 10, inputOutput, "(SPEC_VRML | SPEC_X3D30 | SPEC_X3D31 | SPEC_X3D32 | SPEC_X3D33)"],
    global => [SFBool, FALSE, inputOutput, "(SPEC_X3D31 | SPEC_X3D32 | SPEC_X3D33)"],
    on => [SFBool, FALSE, inputOutput, "(SPEC_VRML | SPEC_X3D30 | SPEC_X3D31 | SPEC_X3D32 | SPEC_X3D33)"],
    texture => [SFNode, NULL, inputOutput, "(SPEC_VRML | SPEC_X3D30 | SPEC_X3D31 | SPEC_X3D32 | SPEC_X3D33)"],
    _dir => [SFVec4f, [0, 0, 0, 1], initializeOnly, 0],
    _loc => [SFVec4f, [0, 0, 1, 0], initializeOnly, 0],
    _upVec => [SFVec4f, [0, 0, 0, 1], initializeOnly, 0],
    __projTexture => [SFNode, NULL, inputOutput, 0],
    }, "X3DGroupingNode"),
```