



X3D Large Scale Terrain Rendering Extensions

Alan D. Hudson, Justin Couch, Stephen N. Matsuba

Abstract

Traditional X3D modeling is not well suited for large-scale geometry modelling where the entire model contains gigabytes of data. This paper proposes a set of nodes that extend the existing geospatial component to add the ability to stream geometry to the scene based on the user's current position, allow for dynamic and configurable displays based on the available source(s) while providing the browser vendor the opportunity to implement highly efficient terrain-specific rendering capabilities.

Background

Large scale terrain rendering requires a very different set of tactics compared to the traditional X3D model. In the traditional model, all the data to be rendered is directly contained in the file and any referenced files (eg inlines, textures etc). Once the file is read, everything is known. In the geospatial world, this can potentially lead to unmanageable file sizes in the order of terabytes. Just downloading the file itself could take hours or days. Those large files provide all the data, but in reality, the rendering only makes use of a very small subset of it at any one time.

In traditional 2D applications rendering of large-scale terrain data is handled by a very specialised application called a Geospatial Information System, or more commonly known as a GIS. A GIS is responsible for all the terrain management tasks, such as reading the files into memory, stitching maps together, filtering geolocated points of interest based on user-set filters as well as level of detail management. For example, when up in space, sub-meter resolution data is more dense than the pixels on screen and thus it is pointless requesting and using it. The GIS will filter the data to an appropriate level, handing it to the user's application to render. A GIS may also be embedded directly into the application as the primary drawing surface. Some GIS systems have their own 3D renderer, so for the purposes of this paper, we will ignore this functionality as it is not relevant to the X3D market.

In the 3D world, there are at least half a dozen commonly used rendering techniques for large scale terrains. Each technique requires the underlying data to be fetched in its own unique form. A simple grid of data points like an ElevationGrid is not always the most suitable way for these rendering engines to work. In addition, the level of information needed for collision detection of objects with the terrain is different to the detail needed for the terrain rendering itself. Each of these requests can be localised, greatly reducing the amount of data that needs to be fetched over the network.

All of these requirements point to the need for X3D to evolve a system where it must be able to handle streamed data, yet maintain compatibility with X3D's traditional design philosophy. The design of such a system should also support the browser being able to scale the content's detail based on the individual system that it finds itself installed on. While the content developer has the option of asking and instructing the browser on the type(s) of content to display, the browser has the option of filtering it in order to maintain a

reasonable level of performance for the chosen terrain rendering implementation.

Concepts

We start with the basic premise that in any form of rendering, the closer you are to an object, the more detail that is desired – the currently bound viewpoint defines the center of the highest detailed information to be displayed within that layer. This information is used by the browser implementation to interact with the underlying geospatial data source to access the appropriate amount and detail of data.

The browser fetches information based on the location of the currently bound viewpoint. A browser shall be capable of using both `GeoViewpoint` and `Viewpoint` as input for determining what data needs to be fetched and according to its specific rendering strategy. In addition, it may use elevation information to control the level of detail of information that it retrieves and renders. For example, being in space may only require 100Km resolution data, but on the ground would use 1m data. The browser is free to choose the level of detail it feels is appropriate to the location of the currently bound viewpoint, while also attempting to fulfill the requested minimums of the user. This requirement does not require the browser to use this resolution to the full visible limit. Most geospatial rendering algorithms selectively filter data to reduce resolution the further away it is from the current viewing location. This requirement only applies to the near-field viewing.

These nodes do not define a `GeoOrigin` like their non-streamed relatives. As data is brought into the system, the position of the viewer is defined to be an implicit `GeoOrigin`, so that calculations are always most accurate around the current camera position. This will reduce many of the jitter problems that can become apparent in the existing geospatial nodes when navigating far from the `GeoOrigin` that was encountered when the user first loaded the world.

The user may optionally provide advice on the minimum acceptable data resolution and bounds. When the data source is capable of providing data of at least this resolution, then the browser shall be required to fetch and render data of at least that resolution. However, it is acknowledged that not all data sources are equal and sometimes data may be available, but at a lesser resolution than that requested. When this is the case, the browser may ignore the minimum requirements and fetch the best resolution data that it can access. Some data rendering at a lower resolution is always to be considered better than none at all.

The spatial area covered by the data should be determined by the browser, but informed by hints on the node. When the user does not provide any hints, then the browser should use the visibility limit of the currently bound `NavigationInfo` to set the bounds of data that should be rendered. It is expected that the browser will typically have more data beyond this limit as part of its internal caching strategy for high performance rendering engine, this is just about defining the currently visible limits. If the visibility limit is set to infinite, the browser is free to choose its own bounds (within reason) based on the performance criteria hints that the user may provide.

Geospatial sources of data represent their contained data in one of two forms – bitmaps or vectors. Bitmap data is used to cover every section of the nominated space with some form of meaningful data. The most common example of this is vegetation data. Each pixel of the bitmap corresponds to a vegetation type at that given grid square (where the size of the grid square is a property of the underlying data source). Vector data is used to represent information that only travels between given points. Examples of data in this form

are political boundaries and road centerlines. A 2D rendering of this information typically does not allow for much stylistic variance. The 3D world, however, has many different ways of rendering them. Vegetation information may be rendered as the raw bitmap, a set of splatted textures, fixed 3D models or an intelligent modeling algorithm. Which of these options to use is highly dependent on the individual user's machine. A really fast machine could easily handle the full 3D geometry, but an old, obsolete machine would barely be able to handle a simple textured model. To cater for both ends of the spectrum, this proposal takes the approach of not letting the content author explicitly state what to use, only to provide performance hints to the browser. With geospatial rendering, it is trivially easy to grind an end-user's computer to a standstill with very simple bad design choices that "works OK on my computer". Here we make use of the X3D design philosophy of letting the end user choose what works best for them, not what the content author forces them to have.

Browser Hint Properties

Despite allowing the browser almost full control over what is being rendered, the user still needs the ability to give the browser hints about what is considered the preferred optimisation strategy. One application may want to focus on speed, another on detail etc. A new browser property is defined that can be used, along with the given sets of values:

```
GEOSPATIAL_RESOLUTION: "SPEED", "DETAIL", "FIXED_FRAME_RATE"
```

Optimisation for speed tells the browser that it is OK to drop spatial resolution in order to keep the frame rates as high as possible. It is mostly likely that the browser will only use the minimum resolution terrain data sources and not provide any detailed model rendering.

Optimisation for detail tells the browser to favour using 3D models for the overlays rather than simple lines and textures. Terrain detail nearest the viewpoint will be higher, and the detail falloff will be further from that location.

Optimisation for fixed frame rates allows the browser to raise and lower the detail levels so that a constant frame rate is achieved. Typically this is used in simulator-style or immersive systems where the goal is to render at a constant frame rate, such as 30 frames per second.

Nodes

The following collection of nodes are proposed to provide streamed geospatial extensions. These nodes would be added as part of the Geospatial Component at level 2.

Abstract data types

```
X3DGeospatialStreamedObject {
    SFFloat [in,out] minimumResolution 0 [0,∞)
    MFString [] geoSystem ["GD","WE"] [see 25.2.3]
    SFString [] dataType "" ["VECTOR" | "RASTER" | .... ]
    SFString [] layerType ""
    MFString [] source []
}
```

A streamed object provides data to the rendering system on the fly as the content is being

rendered. It does not specify the protocol, method or strategy to be used to fetch this data. The browser implementation chooses a method appropriate to the source and its own implementation of large-scale terrain rendering strategies. For example, if a browser implements the ROAM strategy then it would need to interact with the underlying source as sets of tiled data, while using a CLOD strategy would require a continuous stream of input.

The `dataType` field defines the type of data that can be expected from this streamed source. It allows the containing node to decide whether this source is applicable to its use. For example using raster data as an input for an indexed line set's Coordinate node would not be useful or usable. Two types are defined for the value, with other options available on an implementation-specific basis:

“VECTOR” represents streamed data as 3D coordinates.

“RASTER” represents streamed data as pixel data in a rectangular grid.
The default value for this field is set by the concrete node definitions.

The `layerType` field provides an informative description of the data that this source represents. It could be used in a user interface or just as a unique way of identifying the specific data to be used. In addition, the browser implementation may choose to apply additional semantics based on this layer definition. For example it may choose to render vegetation data using 3D models rather than just as a raster overlay. River or road data may be treated in a similar fashion. If the browser chooses to interpret this field in this way then it is recommended that user interface options as well as browser options be provided to select which to show. The following type values are defined to have specific meaning:

"Vegetation",
"RoadCenterLine",
"River",
"CoastLine",
"Powerline",
"Political",
"NationalBoundary",
et cetera.

The source type string lists a section of source locations that this content recommends fetching data from in order of preference. The protocols used to request and interpret the returned data are not specified. The browser may choose to use all sources in order to fulfill the requests. Each source may only have information from a specific part of the world, so a browser is free to pull data from any source in order to fulfill the required data information. A special URL protocol—`localgeo`—states that the browser may optionally choose to fetch the data from any local source or other internal implementation-specific source that it knows it has access to.

The `minimumResolution` field defines the user-required minimum data resolution to be rendered. The resolution is specified in meters. A value of zero states that the user does not care what is available and the browser is free to choose a resolution based on some implementation-specific reasoning.

```

GeoStreamedOverlay : X3DNode, X3DGeospatialStreamedObject {
    SFBool    [in,out] enabled           TRUE
    SFNode    [in,out] metadata         NULL          [X3DMetadataObject]
    SFFloat   [in,out] minimumResolution 0              [0,∞)
    MFString  [in,out] stylePreference  ""              ["CENTERLINE", "MODEL", "DECAL"]
    SFString  []      dataType           "RASTER"       ["VECTOR" | "RASTER" | ... ]
    MFString  []      geoSystem          ["GD", "WE"]   [see ISO 19775-1, Part 1: 25.2.3]
    SFString  []      layerType         ""
    MFString  []      source             []
}

```

This node represents one of the overlays that are available to be rendered. Typically this node will be generated as output from the `GeoSourceManager` and then passed directly to the `GeoStreamedElevationGrid`. The `enabled` field defines whether this overlay should be currently rendered. Data can still be fetched, but rendering is not performed.

The `stylePreference` field defines the user-preferred rendering style to be used for this layer, in order of preference. For example, this may be used to instruct the renderer to take road centerline vector data and render it as a decaled texture over the terrain rather than just a line set describing the centerline. Three values are defined by the specification. Implementations are permitted to provide other values.

"CENTERLINE": Render the data as a set of line. Only applicable if the data type is vector. Has no meaning for raster data.

"MODEL": Render the data as full 3D models. For example, vegetation raster map is turned into algorithmically generated tree, grass and flower models.

"DECAL": Lay the data over the base elevation grid as a decaled texture.

Vector data has an appropriate image selected, raster data is directly mapped using an implementation-specific technique.

```

GeoStreamedCoordinate : X3DCoordinateNode,
                        X3DGeospatialStreamedObject {
    SFNode    [in,out] metadata         NULL          [X3DMetadataObject]
    SFFloat   [in,out] minimumResolution 0              [0,∞)
    SFString  []      dataType           "VECTOR"       ["VECTOR" | "RASTER" | ... ]
    MFString  []      geoSystem          ["GD", "WE"]   [see ISO 19775-1, Part 1: 25.2.3]
    SFString  []      layerType         ""
    MFString  []      source             []
}

```

This node describes a set of streamed geospatial coordinate data. It can be used as input for standard geometry such as line and triangle sets. The geometry values are provided based on the user's current location.

When using this node, it is recommended that all other geometry input is not provided and the default automatically generated normals, colors and texture coordinates are used. If these other fixed types are used, the behavior is undefined.

```

GeoStreamedElevationGrid : X3DGeometryNode,
                           X3DGeospatialStreamedObject {
    SFBool    [in,out] enableBase       TRUE
    SFNode    [in,out] metadata         NULL          [X3DMetadataObject]
    SFFloat   [in,out] mimimumRange     0              [0,∞)

```

```

    SFFloat [in,out] minimumResolution 0 [0,∞)
    MFNode [in,out] overlays [] [GeoStreamedOverlay]
    SFDouble [] creaseAngle 0 [0,∞)
    SFString [] dataType "RASTER" ["VECTOR" | "RASTER" | ... ]
    MFString [] geoSystem ["GD","WE"] [see ISO 19775-1, Part 1: 25.2.3]
    SFString [] layerType ""
    MFString [] source []
}

```

This node represents an elevation grid that sources its geometry from an external stream. Resolution, normals and texture coordinates are assumed to be calculated on the fly. Although this method can work with the normal appearance nodes, it is most likely that it will be better to use the overlay capability to provide the dynamic streamed texturing rather than rely on an external mechanism.

The overlay field defines the current set of geometry that can be rendered over the top of the elevation as decals. The declaration order defines their rendering order. Index 0 is rendered closest to the terrain, and index n is rendered furthest from the terrain.

The `creaseAngle` field is used to determine when to smooth shade or hard edge, as per the other geometry types.

The `minimumRange` field defines the minimum acceptable viewable distance that the user wishes to have. It also defines a linear distance in the local ground plane coordinates axes from the user's current location. If this value is less than the `visibilityLimit` of the currently bound `NavigationInfo`, then the browser may use this value to guide how much terrain geometry needs to be loaded and managed. If this value is greater than the current `visibilityLimit`, then it may be ignored by the browser. This defines the minimum range, so that if the browser determines that it can support greater distance, it may choose to render more terrain than is suggested by this. Unless there is no underlying data available, the browser shall always render at least this amount of geometry.

```

GeoSourceManager {
    SFNode [in,out] metadata NULL [X3DMetadataObject]
    MFString [out] layerTypes
    MFString [out] dataTypes
    MFString [] source []
}

```

This node represents the information that the browser is able to determine from the provided sources. This can be used by application code to configure on the fly what the user can build in their world. For example, a script could be written that queries this manager output and uses it to configure the overlays.

```

GeoDragSensor : X3DDragSensorNode {
    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
    SFVec3f [out] trackNormal_changed
    SFVec3d [out] hitGeoCoord_changed
    SFNode [] geoOrigin NULL [GeoOrigin]
    MFString [] geoSystem ["GD","WE"] [see ISO 19775-1, Part 1: 25.2.3]
}

```

This node creates a drag sensor that follows the surface of the underlying terrain. When a drag is in progress, the output is determined by the intersection point of the terrain and the input device. The normal output is the surface normal at that intersection point.

The `autoOffset` field is ignored for this node as it is derived from the `X3DDragSensorNode` definition.

`trackPoint_changed` will generate something in the local 3D coordinate that has no geospatial reference. However, the authors feel that it is odd to have this component with these geospatial nodes as it is on `GeoTouchSensor`.

`geoOrigin` is provided for compatibility with the fixed geospatial nodes. Perhaps we could modify Level 1 to add this node since it is not required for streaming capability.

Closing Remarks

This proposal defines a set of nodes and concepts that allow for streaming geospatial data to a scene. It strikes a balance between content author configurability, end user system capabilities and the browser implementation burden. It explicitly avoids definition of networking protocols or interactions between the browser and the underlying data source of the geospatial data. There are many potential options for this capability, both open standards and proprietary, and thus the realm of a separate discussion.