Modeling the Digital Earth in VRML

Martin Reddy[*a], Yvan G. Leclerc[a], Lee Iverson[a], Nat Bletter[a], Kiril Vidimce[b]

[a]SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025.
[b]Dept. of Computer Science, Mississippi State University, MS 39762

ABSTRACT

This paper describes the representation and navigation of large, multi-resolution, georeferenced datasets in VRML97. This requires resolving nontrivial issues such as how to represent deep level of detail hierarchies efficiently in VRML; how to model terrain using geographic coordinate systems instead of only VRML's Cartesian representation; how to model georeferenced coordinates to sub-meter accuracy with only single-precision floating point support; how to enable the integration of multiple terrain datasets for a region, as well as cultural features such as buildings and roads; how to navigate efficiently around a large, global terrain dataset; and finally, how to encode metadata describing the terrain. We present solutions to all of these problems. Consequently, we are able to visualize geographic data in the order of terabytes or more, from the globe down to millimeter resolution, and in real-time, using standard VRML97.

Keywords: 3-D geography, terrain visualization, multi-resolution, georeferencing, virtual reality, VRML, Java.

1. INTRODUCTION

The problem of accurately representing geographic data is one that presents the Virtual Reality Modeling Language (VRML) with perhaps one of its greatest challenges: it requires the ability to display massive quantities of geometry and imagery, support numerous alternative coordinate systems, implement application-specific navigation schemes, and model data to very high precision.

The importance of geographic representations was recently highlighted by Al Gore, Vice President of the United States, in a speech that he made to the California Science Center entitled, "The Digital Earth"[1]. In this speech, Gore challenges the scientific community to build a "multi-resolution, three-dimensional representation of the planet, into which we can embed vast quantities of geo-referenced data". Gore proceeds to detail various issues and desirable features of such a representation, including the necessity to

1. Integrate presently available georeferenced data
2. Model the world to 1 m resolution
3. Distribute all data across thousands of servers and organizations
4. Support many kinds of data, e.g. imagery, buildings, weather, etc.
5. Build potentially on key Web and Internet standards
6. Develop metadata standards to describe the georeferenced information

Here, we attempt to illustrate how VRML can be used to model the digital earth. Much of the work that we have performed in this domain has been contributed to the GeoVRML Working Group (http://www.ai.sri.com/geovrml/). GeoVRML is an official working group of the Web3D Consortium that was formed with the specific goal of producing methods and tools for representing geographic data using VRML. The group is currently producing a GeoVRML 1.0 recommended practice document and open source sample implementation which is scheduled to be submitted to the Web3D Consortium by the end of 1999. The work presented in this paper forms a fundamental basis of the GeoVRML 1.0 deliverable. We have currently implemented all of the material that is presented in this paper, except for some of the navigation techniques presented in Section 6.

[*] Correspondence: Email: reddy@ai.sri.com; WWW: http://www.ai.sri.com/digital-earth

2. PREVIOUS WORK

VRML has drawn a lot of interest from various sectors involved in the creation and visualization of geographic information. For example, the Virtual Field Course (VFC) at the University of Leicester, UK, utilizes VRML and Java to present students with 2-D and 3-D views of fieldwork locations to enhance their cognition of the real environment[2]. The Naval Postgraduate School (NPS), in association with the US Geological Survey (USGS), has developed a multi-resolution model of the Monterey Bay in VRML, using raw bathymetry (elevation below sea level) data for a 2.5×2.5 degree region of the bay. Researchers at NASA Goddard Space Flight Center have produced several VRML terrain models, including a visualization of Hurricane Linda off the west coast of Mexico (http://vrml.gsfc.nasa.gov/). Abernathy and Shaw presented work to visualize the course of a 1997 relay race through the San Francisco Bay Area, using VRML to model the terrain and to overlay a track obtained from Global Positioning System (GPS) receivers[3]. The Image Processing Group at the University of Zagreb developed an interactive map of Croatia, where users could select a region and then view this in 3-D, using VRML.

For the most part, many of the VRML terrain models that have been produced are not multi-resolution, and hence are limited in the amount of geometry and imagery that can be effectively displayed. In addition, no work has yet faced the issue of modeling large-scale areas to very high precision, such as the world to 1 m resolution. In addition, little effort has been made to use or preserve the original georeferenced data in the VRML terrain representations. We address all of these issues in this paper and present solutions that we hope will improve the generality and fidelity of geographic support in VRML.

3. TERRAIN REPRESENTATION

Many current VRML applications proudly advertise the fact that they have been developed in under 100 K and hence can be quickly downloaded; also that they are geometrically noncomplex and hence the browser can render them at interactive frame rates. In our problem domain, we have massive volumes of complex data, but we still desire minimal download times and high frame rates. For example, storing color imagery for the entire world at 1 m resolution would require over one petabyte (10^{15} bytes) of backing store. If we had elevation data for the world at 30 m resolution, this would produce a geometric model of over 500 billion (5×10^{11}) polygons. Obviously, it is therefore essential for us to implement some form of level of detail (LOD) management.

3.1. Multi-resolution Terrain

LOD is best achieved for terrain applications using a hierarchical data structure such as a quad-tree[4,5,6]. This involves progressively downsampling an image or elevation bitmap to produce a multi-resolution pyramid. Each level of this pyramid is then segmented into a grid of equally-sized rectangular tiles, e.g. 128 x 128 pixels. A tile at one level of the pyramid will therefore map onto four tiles on the immediately higher-resolution level, i.e. the tiles at the higher-resolution level cover half the geographical area of the former. Using such a representation, we can progressively display higher resolution data around some area of interest (e.g. the user's viewpoint) while other regions remain in low resolution. The use of tiling also allows us to only fetch and display sections of the dataset that are visible from a certain vantage point. These concepts are illustrated in Figure 1.

3.2. Tree Files and Terrain Tile Files

We implement the above terrain representation by introducing two primary types of VRML files: Tree files and Terrain Tile files. These are the basic building blocks of our VRML terrain representation.

Tree files : These implement part of the multi-resolution structure for the entire globe. In effect, these files are the glue that holds the Terrain Tile files into a global quad-tree structure. The Tree files enable us to split the entire LOD hierarchy over multiple files and to abstract the LOD structure from the actual terrain data. It is also possible for us to create different LOD tree depths for different regions of the globe. For example, we could have 100 km resolution data for the entire globe but recursively insert higher-resolution datasets for smaller regions of interest, e.g. a 1 km resolution dataset for the conterminous United States and a 1 m resolution dataset for Yosemite Valley, CA. We store these files using the file and directory name convention "*dataset*/trees/*r*/p*x*p*y*.wrl", where *r* is the Tree file resolution, and (*x*,*y*) specify the coordinates of the Tree file in that level.

Terrain Tile files : These contain the actual terrain data for a single tile of a pyramid. This includes the elevation geometry and links to the texture-map imagery for that specific tile of terrain. There will exist a separate pyramid of Tile files for each terrain dataset. Using the previous example, the 1 km U.S. dataset and the 1 m Yosemite dataset will each have its

own pyramid of Tile files. We use the following file and directory name conventions for Tile files: "*dataset*/vtile/*n*/p*x*p*y*.wrl", where *n* is the level of the pyramid (0..n), and (*x*,*y*) specify the coordinates of the Terrain Tile file at that pyramid level.

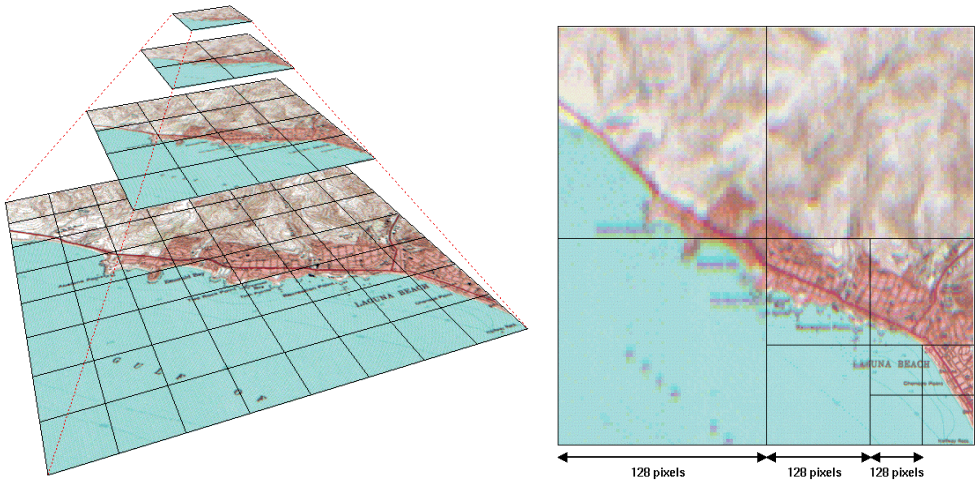

Figure 1: Illustrating a tiled pyramid representation. The left image shows four different resolutions of a digital map where each level has been segmented into a regular grid of tiles. The right image demonstrates the use of a quad-tree technique to alter the resolution of an image in different regions.

4. FUSION OF MULTIPLE DATASETS

One of the principal goals of our work is to represent multiple types of georeferenced data for any given region. For example, we desire the ability to switch between alternative terrain imageries; to display cultural features such as roads, buildings, and lines of communications; to visualize weather phenomena such as clear air turbulence isosurfaces or wind vectors; and to provide terrain annotations such as a city name or a mountain height (see Figure 2).

4.1. The GeoTile Node

In order to support this functionality, we introduce an extra layer of abstraction into our terrain representation. This is done using a new node called GeoTile that is included in each Tree file. A GeoTile contains descriptions and links to all available data for a single region at one level of detail. The node simply contains fields to specify a list of alternative terrain URLs (Universal Resource Locators) for that tile, and a list of URLs for all cultural or other features that extend into that tile (along with fields to provide short textual descriptions of each URL). With this provision, we can embed arbitrary amounts of georeferenced data into our terrain representation.

By adding this extra layer to our structure, we also simplify the task of maintaining and adding new datasets into our global structure. For example, when we want to add a new image pyramid to the structure we can generate the Terrain Tiles in isolation and then simply add links to these tiles from the appropriate GeoTiles. In addition, because each dataset is stored independently and only referenced via the GeoTiles, it is possible to selectively display any desired combination of datasets and to do so without needlessly loading data that will not be displayed.

4.2. The GlobalState Node

The integration of terrain features such as roads and buildings into a tiled pyramid structure raises a difficult issue. That is, it is possible for features to extend beyond tile boundaries, e.g. a road could cover multiple tiles, or a large building might sit on the boundary between two tiles. One way to deal with this problem would be to dissect the geometry for all ground features along tile boundaries at every resolution. This would constrain all features in a tile to be contained entirely within that tile. Another solution would be to simply include a link to the same feature in all of the GeoTiles that it projects into. We have adopted the latter approach as it does not constrain the cultural features to the same resolution range as the terrain, and it requires no modification to feature geometries. However, one problem is that we would load, store, and render duplicate

copies of each feature for every loaded GeoTile in which it occurs. We therefore treat features specially within the GeoTile node by keeping a count of the number of times a particular feature URL has been loaded. Whenever a file is to be loaded we increment its reference count and perform the load operation only on the first invocation. Similarly, we unload a feature file only when its reference count returns to zero.

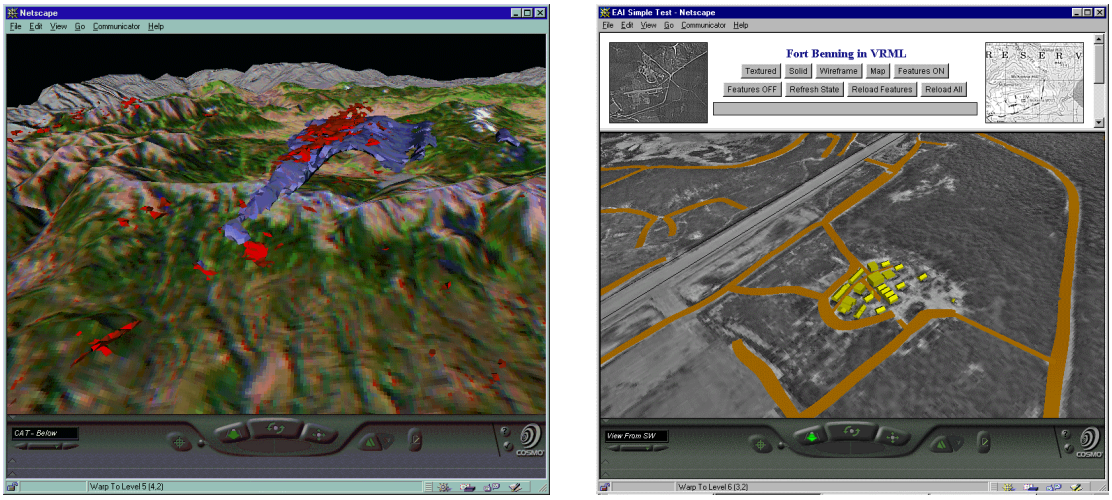

Figure 2: Two examples showing the embedding of georeferenced objects. The image on the left shows a weather simulation over the Colorado Rockies, while the image on the right shows an airstrip with georeferenced 3-D geometry for roads (orange) and buildings (yellow). These and other terrain datasets can be browsed from http://www.ai.sri.com/VRMLSets.

To implement this reference count system, we must be able to maintain state that is global to the entire scene. This can be done by simply declaring static class variables inside a Java script. However, a further complication arises because features may extend over multiple tiles. The tile that originally loaded a feature may get unloaded because it is no longer visible, and hence the feature geometry would get unloaded although it, itself, is still visible. We therefore developed a GlobalState node that is used to manage the loading and unloading of all features. This node stores all of the feature geometry and also provides an interface to select which features and terrain datasets are to be displayed. The latter facility enables the use of an External Authoring Interface (EAI) program to display a list of available terrain and feature sets and allow the user to select which data to view at any time.

5. GEOGRAPHIC COORDINATE SYSTEMS

VRML97 uses a right-handed, Cartesian coordinate system to model all objects in 3-D space[7]. In addition, the positive Y axis is defined as the up direction. In terms of georeferencing, this coordinate system is most similar to a geocentric coordinate system, where all locations are specified in units of meters as an (x,y,z) offset from the center of the planet. However, most georeferenced data are provided in some geodetic or projective coordinate system.

A geodetic (or geographic) coordinate system is related to the ellipsoid used to model the earth, e.g. the latitude-longitude system. A projective coordinate system 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[9]. Each of these coordinate systems was designed for slightly different applications and offers particular advantages and restrictions. For example, some projections can represent only small-scale regions, others are conformal (same scale in every direction), and others can be equal area (projected area corresponds to the earth's physical area over the entire projection). Figure 3 illustrates some contemporary coordinate systems.

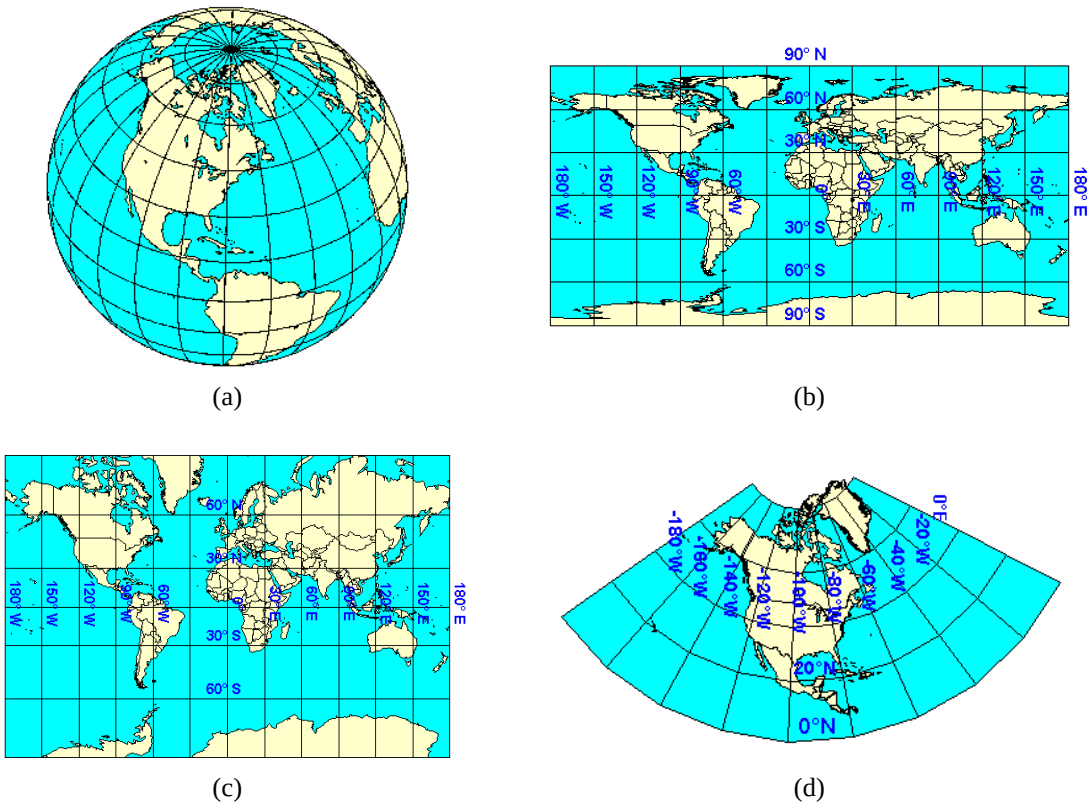

Figure 3: Examples of geodetic and projective coordinate systems. (a) Orthographic projection, used for perspective views of the earth, moon, and other planets, (b) latitude/longitude graticule, used to locate points on the earth's surface via a grid of meridians and parallels, (c) Mercator projection, used for navigation or maps of equatorial regions, (d) Lambert Conformal Conic, used by USGS for topographic maps. (Images adapted from Dana[10]. Reproduced with permission.)

5.1. The geotransform Java Package

We would like to directly specify coordinates in a VRML file using geographic coordinate systems. To do this, we require software to perform fast transformations from each of these coordinate systems into geocentric coordinates that we can pass to the VRML browser. We have therefore produced a Java package to perform this conversion. This package, called geotransform (http://www.ai.sri.com/geotransform), is based upon freely-available C source code developed by the SEDRIS project (http://www.sedris.org/). Currently, this package provides support for the lat/long, UTM, and geocentric coordinate systems.

5.2. The GeoCoordinate and GeoElevationGrid nodes

Using the geotransform Java package, we have developed two new nodes that enable a user to specify locations in geographic coordinate systems. The GeoCoordinate node functions as a replacement for the Coordinate node and can be used wherever a standard Coordinate node can be used, e.g. in an IndexedFaceSet, IndexedLineSet, or PointSet. The GeoCoordinate node offers a point field to specify a list of coordinates; however, the node also contains a geoSystem field to specify the coordinate system of the values in the vertex list. For example, specifying a geoSystem of "GDC" (geodetic) means that the vertex list consists of (latitude, longitude, elevation) tuples, where latitude and longitude are specified in units of degrees and elevation is measured in units of meters above the ellipsoid. These coordinates are transparently transformed to geocentric values by the GeoCoordinate node and then passed to the VRML browser to be rendered.

The limitations of the VRML97 ElevationGrid node for serious geographic applications are well known[3]. The primary problem is that this node assumes that all heights are relative to a flat plane, but planets are curved objects, not flat. ElevationGrids are therefore useful only for modeling local regions where the earth's curvature is not significant (in the order of roughly 100 km^2). We have therefore built a new node, called GeoElevationGrid, also using the geotransform Java

package. This node is used in exactly the same manner as an ElevationGrid, except that the height field is relative to the surface defined by the coordinate system in the geoSystem field. The xspacing and zspacing fields are delta values in that coordinate system. Thus, if a non-Cartesian coordinate system is specified, such as UTM, then the zero-height surface will have non-zero curvature. Figure 4 provides an example of this functionality.

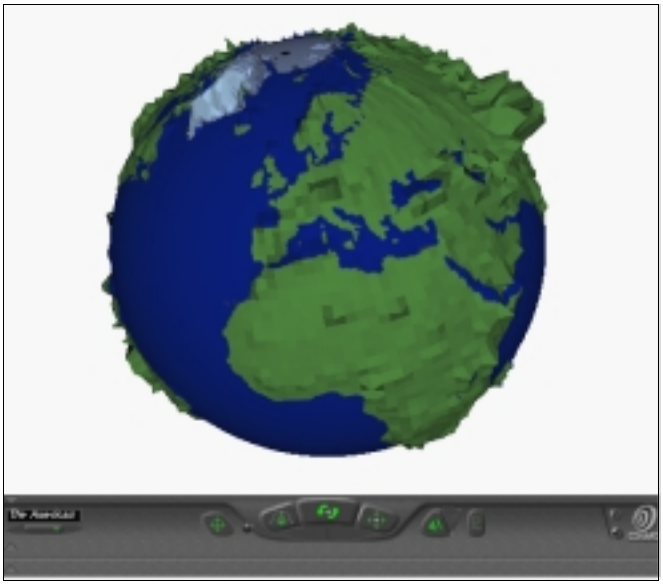

Figure 4: Illustrating the use of the GeoElevationGrid node to insert latitude/longitude coordinates directly in a VRML file. These coordinates are transparently transformed to geocentric coordinates to produce an ellipsoidal model of the earth. We have exaggerated all heights by a factor of 200 to highlight highly mountainous regions such as Greenland and the Himalaya.

5.3. Facing the Single-Precision Problem

There is a further problem to face when dealing with geographic coordinate systems: precision. VRML97 provides support for single-precision floating point numbers, designated as SFFloat and MFFloat[7]. Single-precision values are represented using 32-bits with a 23-bit mantissa[8], thus providing around 6 digits of precision ($2^{23} = 8.39 \times 10^6$). Given that the equatorial diameter of the earth is 12,756,274 m (under the WGS 84 ellipsoid[9]), we will be able to represent geocentric values only down to the order of 10's of meters. Below this threshold, we will experience various floating point rounding artifacts such as vertices coalescing and camera jitter. We observe, however, that single-precision is sufficient to render a single view. It is therefore clear that we must attempt to simulate higher precision by composing a number of single-precision local coordinate frames and transforming these into a local Cartesian coordinate system for VRML to display.

In order to perform this task, we introduce the GeoOrigin node. This defines an absolute georeferenced location and an implicit local coordinate frame against which geometry is referenced. This is done by specifying a double-precision, absolute location in the geoCoords field. We store this coordinate as a string so that it can be parsed by a Java script and stored internal to that script as a double-precision value. The coordinate system used to specify the coordinate in the geoCoords field is defined by a geoSystem field (e.g. "GDC" or "UTM").

The GeoOrigin node can then be instanced in an additional geoOrigin field of the GeoCoordinate and GeoElevationGrid nodes. Given that these latter two nodes contain geometry for a single coordinate frame, we can then translate their absolute geocentric coordinates into a local Cartesian coordinate system where the VRML world's origin corresponds to the geoOrigin location. For example, if we specify the geoOrigin of a GeoCoordinate node to be "Z13, 310385.0 4361550.0" in UTM coordinates, then this will be transformed internally to a double-precision geocentric coordinate of (-1459877.12, -4715646.92, 4025213.19). If we then supply an absolute UTM coordinate in the GeoCoordinate point array of "Z13, 310400.0 4361600.0", then this will be transformed internally to a geocentric coordinate of (-1459854.51, -4715620.48, 4025252.11). Finally, we transform this absolute geocentric coordinate to a single-precision local Cartesian coordinate system by subtracting the geoOrigin location to give (22.61, 26.44, 38.92).

6. NAVIGATION ISSUES

The topic of efficient navigation of our terrain structures is crucial. Without a means to interact with the data in a timely and appropriate manner, our work will be of limited practical value. There are a number of dimensions to the issue of navigating large, planetary models. We will discuss some of the most important issues here.

6.1. Navigating Deep LOD Hierarchies

Normally, LOD is used to switch between two or three representations of a model[11]. In our case, we will require at least around 17 levels of detail to produce a multi-resolution model of the earth down to 1 m accuracy. One problem is that the VRML97 specification does not specify when files should be loaded or unloaded, e.g. when an Inline node should load the file specified by its url field[7]. This behavior can therefore vary between different browsers. Currently, we find that most VRML browsers load all Inline scenes at once. This is a sensible approach for small scenes with a few Inline nodes; however, if we have a dataset that is 1 TB in size, the browser will attempt to load all these data into memory. In order to circumvent the obvious problems associated with this behavior, we have developed two different approaches for navigating deep LOD hierarchies:

Anchor/LOD tree files: We observe that loading around three or four terrain levels of detail gives an acceptable tradeoff between download time and interactivity. We therefore produce Tree files that each contain a small LOD hierarchy, with each tile being an Anchor node to a higher-resolution tree file. This approach uses standard VRML97 nodes to enable navigation of deep LOD hierarchies; but it has the disadvantage of requiring the user to click over areas to receive higher resolution, and then only the small region of interest is visible.

QuadLOD tree files: A more sophisticated approach is to employ a Java script to manage the loading and unloading of data. We therefore produced the QuadLOD node. This node implements a specialized quad-tree LOD facility where the four higher-resolution children of a tile are loaded only when the user enters a specific proximity volume around the tile. These children are also unloaded when the user leaves this volume. In addition, we cull nonvisible tiles from the scene graph and implement a simple Least Recently Used (LRU) tile caching mechanism to reduce network downloads. We can therefore produce tree files that each contain a single QuadLOD reference. Using this solution, users can navigate arbitrarily deep into a multi-resolution dataset with efficient file and memory management being performed on the fly.

6.2. Terrain Following

We have already noted that the earth is round, not flat. As we navigate over the earth's surface we should therefore expect to follow a curved flight path. However, the default navigation methods such as WALK and FLY propel the user only along a linear flight path parallel to the X-Z plane. We therefore desire a navigation method that will maintain a particular height above the surface of the earth. To do this we need to know the up vector for a particular region of terrain. VRML implicitly assumes that the Y-axis is up, but when dealing with a curved surface the actual vector varies continuously, for example, it could be modeled by the 3-D normal to the plane that is tangent to the reference ellipsoid at the region in question.

6.3. Altitude-based Velocity

The velocity at which the user can navigate should be dependent upon that user's height above terrain. For example, when flying through a valley at a height of 100 m above the terrain, a velocity of 100 m/s could be considered relatively fast. However, when viewing the entire globe from space at an altitude of 20,000 km, zooming in at the same speed would be painfully slow. We should therefore scale the velocity of the user's navigation to achieve a constant pixel flow across the screen. A simple linear relationship between velocity and distance from the planet ellipsoid proves adequate for this purpose.

6.4. Active Maps

When flying low over an area of terrain, it is often difficult to maintain a context of position in the world. We therefore employ a map display that is managed by a Java applet. Through the EAI, it is possible to obtain the location of the user in the geographic environment, e.g. from the position_changed eventOut of a ProximitySensor placed around the entire scene. We can then project this 3-D geocentric coordinate onto the map display so that the user can easily ascertain the location in the world. In addition, we can allow the user to click over the map and then move the viewpoint directly to that location. This can be done by updating and binding a Viewpoint node in the VRML scene graph.

7. METADATA

Metadata are simply data about data. In our context, this means information that describes a particular terrain dataset or element of that dataset, such as a name, location, resolution, size, source attribution, data accuracy, format, currentness, or copyright notices. The issue of how best to represent metadata is one that has received much attention recently. The Federal Geographic Data Committee (FGDC) has produced an extensive standard for storing metadata relating to digital geospatial data[12]. A lot of time and effort has been invested in this SGML-based standard, referred to as the Content Standard for Digital Geospatial Metadata (CSDGM). We would like to provide support for this work, but we also do not want to require this as the only representation, because CSDGM can be complex to produce. Indeed, the FGDC sees the description of terrain data using CSDGM as a function performed largely by some data librarian body.

We therefore propose the use of a new node, GeoInformation, which provides the ability to specify a url field to locate a CSDGM file, if available. In addition, this node enables an arbitrary list of keyword/value pairs to specify a minimum searchable set of metadata inside the VRML file. We specify some standard keywords that should be used, but allow developers to include additional keyword/values to support their own applications. We have based our keyword names on the fields suggested for the Denver Core, a set of metadata elements identified by the Metadata Summit in February 1996:

1. `Theme_Keywords`
2. `Place_Keywords`
3. `Bounding_Coordinates`
4. `Abstract`
5. `Purpose`
6. `Time_Period_of_Content`
7. `Currentness_Reference`
8. `Geospatial_Data_Presentation_Form`
9. `Originator`
10. `Title`
11. `Language`
12. `Resource_Description`

8. CONCLUSIONS

We have presented solutions to facilitate the visualization of massive, data-rich, georeferenced objects. This was done with the introduction of various new nodes for VRML97: QuadLOD (to manage the progressive loading/unloading of tiled data), GeoTile and GlobalState (to fuse and allow selection between georeferenced features), GeoCoordinate and GeoElevationGrid (to enable specification of coordinates in geographic coordinate systems), GeoOrigin (to define a coordinate frame for offsetting absolute coordinates to single-precision), and GeoInformation (to encapsulate metadata for a dataset). These nodes provide a number of capabilities that were not previously available in a web-based graphics file format. Specifically, these nodes:

1. Represent spatial data using geographic coordinate systems such as lat/long or UTM.
2. Preserve the original geographic data and use these data to produce a visualization.
3. Integrate geographic data from different sources and in different coordinate systems.
4. Deal with single-precision floating point inaccuracies.

The nodes described here are available from http://www.ai.sri.com/~reddy/geovrml/protos/index.shtml. In addition, we have released the source code to the library and utilities that we developed for generating tiled pyramids of source data, and for producing the VRML terrain representation that we have described here (http://www.ai.sri.com/tsmApi/). Sample terrain datasets are also provided to illustrate these concepts (http://www.ai.sri.com/VRMLSets/). It is hoped that by providing these materials we will help to foster further innovation in this area.

ACKNOWLEDGEMENTS

This work has been funded in part under the following DARPA programs: MAGIC-II (Multidimensional Applications Gigabit Internet Consortium), sub-contract 12165SRI under contract no. F19628-95-C-0215, and BADD (Battle Assessment and Data Dissemination), contract no. MDA972-97C-0037. DARPA has approved this paper for public release, distribution

unlimited. Terrain imagery and elevation data supplied by the U.S. Geological Survey, EROS (Earth Resources Observation System) Data Center.

REFERENCES

1. Gore, A. (1998). “The Digital Earth: Understanding Our Planet in the 21st Century”. Speech delivered at the California Science Center (CSC), Los Angeles, CA. 31 January 1998.
2. Moore, K. (1997). “Interactive Virtual Environments for Fieldwork”. *British Cartographic Society Annual Symposium* 1997, University of Leicester. 12-14 September.
3. Abernathy, M. and Shaw, S. (1998). “Integrating Geographic Information in VRML Models”. *Proceedings of the Third Symposium on the Virtual Reality Modeling Language*, Monterey, CA. February 16-19, pp. 107-114.
4. Falby, J.S., Zyda, M.J. Pratt, D.R. and Mackey, R.L. (1993). “NPSNET: Hierarchical Data Structures for Real-Time Three-Dimensional Visual Simulation”. Computer and Graphics 17(1):65-69. Pergamon Press, UK.
5. Hitchner, L. E. and McGreevy, M. W. (1993). “Methods for User-Based Reduction of Model Complexity for Virtual Planetary Exploration”. *Proceedings of the SPIE: the International Society for Optical Engineering*, vol. 1913, pp. 622-36.
6. Leclerc, Y. G. and Lau, S. Q. (1995). “TerraVision: A Terrain Visualization System”. Technical Note 540, AI Center, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025. Available electronically from: http://www.ai.sri.com/pubs/technotes/aic-tn-1994:540/document.ps.gz
7. ISO/IEC 14772-1:1997 (1997). “The Virtual Reality Modeling Language”. December 1997. http://www.vrml.org/Specifications.
8. IEEE 754-1985 (1985). “IEEE Standard for Binary Floating-Point Arithmetic”. IEEE Standards Publication.
9. Snyder, J. P. (1987). *Map Projections – A Working Manual*. U.S. Geological Survey Professional Paper 1395, U.S. Government Printing Office, Washington, DC.
10. Dana, P. (1998), “Geographer’s Craft Project Web Page”, Department of Geography, University of Texas at Austin. http://www.utexas.edu/depts/grg/gcraft/notes/mapproj/mapproj.html.
11. Marshall, D., Story, D., and Maxwell, D. (1997). “Authoring Compelling and Efficient VRML 2.0 Worlds”. SIGGRAPH ’97 Course #28, August 1997.
12. FGDC-STD-001-1998. (1998). “Content Standard for Digital Geospatial Metadata”. Version 2.0, FGDC, USGS, 590 National Center, Reston, CA 20192. http://fgdc.er.usgs.gov/standards/.