DHPC-043

Interactive Viewing of 3D Terrain Models using VRML

Alvin Leung
School of Computer and Information Science
Syracuse University, Syracuse NY 13244, U.S.A.
alvin@npac.syr.edu

Paul Coddington
Northeast Parallel Architectures Center
Syracuse University, Syracuse NY 13244, U.S.A.
and
Department of Computer Science*
University of Adelaide, Adelaide SA 5005, Australia
paulc@cs.adelaide.edu.au

6 April 1998

Abstract

Many applications, including 3D Geographic Information Systems, weather simulation and flight simulation, rely on 3D terrain models. A system that supports these applications should ideally provide real-time interactive rendering of section of a large set of terrain data that may be accessed remotely. We present a Web-based system for interactive 3D terrain rendering using Virtual Reality Modeling Language (VRML), a Web standard for presenting 3D virtual worlds. We describe an implementation of a 3D terrain model in VRML 1.0, and some ideas for improving the implementation using the new features available in VRML 2.0. We discuss the advantages and disadvantages of VRML for constructing, accessing and visualizing 3D terrain models.

Keywords: VRML, 3D graphics, terrain rendering.

*Current address.

1 Introduction

Many applications, such as 3D Geographic Information Systems (GIS), weather simulation and flight simulation, use terrain models in their presentations. Our aim is to provide such applications with real-time interactive rendering of terrain data that is accessed remotely using a client/server system. Of particular interest are datasets for very large areas that may require gigabytes of storage, but the user may only want to view a relatively small area of the full terrain model.

To provide real-time interactive visualization of a remotely-stored 3D terrain model, a dedicated rendering server and a custom-built client program could be used, as with the parallel rendering program developed by NASA JPL [6], which was used for visualizing 3D terrain models of Mars and Venus. In this case the rendering is done by a server program, and the results are sent over the network for display in the user interface on the client. However this can only simultaneously serve a small number of independent clients, because of the substantial computational resources required on the server. It is therefore not feasible for many applications, such as general educational use.

Alternatively, the data could be created in a proprietary 3D graphics format for either a general 3D visualization program, such as SGI's OpenInventor [10], or a 3D GIS program. The data could be downloaded from a standard FTP or Web server, and then loaded into the visualization program. In this case, the rendering is done on the client, which is much more scalable than server-side rendering.

We have adopted the latter approach, but rather than using a proprietary 3D visualization program and data format, we have investigated the use of Virtual Reality Modeling Language (VRML) [8], a standard for using 3D graphics on the Web.

2 Terrain Models

A 3D terrain model consists of two basic components: a Digital Elevation Model (DEM), which describes the topography of the terrain; and some geographical information for the same area, such as satellite images, aerial photographs, or geospatial data such as land use or vegetation cover. The DEM is a 2D array of values of the height at a specific grid position. By connecting the points in the DEM, a polygon-based 3D representation of the terrain can be generated. The spatial data can be represented as a 2D image that can be draped over the polygon-based DEM to create a 3D terrain model consisting of texture-mapped polygons, where "texture" refers to a set of pixels that are used to color the polygon. Standard computer graphics algorithms for rendering texture-mapped polygons can then be used to display the terrain model on a computer.

An alternative to texture-mapping is to associate a color with each vertex of the polygon, and then use an interpolation algorithm, such as Gouraud shading [4], to color the polygons. However, texture mapping allows us to reduce the number of polygons to be rendered, while maintaining the resolution of the surface image. In order to support real-time rendering of the terrain, the number of polygons should be minimized. The terrain model should also be available in multiple resolutions, since this allows the model to be used on computers with widely varying memory, compute and graphics capabilities, but more importantly, it allows the rendering engine to use a lower resolution model for terrain that is distant from the viewer, further reducing the number of polygons to be rendered. Also, for terrain models that are accessed over a wide-area network, the client can choose the resolution based on the bandwidth available. The availability of multiple resolution data sets is particularly important for use over the Web, where the network bandwidth and the computational and rendering capabilities of clients can vary by orders of magnitude.

We have developed a novel wavelet-based mesh optimization algorithm to minimize the number of polygons used to model the terrain [5]. The algorithm produces an irregular grid of points based on a specified error tolerance for the height values, and then creates a set of polygons using Delauney triangulation. The resulting irregular mesh DEM is sparser where the terrain is flatter or smoother,

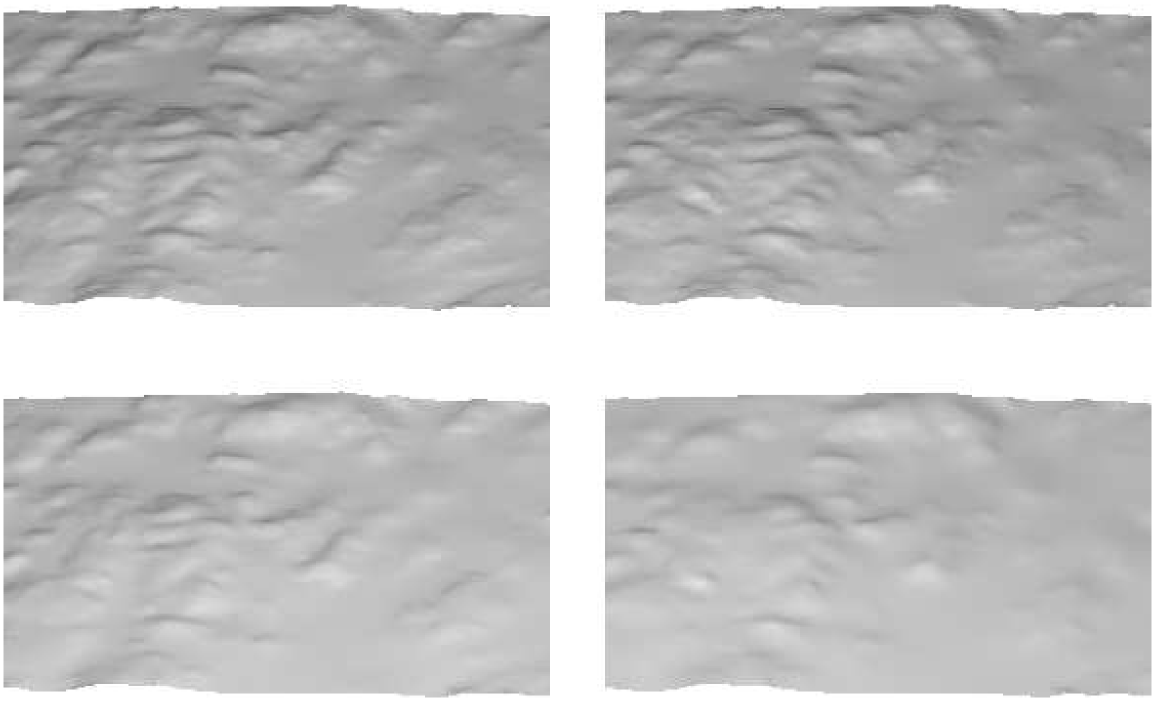

Figure 1: A collection of rendered VRML terrain model colored white. The error tolerances are A (upper left), 2A (upper right), 4A (lower left) and 8A (lower right), where A = 5 meters.

and more dense where the elevation changes more rapidly.

The data set used in our prototype implementation is a terrain model consisting of a 30m resolution Landsat satellite image and a 100m resolution DEM of an 89x89 km area around Albany, New York. The image and the DEM have been registered to the same coordinate system. The image data is approximately 3000x3000 pixels.

Because terrain models can cover very large areas, they are usually broken into smaller "bite-sized" pieces by creating a tiling of the area, with each tile being small enough to be handled by the computer doing the visualization or processing. Our sample data set has been broken up into a 7x7 tiling, with each tile consisting of 128x128 points in the DEM, which corresponds to a polygonal representation of approximately 32,000 triangles. Applying our wavelet mesh optimization algorithm with a small error tolerance A reduces the number of polygons to 67% of this, with no distinguishable change to the terrain model. Increasing the error tolerance to 2A, 4A and 8A gives terrain models with only 50, 27 and 12 percent respectively of the number of vertices and polygons. The rendered images still show the dominant features of the terrain, even at the lowest resolution, as shown in figure 1.

3 VRML

VRML is a Web standard for encoding a 3D virtual world, so a VRML world can be accessed remotely using standard Web servers and browsers, and visualized interactively on the client using widely-available VRML viewers that provide state-of-the-art rendering of the world. The functionalities offered by the VRML 1.0 standard [2, 8] include:

basic shapes `AsciiText`, `Cone`, `Cube`, `Cylinder`, `IndexedFaceSet`, `IndexedLineSet`, `PointSet` and `Sphere` are the basic building blocks of a VRML world. A polygonal object is encoded by using the `IndexedFaceSet` node.

geometry `LOD`, `Coordinate3`, `ShapeHints`, `Normal` and `NormalBinding` serve as the "adjectives" of the objects. `LOD` allows different levels of detail to be used in the description of an object. `Coordinate3` contains the set of vertices of a polygonal object.

material `Material`, `MaterialBinding`, `Texture2`, `TextureCoordinate2` and `Text2Transform` decorate the virtual world just like paint in the real world. There are two ways to paint the virtual world. First, a color can be assigned to each vertex or polygon. Second, a texture (an arbitrary set of pixels) can be pasted onto a polygon.

transformation `MatrixTransform`, `Rotation`, `Scale`, `Transform` and `Translation` provide a way to position, scale and align an object with respect to the user's viewpoint and/or other objects.

web accessing functions `WWWAnchor` and `WWWInline` enable the VRML world to access additional Web resources. `WWWAnchor` is used to embed hyperlinks in the world to other worlds or Web pages. `WWWInline` provides an avenue to incorporate other VRML worlds into the current world.

Although VRML 1.0 is a convenient way to present 3D information over the Web, it has several shortcomings. The main drawback of the VRML 1.0 specification, which originated from the Silicon Graphics's OpenInventor [10] file format, is that it only allows a statically defined world. This characteristic is inherent from its roots as a file format. However, this philosophy is not compatible with the trend of the Web, as shown by the success of Java and Javascript, which allow a more flexible and dynamic interface to the Web. Furthermore, the static definition of the world implies that the client has to store and process information proportional to the size of the entire world. This forces the author to put an upper bound on the size of the VRML world that a viewer can traverse, based on the limited memory and CPU power of the client machine. This is a major problem for representing large-scale terrain models in VRML.

Another problem is that VRML objects waste a lot of bandwidth. The VRML 1.0 specification focusses on the specification of 3D graphics objects, with little regard to the issues of minimizing download time. For example, the VRML files must be in ASCII format. Compression of the ASCII files using `gzip` is supported, but a compressed binary format would require far fewer bytes. This is an important issue for accessing potentially very large terrain data models over the Web.

4 Terrain Visualization using VRML

Because a VRML browser is designed to be a generic VRML renderer, it is not optimized for terrain rendering, so hints must be provided to the browser on when and where it can conserve rendering effort. A carefully structured VRML file can enhance the rendering performance of the browser for displaying a terrain model. This can be achieved by providing multiple levels of detail for the terrain model, and instructions on when to display what level of detail. By using our mesh optimization algorithm, a multi-resolution terrain model can easily be generated. As a result, the browser can provide high quality images with better frame rates by minimizing the number of polygons to be rendered in the terrain model, based on the viewing parameters.

As mentioned in section 2, the terrain model must first be split into manageable tiles, which are independent VRML worlds. In VRML 1.0, there is no way to transfer between these worlds. In our prototype implementation, each world is a `128x128` grid, covering an area that is `12.8 km` on a side.

There are two ways of serving the VRML data. The usual way is to store it as VRML files. Alternatively, a Common Gateway Interface (CGI) program running on the Web server can be used to extract a section of terrain data stored in any format, and construct the VRML file for the terrain model on the fly. Since we allow the VRML file to be created based on a number of different parameters that can be specified by the user, such as error tolerance and tile size, we found the latter approach to be more convenient.

The structure of the VRML file for the terrain model is shown in figure 2. The `LOD` nodes are used to further subdivide the terrain model, by providing an additional tiling within each VRML world. The purpose of this tiling is to allow multiple data sets of varying resolution to be provided for each LOD node. This allows the VRML browser to render each LOD node at an appropriate resolution, so that the nodes further from the user are rendered at a lower resolution, thus reducing the rendering effort. A major additional advantage of using LOD nodes is that the browser can initially download only the lower resolution data for distant LOD nodes, thereby substantially reducing the startup time. As the user moves over the terrain, the higher resolution data can be downloaded as needed (although this depends on the implementation of the browser).

In figure 2, each `LOD` node contains four children for four different resolutions of the terrain data set. The first set is created using the user-specified error tolerance, `A`. The second, third and fourth data sets are computed with error tolerance levels of `2A`, `4A` and `8A`, respectively.

The data for each of the four resolution levels within each LOD node is specified using `WWWInline` nodes, each of which has the URL for a self-contained VRML file for a given region and resolution. Since we are using a CGI program to generate the VRML file on the fly, the URL points not to a VRML file, but to a CGI program to retrieve the corresponding VRML terrain data, as well as information for the size and location of the bounding box (`bbox` in figure 2), which is used by the browser to decide if a `WWWInline` node is visible or not.

When one of the `WWWInline` nodes is activated by the VRML browser, a CGI program is spawned by the Web server to dynamically generate the specified VRML terrain data [5]. The terrain data is stored as a table of wavelet coefficients in a database. The CGI program accesses this table via SQL commands, and carries out the processes shown in figure 3. A subset of the wavelet coefficients is selected and passed onto the reconstruction algorithm to create a sparse DEM. This is passed to a triangulation algorithm. The resulting polygonal information is encoded using VRML. Finally, the VRML file is shipped to the client for presentation. The structure of our VRML terrain data is shown in figure 4.

In addition to assembling the polygonal terrain data, the CGI program is also responsible for generating the texture map for all the polygons and the color for each vertex. The texture map of a VRML object can be stored either as in-line ASCII format or in an external image file format (JPEG [7] is suggested and PNG [3] is strongly recommended by the VRML specification as part of the implementation of the browser).

Since the texture map is a large proportion of the VRML file size, the resolution of the texture map is varied according to the accuracy of the terrain model. We use the highest resolution texture map for error tolerance levels `A` and `2A`. For the `4A` and `8A` resolution terrain data, we use half and quarter resolution texture map images, respectively, so that the download time is reduced for these low resolution models. In addition to the reduction of the network transfer time, the lower resolution texture maps also reduce the amount of memory required on the client, which can otherwise be a limiting factor in displaying a large terrain model.

```
#VRML V1.0 ascii

Separator {
   LOD {
      range [ 9000, 18000, 36000 ]
      center 3153.153150 579.000000 3153.153150
      WWWInline {
         name "http://machine/cgi-bin/terrain_block_cgi?TableName=Albany
&DEM_width=128&DEM_height=128&Tolerance=5&BlockSize=64&X=0&Y=0&LODLayer=0"
         bboxCenter 3153.153150 579.000000 3153.153150
         bboxSize 6306.306300 122.000000 6306.306300
      }
      WWWInline {
         ...
      }
      WWWInline {
         ...
      }
      WWWInline {
         ...
      }
   }
   LOD {
      range [ 9000, 18000, 36000 ]
      center 9509.509500 398.000000 3153.153150
      WWWInline {
         ...
      }
      ...
   }
   ...
}
```

Figure 2: The structure for VRML terrain data with `LOD` and `WWWInline` nodes. Each `LOD` node contains four `WWWInline` nodes as children. The `WWWInline` nodes incorporate the terrain data for a particular region and resolution by activating a CGI program with a specific set of parameters. The `range` parameter specifies at what range (from the `center` of the `LOD` node) the various `WWWInline` ¿ nodes are activated.

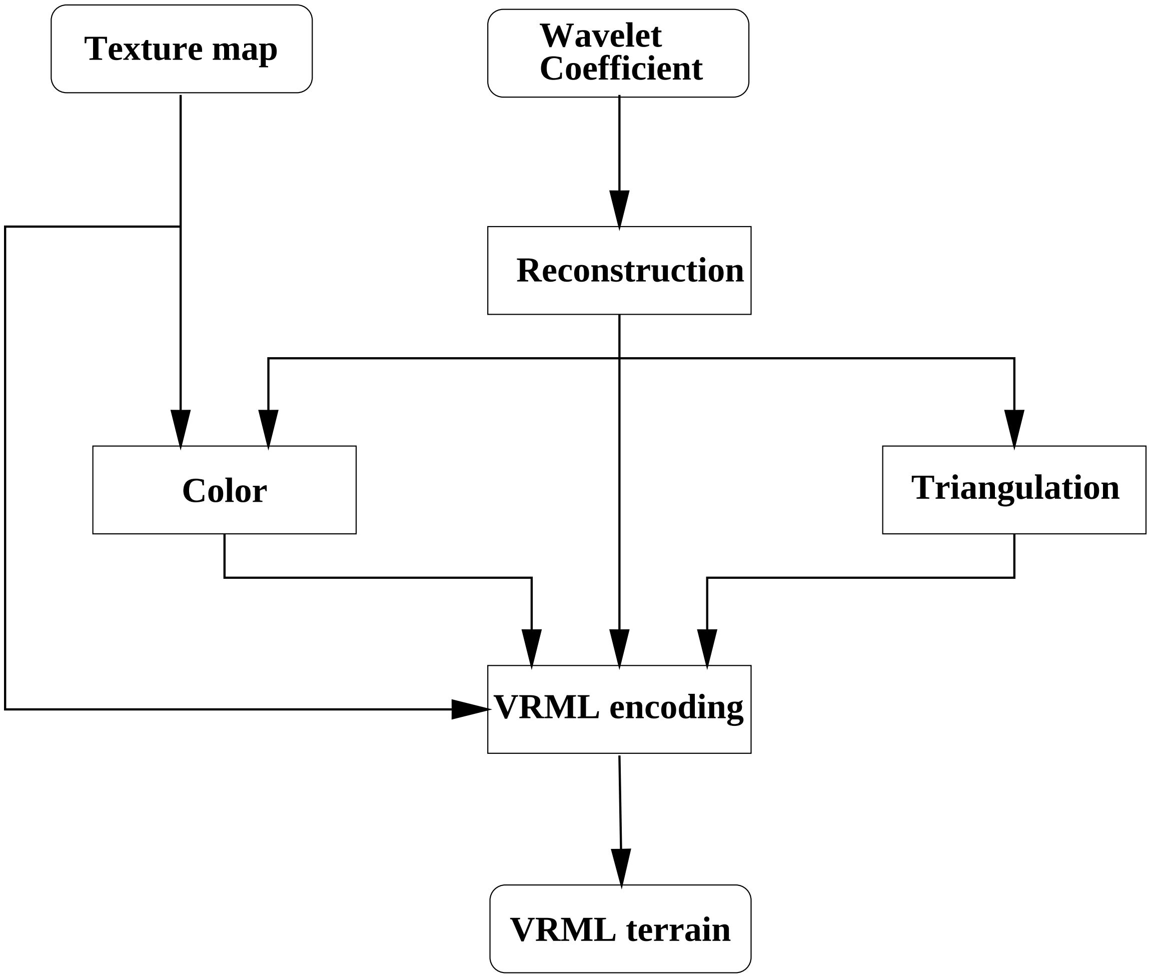

Figure 3: The process of obtaining a VRML terrain model from a set of wavelet coefficients and a texture map is shown. The round corner boxes indicate data. The square boxes are processes.

```
#VRML V1.0 ascii

Separator {
   ShapeHints {
      vertexOrdering COUNTERCLOCKWISE
      faceType CONVEX
      creaseAngle 0.5
   }
   Material {
      diffuseColor [ ... ]
      ambientColor 0.1 0.1 0.1
   }
   Texture2 {
      filename " ... "
# or
      image ...
   }
   TextureCoordinate2 {
      point [ ... ]
   }
   Coordinate3 {
      point [ ... ]
   }
   IndexedFaceSet {
      coordIndex [ ... ]
   }
}
```

Figure 4: The structure for the VRML file describing the terrain data. `ShapeHints` provides basic rendering directions, such as the order of the vertices, the type of polygon and how to shade the polygon's edges. `Material` contains the color values for each vertex and the ambient light value for the whole terrain. `Texture2` specifies the texture for the terrain. The texture can be stored in an image file or as inline data. `TextureCoordinate2` is used to pin down the orientation of the texture map, when applied to the terrain model (this node is included to bypass a bug in Webspace in handling the texture). `Coordinate3` contains all the vertices' 3D coordinates. `IndexedFaceSet` provides the connectivity to form the polygons from the list of vertices.

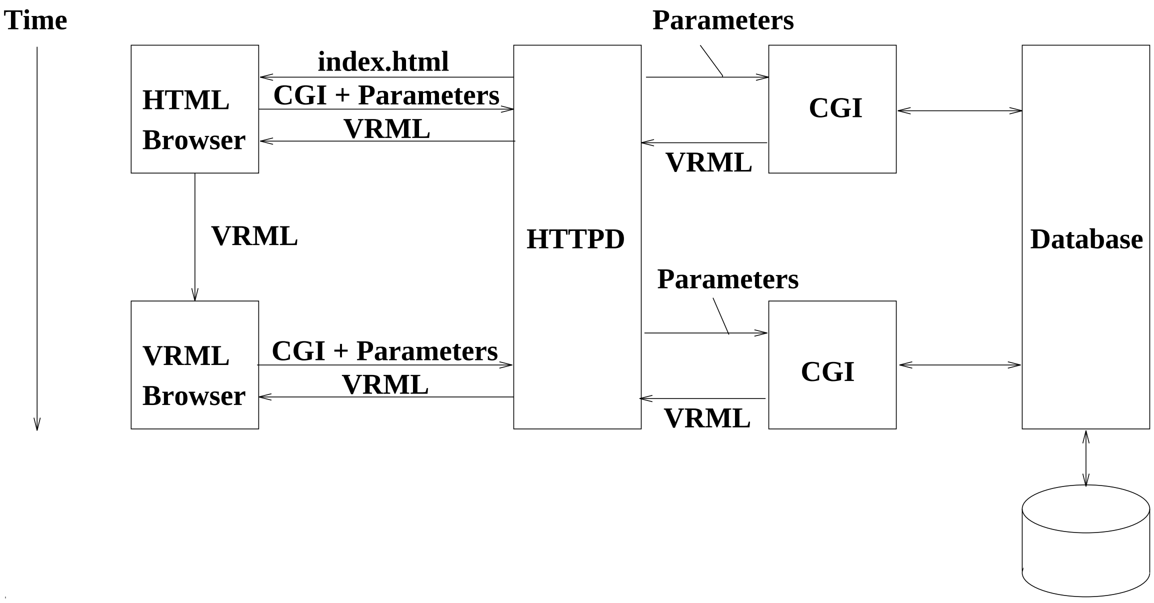

Figure 5: The operations of the Web-based terrain visualization system.

5 Implementation Issues and Results

In this section, we discuss implementation and performance issues of terrain models developed using VRML 1.0. Most of these tests were done using the Webspace 1.1 VRML browser [9].

Before the user can enter the VRML world, they will first go through a Web page interface, which transfers them from a 2D information oriented space to a 3D world. The Web page for the prototype system consists of a brief description on how to set the adjustable parameters for specifying the resolution of the terrain model (so that a lower resolution model can be requested for a client with limited bandwidth and rendering capabilities), and a clickable 2D map of the area for choosing the starting position. These parameters are handed off to the CGI program on the server, which generates the VRML file for the specified grid section of the terrain model. The client then spawns a VRML browser to render the VRML file. The operation of this system is shown in figure 5.

A major problem in developing VRML terrain models that require multi-resolution data sets, is that VRML browsers have problems matching the boundaries of `LOD` nodes with different resolutions, and strange artefacts often occur in the rendering. This is possibly due to the limited accuracy of the floating point representation of the polygon positions. As a result, we must take appropriate steps to ensure that areas with different resolutions join seamlessly. For this reason, the edge vertices of all our terrain models are extracted at the highest resolution, while the internal vertices are obtained using our mesh optimization technique [5].

This is a major drawback for constructing VRML terrain models, since it greatly restricts the number of LOD nodes, and the number of different resolution levels, that can effectively be used. If there was no edge restriction, the LOD nodes could have many resolution levels, all the way down to just 2 triangles to represent an entire node. That would allow much larger worlds to be constructed, since we could divide them into many LOD nodes, but downloading the data would still be fairly fast, since the distant LOD nodes would be at a very low resolution and therefore contain only a small amount of data. However having to keep the edges at a fixed resolution means that the size of the lower resolution data remains quite high, which restricts the side of the VRML world. Also, the number of polygons in the terrain model increases as the number of `LOD` nodes (tiles) is increased, since this creates more edges. For this reason, we found that the optimal number of `LOD` nodes for rendering our `128x128` example terrain segment was only 4. This was the main limiting problem we encountered in using VRML 1.0 for terrain rendering.

A VRML browser is intended to provide an interactive 3D environment to the user. If the browser is executed on a high-end graphics workstation, it can maintain a detailed and fully texture-mapped terrain model on screen at all times. However, in order for a browser to maintain its interactive response and a reasonable frame rate when rendering a complex scene on a client with lesser capabilities, it must render the image with less computationally demanding techniques. For example, to avoid CPU-intensive texture map manipulations, the browser may render the image by using Gouraud shading [4], or even a wire-frame representation, when the user is moving over the terrain. When the user stops moving, the fully texture-mapped model is rendered on screen. Unfortunately many VRML browsers do monochrome rather than color Gouraud shading (so the terrain will look like figure 1) unless colored vertices are provided in the VRML file in addition to the texture map (even though the browser should be able to interpolate the texture map). This adds about 25% to the size of the VRML file.

Another browser implementation problem is that although the VRML specification advice is to use the GNU compression tool `gzip` to reduce the data size, we have yet to find a browser that can accept compressed VRML data from a CGI program within a `WWWInline` node. This is a major drawback in using a CGI program to create the VRML data on the fly, since it leads to long delays in the presentation, induced by both the CGI activation (latency) and the transmission delays (bandwidth) due to the larger data size of the uncompressed VRML file.

If all the VRML terrain model data is available locally on the client, and the client has good graphics capabilities, the rendering performance and frame rate for interactively exploring a terrain model using a VRML browser is quite good. So the key question is how does this change when the data must be downloaded over a network which may have quite high latency and low bandwidth?

For the sample data set we have used, of a `128x128` grid, table 1 shows the transfer time for different network bandwidths of the texture maps for the VRML terrain model using different resolutions and compression schemes. Clearly JPEG compression is the only reasonable option for interactive performance. Table 2 shows times to download the VRML file of the terrain model with vertx colors but without the texture map data (which will be read from an external file in the case of a JPEG texture map). The total download time can be estimated by summing the times to download the VRML file and the texture file for a particular resolution.

Note that these times are best-case values, assuming peak network performance and no latency, although the latency can be of the order of a second for every VRML file and texture map for every LOD node. Even so, we can see that for bandwidth of less than 1 Mbits/sec, the overall download time for a section of the VRML terrain model is too slow for real-time rendering of a large area of terrain, which may require many tiles of this size. Consequently, terrain visualization using VRML 1.0 is of limited usefulness. In the next section, we will show how to exploit the new features offered by the VRML 2.0 standard to help overcome these problems.

6 Improvements using VRML 2.0

The VRML standard has advanced to version 2.0 [1], and some VRML browsers now support many of the VRML 2.0 features. The new VRML specification not only corrects many shortcomings of the old standard, such as the static definition of the 3D world, it also provides many new features, such as sensors and scripting. We highlight some features that can be used to present a better VRML terrain model:

Format	Resolution	Bandwidth			
		10Mbits/sec	1Mbits/sec	128kbits/sec	38.8kbits/sec
ASCII	full	1.7	17.7	142	468
	half	0.446	4.46	35.7	118
	quarter	0.113	1.13	9.00	29.7
gzip compressed ASCII	full	0.528	5.28	42.3	139
	half	0.133	1.33	10.6	35.0
	quarter	0.0333	0.333	2.66	8.78
PNG	full	0.421	4.21	33.7	111
	half	0.105	1.05	8.44	27.8
	quarter	0.0265	0.265	2.12	6.99
JPEG	full	0.0243	0.243	1.94	6.41
	half	0.00766	0.0766	0.613	2.02
	quarter	0.00242	0.0242	0.194	0.639

Table 1: The theoretical transfer time (in seconds) for the texture map for the 128 $\times$ 128 VRML terrain model using different texture formats and resolutions.

	Error Tolerance	Bandwidth			
		10Mbits/sec	1Mbits/sec	128kbits/sec	38.8kbits/sec
Uncompressed	5	0.624	6.24	50.0	165
	10	0.470	4.70	37.6	124
	20	0.280	2.80	22.4	73.8
	40	0.156	1.56	12.5	41.2
gzip Compressed	5	0.134	1.34	10.8	35.5
	10	0.105	1.05	8.44	27.8
	20	0.0649	0.649	5.19	17.1
	40	0.0375	0.375	3.00	9.90

Table 2: The theoretical transfer time (in seconds) for a 128 $\times$ 128 VRML terrain model (excluding the texture map) with different tolerance levels.

ElevationGrid provides a convenient way to encode and present a uniformly sampled 2D surface using a matrix of height values. Since the values are uniformed distributed, the connectivity information is implicitly defined, so VRML 2.0 encoding of a 2D surface is much more compact than in VRML 1.0. However, this is not useful for encoding an optimized irregular mesh DEM.

NavigationInfo contains the description of the physical properties, such as terrain following, speed and visibility, of the viewer and viewing model.

ProximitySensor permits the initiation of actions in the VRML world based on the user position. Hence, we can prefetch the needed terrain data and hide the network latency and transfer time.

Script provides a mechanism for the VRML world to incorporate complex animations, which must be defined by a program. The current scripting languages suggested by the specification are Java and Javascript.

Viewpoint can be used to specify a predefined route for a tour. Using this node, we can limit the flight characteristics of the user's virtual craft, so that the user can experience a smoother presentation on a low-end platform.

VisibilitySensor identifies the visible part of the world. This enables us to order the prefetch process, so that the visible terrain data is fetched with higher priority. In addition, the visibility sensors also permit us to selectively turn on or off the proximity sensor to minimize the processing overhead of the VRML browser.

By making use of these VRML 2.0 features, particularly sensors to monitor the user's location and orientation, and Java scripts to more carefully control the downloading and management of sections of the terrain model (tiles) as needed, the VRML terrain presentation can be made more efficient. Using a Java script should also enable the dynamic downloading of other VRML worlds, thus removing the static restriction of VRML 1.0. So in this section, tiles can refer to either different LOD nodes of a VRML world, or different VRML worlds that make up a large terrain model.

With a Java controller script, it should also be possible to dynamically adapt the resolution (or error tolerance) used for the LOD nodes in order to suit the bandwidth and computation power available to the client, rather than having to specify them in advance.

To reduce the number of polygons in the VRML world, we can use only the tiles for the region that is visible to the user, and a few other tiles for buffering, as shown in figure 6. To hide the latency, download time and processing time required in accessing a new tile, we can attempt to predict the user's motion and use a buffer zone in the direction of travel. We can also emulate the effect of the earth's curvature by removing tiles that are beyond a certain distance, which can be adjusted depending on the capabilities of the client machine. We can also emulate the effect of the earth's curvature by removing tiles that are beyond a certain distance, which can be adjusted depending on the capabilities of the client machine.

In order to determine which grid sections are visible to the user, we must install a visibility sensor for each of the tiles. The visibility sensor covers an area slightly bigger than the associated tile, so that the tile can be added to the VRML scene graph before the terrain model data is needed for rendering the scene. The sensor grid can be coarser further from the viewer, as shown in figure 7.

A large number of visibility sensors would incur high processing overhead on the browser. Fortunately, the visibility sensors can be turned on or off. A hierarchical sensor net could be used to reduce the processing overhead. When a parent sensor is visible, it enables all its children sensors. When the parent sensor is no longer visible, all its children sensors are disabled, so that the browser need not check the children sensors against the view volume.

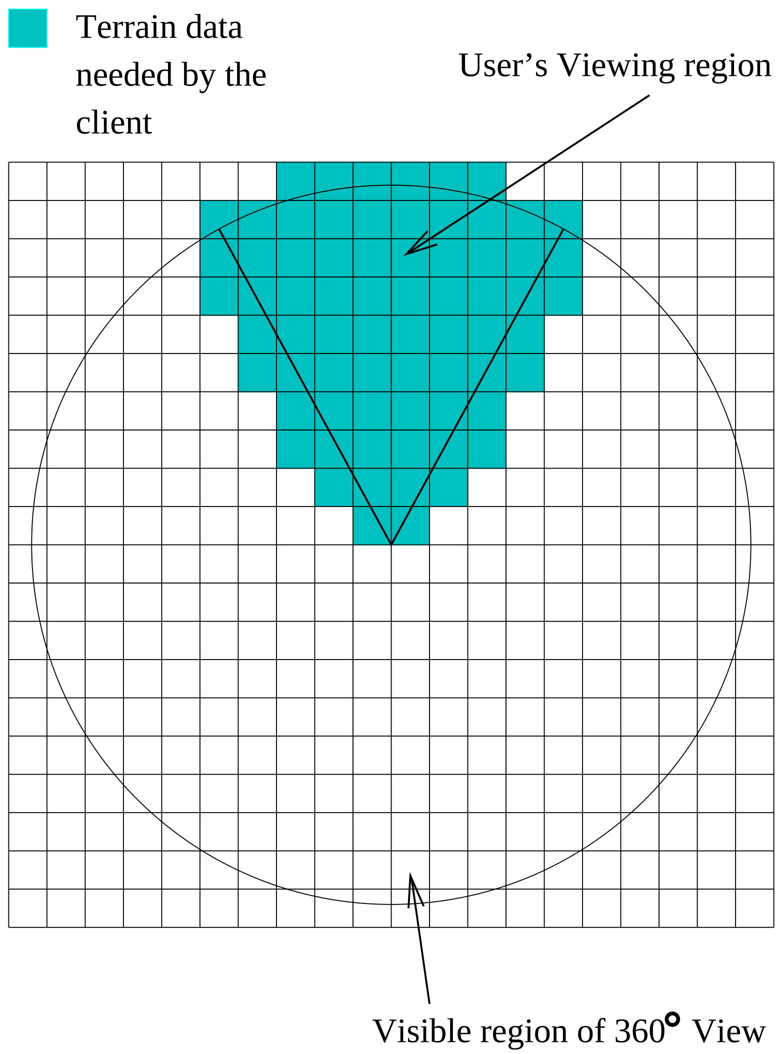

Figure 6: Only a small part of the grid is visible to the user. The circle indicates the viewing limit at a particular altitude. The highlighted region is the only part of the terrain data needed by the client.

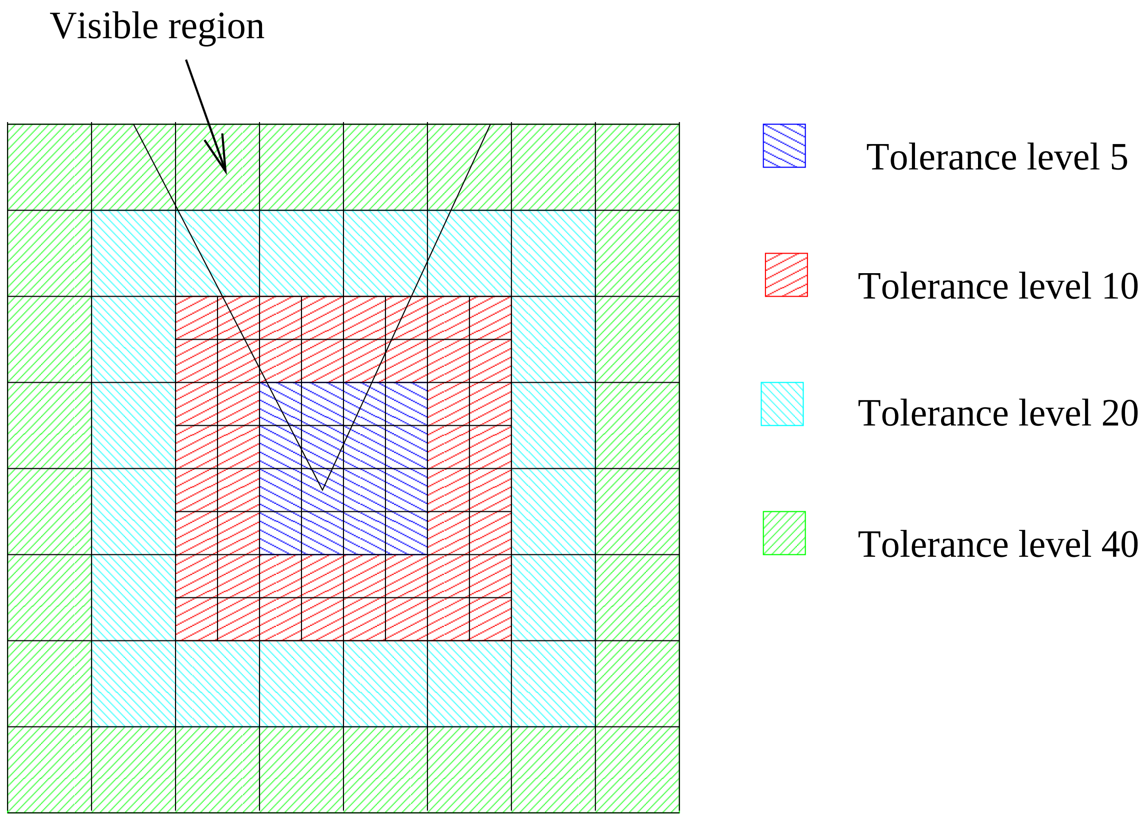

Figure 7: The resolution of the terrain model is reduced for positions further away from the user. In the center of the VRML world, the visible sensor grid has a smaller grid section. These grid sections are furnished with the high resolution terrain data. The smaller grid sector provides us a finer resolution on resolving the user view volume, so that a minimum number of high resolution grid section are used to render the image.

By making use of scripts, it may also be possible to dynamically adjust the VRML data on the client side, which would allow us to overcome the problem of matching edges of LOD nodes of different resolutions. A more intriguing possibility would be to dynamically create all the VRML files (as in figure 3) on the client side rather than the server side. This would enable us to greatly reduce the amount of data that needed to be downloaded.

7 Conclusions

We have shown that by careful choice of the structure of the VRML file, including the use of tilings to segment a large data set, optimized irregular meshes and texture mapping to reduce the number of polygons, LOD and WWWInline nodes to provide lower resolution data sets, and JPEG compression of the texture maps and gzip compression of the ASCII VRML file, it is possible to create a terrain model using VRML 1.0 that can be used for a modest area of terrain, as long as a low latency, high-bandwidth (broadband) network is used.

We have also highlighted the substantial advantages provided by the new features of VRML 2.0, and discussed some mechanisms that should allow much better performance for the visualization of terrain models of larger area. The use of sensors and Java scripts could provide a more effective use of available bandwidth, the hiding of latency by more effective prefetching of data, the dynamic loading of new terrain models, and the dynamic adjustment of data resolution to provide consistent frame rates for clients with different rendering capabilities and network bandwidths. We are aiming to implement such a terrain model using VRML 2.0.

Another possibility for future work is the use of Java, in particular the evolving Java3D class libraries for 3D modeling. The advantage of using Java is that being a programming language rather than a data format like VRML, it allows the developer a lot more freedom in controlling the 3D world. The advantage of using VRML is that viewers already exist, and programming a customized viewer using Java3D is a major undertaking. Ideally a marriage of VRML and Java3D would provide viewers that can easily be customized by the addition of Java3D modules.

Acknowledgements

We would like to thank Kryzysztof Walczak, Geoffrey Fox and Wojtek Furmanski for helpful discussions. This work was supported by the Northeast Parallel Architectures Center (NPAC) at Syracuse University. Additional support was provided by the On-Line Data Archives (OLDA) project, through the Research Data Networks (RDN) and Advanced Computational Systems (ACSys) Cooperative Research Centres (CRC), established under the Australian Government's CRC Program.

References

[1] Gavin Bell, Rikk Carey, and Chris Marrin. The Virtual Reality Modeling Language: Version 2.0 Specification, 1996. http://vrml.sgi.com/moving-worlds/index.html.

[2] Gavin Bell, Anthony Parisi, and Mark Pesce. The Virtual Reality Modeling Language: Version 1.0C Specification, 1996. http://vag.vrml.org/vrml10c.html.

[3] Thomas Boutell. PNG (Portable Network Graphics) Specification. http://www.boutell.com/boutell/png/.

[4] James D. Foley, Andries van Dam, Stenve K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, 2nd edition, 1992.

[5] Alvin Leung. *Real-time Interactive Client-Server Terrain Rendering*. PhD thesis, Syracuse University, 1997. Available from http://www.npac.syr.edu/users/alvin/thesis/.

[6] P. Peggy Li and David W. Curkendall. Parallel Three Dimensional Perspective Rendering. *Parallel Computing: From Theory to Sound Practice, Proceedings of EWPC '92*, pages 320–331, 1992.

[7] Mark Nelson. *The Data Compression Book*. M&T Books, 1991.

[8] Mark Pesce. *VRML: Browsing and Building Cyberspace*. New Riders, 1995.

[9] Silicon Graphics Inc. Moving Worlds. http://webspace.sgi.com/.

[10] Silicon Graphics Inc. OpenInventor. http://www.sgi.com/Technology/Inventor.html.