A Design Overview of a Real-time Terrain Rendering Program

Alvin Leung

October 25, 1995

Abstract

In this report, we provide an overview of the design of a real-time 3D terrain rendering program being developed for educational use. The terrain rendering program has four main viewing stages ranging from solar orbit to low altitude fly-by. Each stage has different viewing characteristics and data requirements. Hence, different rendering techniques, such as pixel mapping and bump mapping, are used to generate the images. These techniques are examined in detail. In addition, we briefly discuss the issues related to parallelization, data optimization and image compression.

1 Introduction

The goal of this terrain rendering project is to provide the user a real time interactive viewing environment for the available terrain data. We envision that the user starts out in the solar system, where Mars, Earth and Venus are visible. Then the user chooses to visit one of the three planets.

This journey can be broken into four main viewing stages: stage one, from solar orbit to high planetary orbit; stage two, from high planetary orbit to lower planetary orbit; stage three, from lower planetary orbit to high altitude flight path; and stage four, from high altitude flight path to low altitude fly-by. Each of these stages has distinct viewing characteristics that the terrain viewer must respect. As a result, multiple rendering techniques and data sets are needed to generate the images. In the following sections, the viewing characteristics, data requirements and rendering techniques of each stage are investigated.

2 From solar orbit to high planetary orbit

At this stage, the user is deciding which planet to visit. Hence, the basic viewing requirement is to provide enough visual cue to distinguish the planets. As the user moves closer to a specific planet, we would like to show the most striking features of the planet, for example, the continents and oceans on Earth and the polar caps of Mars. This information is intended to help the user properly orient to the viewer's 3D coordinate system.

2.1 Data Requirement

The data required for this stage is a collection of progressively refined pixel maps of the planets. These pixel maps need to correct for spherical projection to minimize the distortion during the rendering phase. The pixel maps can be derived from any source as long as "correct" colors are used for the major planetary features.

2.2 Rendering Technique

We can render the planetary images by mapping the planetary pixel maps onto the surfaces of the spherical objects (figure 1). Since the pixel map is constructed to cover the entire surface area of the sphere, the size of the sphere is not important. For simplicity, we use the unit sphere.

We use the following mapping functions to map a 2D image onto a 3D spherical surface (as shown in figure 2). The simplicity of the mapping functions can be exploited to reduce computation requirement by using look-up tables and linear interpolation for various values of ϕ and θ.

$$\begin{array}{rcll} u & = & \frac{\cos(\phi)+1}{2}, & \text{where } 0^\circ \leq \phi < 180^\circ \\ v & = & \frac{\theta}{360^\circ}, & \text{where } 0^\circ \leq \theta < 360^\circ \end{array}$$

The characteristics of this mapping are the elongation and compression of the 2D pixel map regions as shown in figure 3. The compression in the v direction is due to the reduction of the circumference of the circles, which are on the surface of the sphere and are parallel to $x-y$ plane. The compression in the u direction is caused by mapping half of the pixel map onto the region bounded by $60^\circ \leq \phi \leq 120^\circ$ and $0^\circ \leq \theta < 360^\circ$. The rest of the pixel map forms the two elongated regions.

This distortion, if applied to a planar Earth image, will shorten all land mass between the Tropic of Cancer and Tropic of Capricorn and lengthen

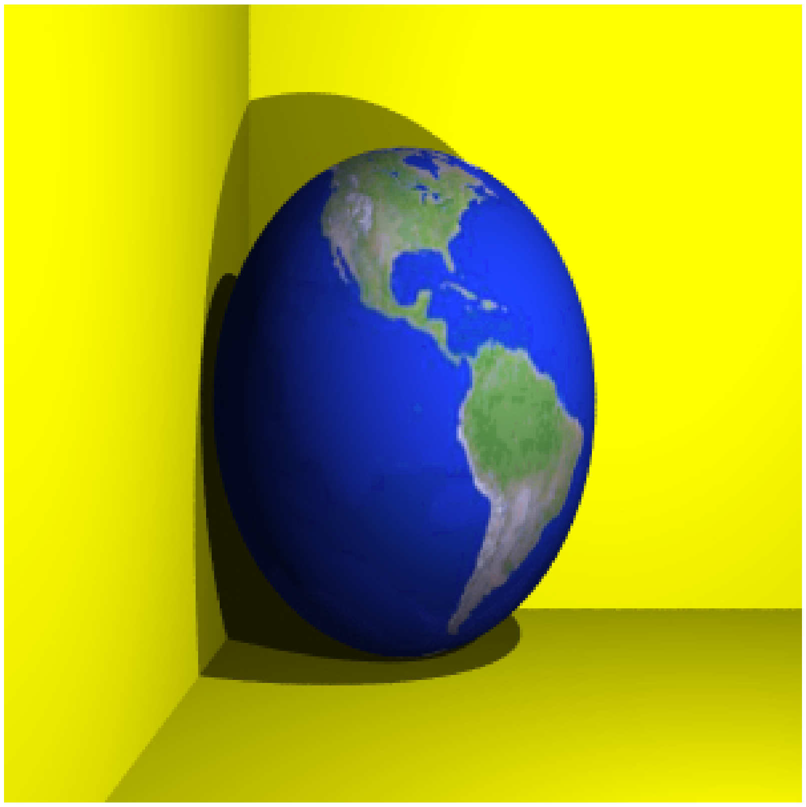

Figure 1: A texture mapped Earth.

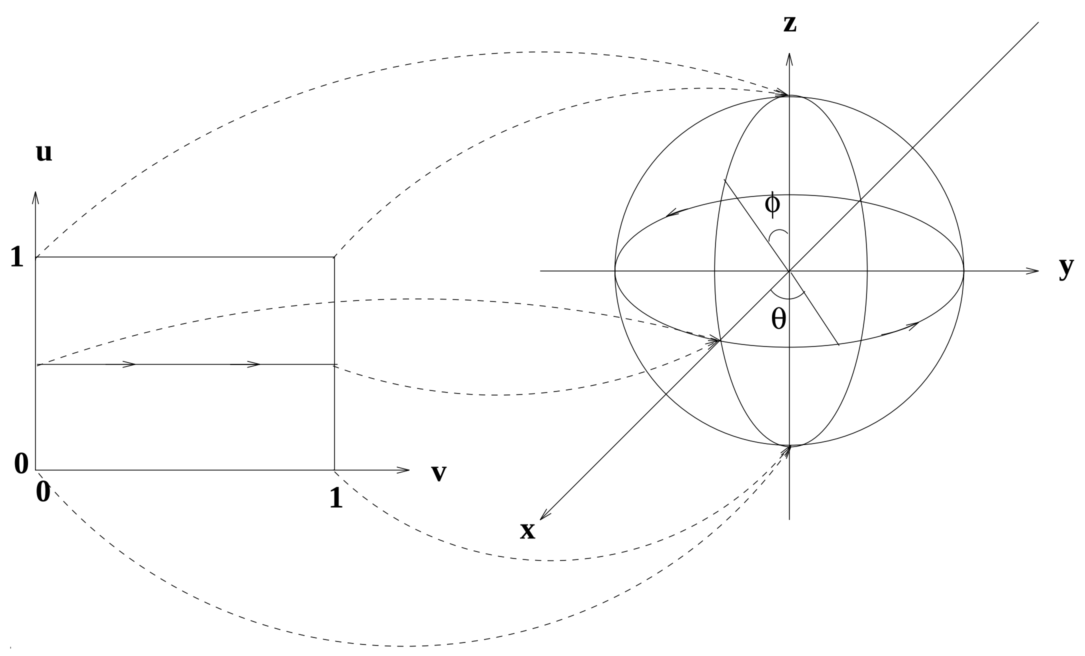

Figure 2: Mapping a 2D plane onto a 3D spherical surface.

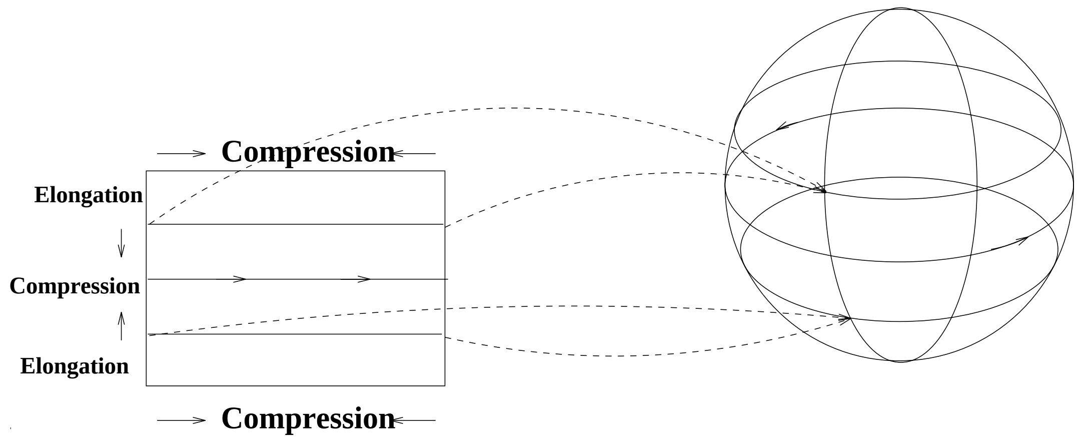

Figure 3: Compressed and elongated regions of the 2D plane.

all the land mass from these lines to the polar region in the u direction. Meanwhile, all land mass located away from the equator is compressed in the v direction. All continents will be distorted beyond recognition. Consequently, correctly prepared pixel maps are essential.

As the user moves closer to a planet, the user expects to see more detailed features of the planet. The rendering program can fulfill this expectation by using a higher resolution pixel map as the user gets closer to a planet. The exact relationship between distance and the resolution of the pixel map used will depend on the quality of the pixel map and the output screen resolution. Furthermore antialiasing can be accomplished easily, since the pixels are already loaded in core.

Combining the low computation requirement of this rendering algorithm with the low maintenance and low storage requirement of the pixel maps, we have an option to implement the first stage of the terrain rendering program on the client. This reduces the training equipment cost for the student to learn to use the terrain rendering program. Moreover, the first stage can also serve as a gateway to other related information server.

3 From high planetary orbit to lower planetary orbit

The user is selecting a Earth's continent size region for the journey. The user can either enter a coordinate or select a recognizable land mark as the starting point. A journey starting at a specified coordinate can be trivially accommodated. On the other hand, for the user to rely on recognizable landmarks will require the ability to distinguish mountain ranges, canyons, ice caps and large bodies of water. For most of these features, a pixel map contains enough information for the user to recognize the feature's characteristics. However, the user requires shadow to distinguish mountain ranges and canyons. Shadow computation is quite involved [2] [6]. At this altitude, the mountain ranges and canyons are just abnormalities (bumps) on the sphere surface. Since bumps do not cast very long shadows under most lighting condition, we can replace the full shadow calculation by a shading computation without significant lost of image quality. The lost image quality is caused by the shading calculation's failure to properly account for shadow casting (figure 4). A point is in shadow, if there is an object blocking the light rays from the light source to the point. The computation required to determine if a point is in shadow or not involves searching part of the

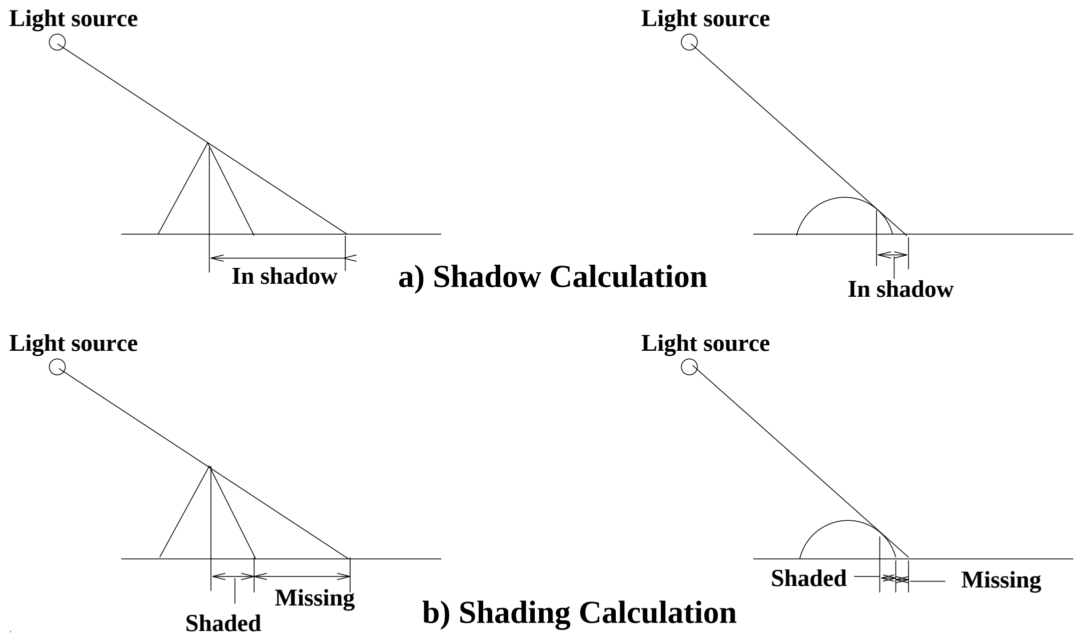

Figure 4: Difference in shadow calculation and shading calculation.

database for the light blocking object. Although this process is not particular challenging, it is CPU intensive. On the other hand, shading computes the brightness of a point according to the direction of the light source, the surface normal of the point, the viewer direction, and various adjustable parameters. Since no other object is considered, no shadow is generated. As a result, the shaded area is smaller than a properly calculated shadow and the error is proportional to the height of the shadow casting object and the angle of the light source. The shading of the terrain can be accomplish by using a displacement map (also known as bump map). In section 3.2, we will describe the algorithm to shade with a bump map.

In the terrain rendering program, the light source remains constant both in intensity and in location. The shading cannot be precalculated because the rotational rate of the planet does not remain constant as the user drops into the planet's gravity well. However, we will most likely to violate some physical laws to avoid disorienting the user. As a result, the shaded area has to brighten as the terrain features rotate toward the sun and to darken as they rotate away from the sun. In addition, experience has shown that continuously varying viewing parameters can help users to grasp the depth

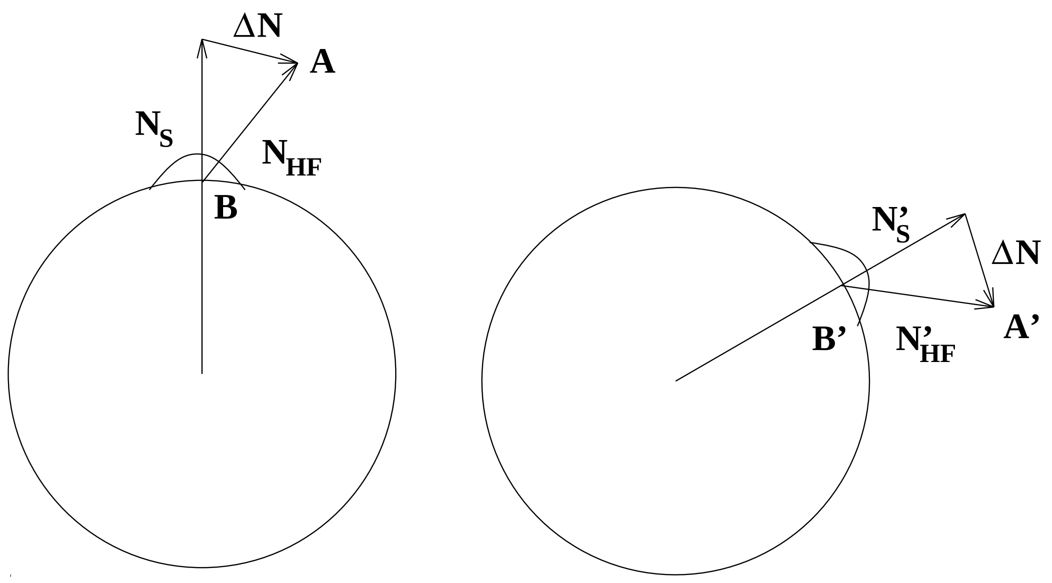

Figure 5: The construction of bump map

cue more easily [3].

3.1 Data Requirement

The data required for this phase is one high resolution pixel map and one bump map of the planet. Both of these maps are corrected for the spherical projection.

The displacement map can be constructed from the height field of the planet. Each entry of the displacement map consists of a small perturbation, $\triangle N$, of the sphere's surface normal, N_{S}. $\triangle N$ is computed by $N_{\mathrm{HF}} - N_{\mathrm{S}}$, where N_{HF} is the surface normal of the terrain (as show in figure 5). The directional information of N_{HF} is encoded with respected to the global coordinate system for a specific orientation of the sphere. The $\triangle N$ is in the local coordinate of each surface normal of the sphere. During the shading calculation, N'_{HF} is reconstructed by combining the perturbation with the surface normal of a view oriented sphere, N'_{S}. Consequently, the transformation to put N_{HF} in the viewer's perspective is avoided.

3.2 Rendering Algorithm

The rendering techniques used in this phase are texture mapping and Phong shading. The texture mapping algorithm remains the same as in section 2.2.

The basic concept of the shading is to multiply an intensity factor, I, to the RGB color value of each pixel. The intensity factor is computed by

$$I = I_P k_d N'_{HF} \cdot L ,$$

where I_P is the maximum intensity value, k_d is a diffuse reflection coefficient, N'_{HF} is the surface normal and L is the vector pointing toward the light source. The turnable parameters are I_P and k_d. I_P governs the brightness of the white light source. k_d control the reflectivity of the surface. By relating k_d with terrain type, we can make oceans and lakes be a bit more reflective than land to enhance the photo realism of the image. An example of a bump map is shown in figure 6.

For this rendering phase, the computation and the storage requirement are substantially more than the previous stage. As a result, the image will be generated on a high performance workstation and shipped across a fast network.

4 From lower planetary orbital to high altitude flight path

The user is further refining the tour destination by selecting a region approximately the size of a nation state. Terrain features reveal considerable details at this altitude. Planetary curvature is still noticeable, but much less pronounce than the previous two stages. The curvature of the terrain model should be reduced smoothly as the user moves closer to the planet. This will permit the user to adapt to the flat projected terrain model for the next stage. A transformation function is needed for the smooth transition between spherical projection and flat projection.

4.1 Data Requirement

The data needed in this stage is the lowest resolution of the terrain data sets. A terrain data set consists of vertices list, edges list, polygons list and texture maps for the polygons. The low resolution terrain data set has a significantly smaller number of vertices and the texture map resolution is much lower than

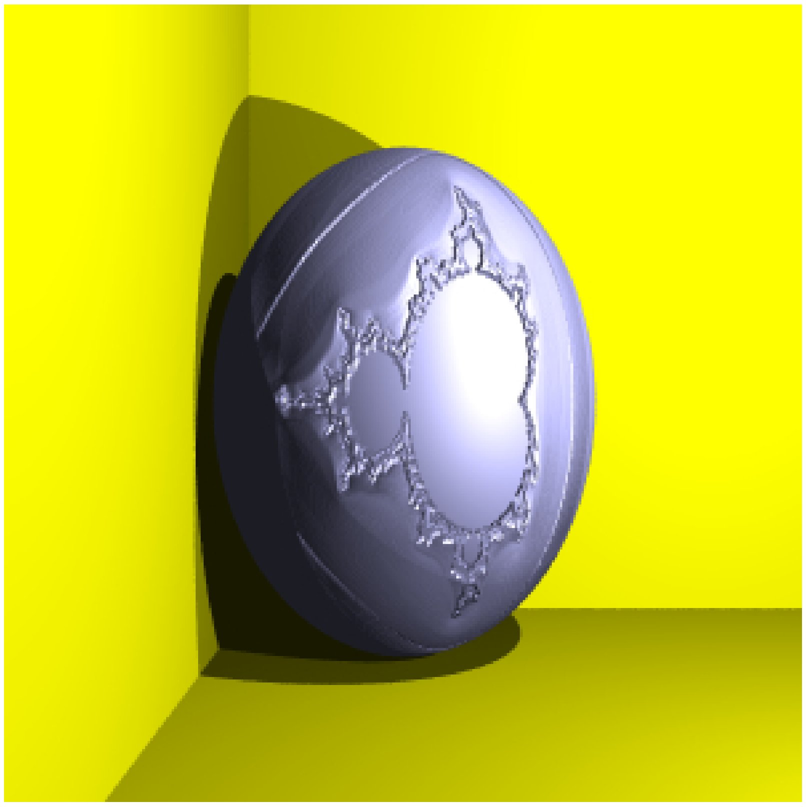
Figure 6: Mandelbrot set is rendered as bumps on sphere.

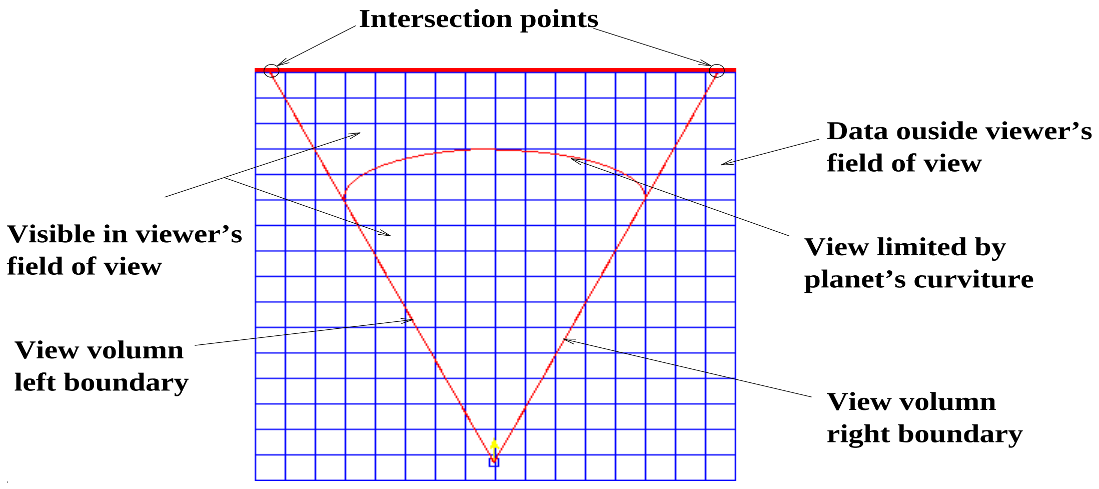
Intersection points
Data ouside viewer's field of view
Visible in viewer's field of view
View limited by planet's curviture
View volumn left boundary
View volumn right boundary

Figure 7: Finding visible data

the original digital elevation data and satellite image; however, the vertices are chosen to retain the details of the terrain features (see section 6 for more information). Furthermore, this data is prepackaged into subblocks. The size of the blocks are chosen to maximize the performance of the rendering phase.

4.2 Rendering Algorithm

Although the low resolution terrain data set is considerably smaller than the full data set, the low resolution data still has significant size due to the redundant information in the vertices, edges and polygons list. Consequently, only part of the data can be loaded in core. An efficient method to determine which part of the data is visible to the user is essential to achieve our goal of constructing a real-time interactive viewing environment. Identifying the visible data can be easily accomplished by using the information given by the viewer's field of view and the curvature of the planet (figure 7).

Since the terrain rendering program is generating images for a fly-by without any fantastic maneuvers, the viewing location and direction at every time step is very similar to the previous and to the next. To exploit the locality of these viewing parameters, the terrain data is first mapped onto a 2D grid and then the visibility identification is applied to the grid as presented in figure 7. Thus, only the visible data blocks are loaded from the secondary storage and processed in the rendering pipeline.

The visible calculation is essentially rendering a polygon, which bounds all visible data blocks, onto a very low resolution "screen," the 2D grid. The bounding polygon is formed by first projecting the user's 3D view volume onto the 2D grid. This gives us the left and right boundaries in figure 7. Then, by calculating the intersection points of the left and right boundaries with the 2D grid's boundary, we find the bounding polygon for all the visible data blocks. Finally, we apply a polygon scan conversion algorithm to mark the visible data blocks (refer to [3] [2] for more information on the polygon scan conversion algorithm). If the planetary curvature is in effect, then the distance from the viewer to each data block is computed. The data block is not visible if it is too far away with respect to the viewer's altitude. Since there is only one polygon with known shape being scan converted, the scan conversion subroutine can be optimized to reduce the computation cost.

The benefit of the visibility calculation is demonstrated in figure 8. The grey squares in the figure represent the visible data blocks and only these data blocks are rendered. Although only a tiny fraction of some of the visible data blocks are displayed, no special rendering techniques are performed to avoid loading these data blocks because they have a high probability for contributing more to the next image. After the viewer turned 4 degrees to the right (figure 8b) with turning rate 240 degree per minutes (40 revolutions per hour), the visibility calculation subroutine determined that two data blocks went out of view and three new data blocks come into view. As a result, choosing an optimal grid size can effectively cache the data blocks. Further, improvement can be made by predicting the viewer's movement.

After the data is loaded into the memory, the standard polygon rendering pipeline, which consists of transformation, clipping, hidden surface removal and rasterization, is used (see [2] for more information). The rendered image is than compressed (see section 6) and delivered via a height speed network to the client.

5 From high altitude flight path to low altitude flight path

By now, the user is examining a county size region at a relatively low altitude. The planetary curvature is not noticeable. Furthermore, the skyline is often masked by terrain features. At a result, we can use flat projected terrain data sets without significant compromising of the image quality.

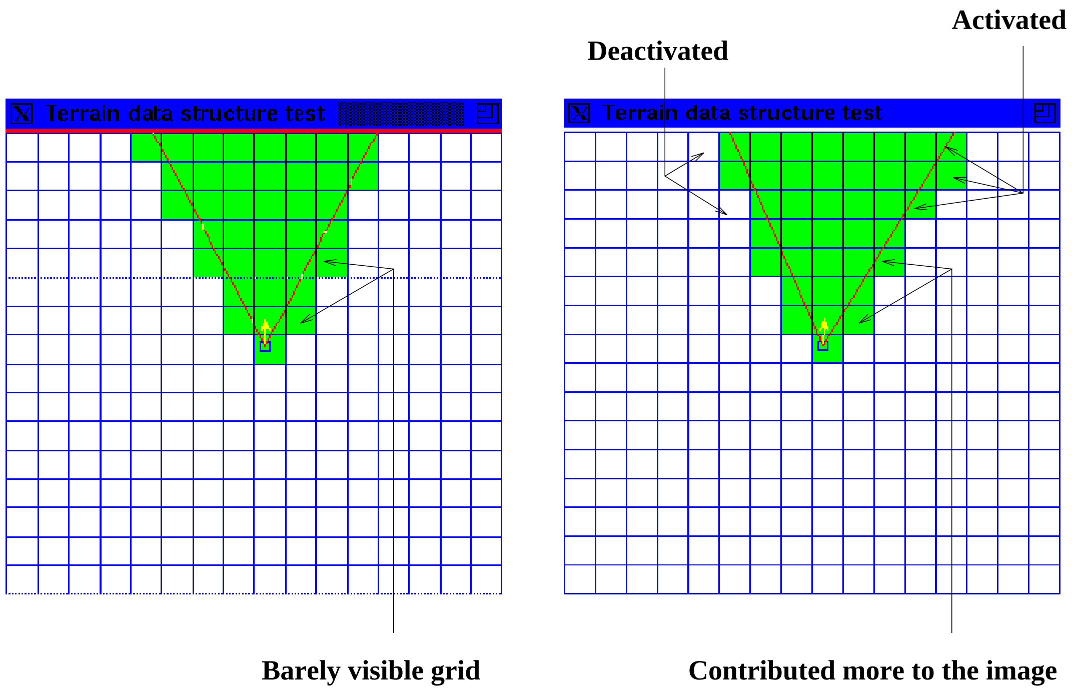
Terrain data structure test
Terrain data structure test
Activated
Deactivated
Barely visible grid
Contributed more to the image

Figure 8: Viewer has turned 4° to the right.

The user's field of view can be divided into far, mid and near field regions. The far field provides the viewer reference points to determine the relative special location and headings with respected to the planet. The mid field shows the relative location and heading of the user with respect to the impending target. The user can then line up the flight path for closer examination. The near field is for detailed examination of an area.

5.1 Data Requirement

The terrain renderer uses three different resolution terrain data sets to deal with the different viewing regions. The far region uses the previous low resolution terrain data, but the data is in flat projection. A medium resolution data set is used to generate the mid region image. This medium resolution data also passes through the data optimization treatment, but it uses more vertices to capture more terrain traits. The near region uses the finest resolution height field data and satellite image. A 2D grid is imposed onto the three data sets. Each data set is prepackaged into subblocks as in the previous stage. Because three data sets are used, we need to guarantee that the data sets merge seamlessly. By using the highest resolution data as the boundary points for all subblocks of the three data sets, we have the perfectly matched subblocks across all resolution data. The trade off is the increase of the number of vertices in the lower resolution data sets. The data optimization for the low and mid resolution is done within each of the subblocks but excluding the boundary points.

5.2 Rendering Algorithm

The rendering algorithm is essentially the same as for the previous stage. However, after the visibility identification procedure is done, the resolution of the data is selected according to the distance of the data block from the viewer. The far away squares will use the low resolution data. The near squares will use the high resolution data. Although the data are in different resolutions, the same rendering pipeline is used to render each subblock. The final image is then delivered to the client by a high speed network. Since this stage demands the highest computation capability for the server, it is best performed in parallel.

6 Future Work

There are at least three areas, parallelization, data optimization, and image compression, that need further study and are not covered in this report. First, in order to increase the rendering speed and the size of the data sets that can be handled, the terrain rendering program must be implemented in parallel. The parallelism of the terrain rendering is in the from of object parallelism, pixel parallelism and frame parallelism. The object parallelism derives from the terrain data's characteristics of non-overlapping and having an unambiguous depth order. Consequently, polygons can be rendered in any arbitrary order and the characteristics ensure the same image is generated. As in all other graphics applications, all pixels in an image are totally independent from each other. This implies the pixels can also be rendered in arbitrary order. By combining object and pixel parallelism characteristics, the terrain rendering program can distribute and render the data efficiently in parallel without compromising the image quality. Because the terrain program is interactive, the program has to generate multiple images per second. However, each image is generated with no dependency on the pixel values of the previous images. Consequently, the terrain rendering program can render a number of images for different time steps in parallel.

Second, because of the aforementioned viewing requirements and characteristics, data optimization is essential to reduce the needed computing power. The goal of data optimization is to reduce the number of vertices in the data set, while retaining the dominant features of the terrain. Although there are data optimization algorithms [5] [4] available, these algorithms are specifically design for optimizing data points that are captured by a 3D digitalization process. These data sets are usually extremely dense and unorganized. On the other hand, the terrain data is relatively lower in density and well organized. Furthermore, the data optimization algorithms tend to capture shape corners and edges very well. At the current resolution of the terrain data, sharp transitions are seldom. As a result, more study is needed to determine the usefulness of these algorithms on terrain data.

Third, video image compression is necessary to effectively use the network bandwidth. A side effect is that video image compression also provides a way to handle computing resources degradation. The MPEG video compression [1], for example, has three type of frames, intra-coded, predictive-coded, and bidirectional predictive-coded. An image is intra-coded, if it is encoded by using the information within that image only. The predictive-coded image uses motion-compensated prediction from a past intra-coded or predictive-coded images. In other words, only the difference of the previous image is sent to reconstruct the image. The bidirectional predictive-coded uses motion-compensated prediction from a past and a future frame, i.e., an interpolation of the two images can be formed. In a computing resources rich environment, each frame is generated and encoded with MPEG compression. In a computing resources poor environment, the information in bidirectional predictive-coded can be "approximated" from the viewing parameters. Hence, a less accurate image will be generated, but the "approximation" may take less CPU cycles to compute than rendering a full image.

References

[1] C. Wayne Brown and Barry J. Shepherd. *Graphics File Formats: reference and guide.* Manning, 1995.

[2] Paul Coddington. CPS 713: Terrain rendering for a geographic information system, 1994.

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

[4] Hugues Hoppe, Tiny DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. *Computer Graphics*, 27:19–26, August 1993.

[5] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. *Computer Graphics*, 26:65–70, July 1992.

[6] Stephanie Wierich and Paul Coddington. Real-time rendering for a geographic information system. Journal of Undergraduate Research in High-Performance Computing, NPAC technical report, SCCS-632, 1994.