Virtual Flythrough over a Voxel-Based Terrain

Ming Wan, Huamin Qu, and Arie Kaufman
Center for Visual Computing and Department of Computer Science
State University of New York at Stony Brook, Stony Brook, NY 11794-4400
{mwan, huamin, ari}@cs.sunysb.edu

Abstract

A voxel-based terrain visualization system is presented with real-time performance on general-purpose graphics multiprocessor workstations. Ray casting of antialiased 3D volume terrain and subvoxel sampling in true 3D space produce high quality images. Based on this rendering algorithm, an interactive flythrough system for mission planning and flight simulation has been developed on an SGI Power Challenge and a virtual reality environment using a Responsive Workbench. Arbitrary stereoscopic perspective views over the terrain and a 6D input device are supported.

1. Introduction

A virtual flythrough system over large areas of terrain has been a focus of 3D computer graphics for several decades. The virtual model display (VMD), such as the Virtual Workbench and the Responsive Workbench, provides a powerful tool for immersing the user within a computer-generated virtual terrain environment. In this paper, we explore the application of such a VMD in the virtual flythrough over voxel-based terrain, where both the immediate visual feedback and high-definition photo-realistic images are paramount.

The topographical and textural features of terrain are usually obtained from a 2D elevation map and a corresponding color aerial or satellite photograph. Two kinds of terrain modeling derived from the elevation map have been adopted for different terrain rendering algorithms: surface-based modeling, which uses a set of tiny triangles to cover the elevation grid [1], and voxel-based modeling, where terrain is represented as a view-independent volume buffer of volume elements (in short, voxels). Using current polygon accelerators, the surface-based terrain can be rendered much faster than a voxel-based one. Thus, most of the current flight simulators are based on the surface model. However, the pioneering volume rendering accelerator, VolumePro, by Mitsubishi Electric [2], based on Cube-4 [3], will support real-time volume rendering. Furthermore, the voxel-based approach has many advantages compared to a surface-based one. For example, the format of the elevation map lends itself to generating a very high resolution 3D volume of terrain and multiresolution volumes. Also, texture mapping for voxels is much simpler, higher quality, and can be preprocessed. More importantly, the voxel-based model is somewhat scene complexity independent, and it is easy to incorporate clouds, haze, flames, and other amorphous phenomena and volumetric objects [4]. Therefore, in recent years, the voxel-based approach has become the new trend for height field visualization systems [5, 6, 7, 8].

The main weakness of the voxel-based approach is that it is expensive in terms of computation and memory. This problem is particularly prevalent in applications dealing with large terrain datasets in virtual reality environments, where real-time update of the terrain scene being viewed is required. Many attempts have been made to speedup volume rendering for large height fields, including designing special-purpose hardware, exploiting multiprocessors, and exploring software acceleration methods. In this work, we are interested in parallelizing a fast volume rendering algorithm that relies on algorithmic optimizations. The most efficient acceleration technique published so far for voxel-based terrain rendering is to incorporate *ray coherence* into the ray casting algorithm, assuming that the terrain model has no 3D shapes such as overhangs, since the terrain is a height field [7, 8]. The basic idea of ray coherence is that, if two rays cast from the viewpoint are projected onto the same line on the base plane of the terrain, the higher ray hits the terrain surface at a position farther away from the viewpoint. Thus, the determination of the intersection between the higher ray and the terrain can be accelerated by starting sampling from the intersection depth of the lower ray rather than from the beginning. In order to decrease the memory of the terrain volume and its accessing time, the original elevation map is often used as 2.5D data [7, 8].

In our voxel-based terrain visualization system, the concept of exploiting ray coherence for accelerating ray casting

is applied during the rendering stage. However, in contrast to the previously proposed rendering algorithms which are based on 2D elevation maps, our proposed terrain rendering algorithm is performed on a true 3D voxel-based terrain model. Due to our pre-antialiased voxel-based terrain modeling and our subvoxel sampling in 3D terrain space, photo-realistic perspective images with reduced aliasing are generated from any view with six degrees of freedom. Real-time rendering rates have been reached by parallelizing the algorithm on a 16-processor SGI Power Challenge. Based on our modeling and rendering algorithms, a virtual flythrough system has been developed on both graphics workstations and a Responsive Workbench with natural 2D and 3D user interfaces. This system is quite applicable to environmental planning, geographic information systems, and, of course, in applications such as flight simulation, mission planning, and command and control.

2. Overview of the System

Our virtual flythrough system takes 3D volumetric terrain data as the virtual environment, which are reconstructed from 2D elevation maps and corresponding color photographs. The user can then interactively navigate over the virtual terrain in a manner similar to but more flexible than the real flythrough. Our system is designed for applications such as targeting and mission rehearsal, where the user needs to recognize the area or identify objects on the ground, rather than only train pilots. To implement this system, the following three stages are involved:

- Data Acquisition: A set of 2D elevation maps and corresponding color photographs, which covers a large range of terrain, is captured by satellites. In order to produce correct terrain information, image warping is used to correct the distortion of the maps due to downward tilt [9].

- Terrain Modeling: The 2D elevation maps and the corresponding color photographs are reconstructed into 3D volumetric terrain data. The reconstructed terrain surfaces in 3D space are then smoothed by using a 3D Gaussian filter or a filtered voxelization based on an oriented box filter [10]. In order to save both memory and accessing time of the terrain dataset during the subsequent navigation, the 3D terrain volume is further compressed into a 2D array.

- Terrain Navigation: Based on our terrain model, the system can interactively generate a sequence of 3D terrain images, while allowing the user to freely navigate over the terrain. Our efficient terrain rendering algorithm supports high-quality perspective projection, interactive rendering speed, and six degrees of freedom viewings.

Figure 1 shows the architecture of our virtual flythrough system. It has been implemented in a virtual reality environment with a Responsive Workbench. The requirements of the virtual reality environment for real-time rendering and high-quality images are met by our voxel-based terrain modeling and rendering algorithms. Our system has also been implemented on general-purpose graphics workstations for its widespread use.

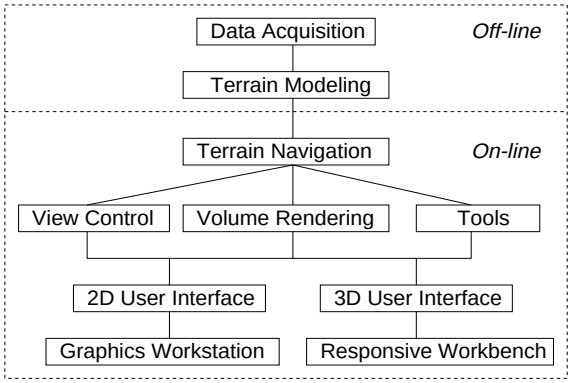

Figure 1. Architecture of our flythrough system.

Our virtual flythrough system is characterized by the following set of features:

- Dynamic Visualization: Our system provides two kinds of dynamic visualization modes: planned navigation and interactive navigation. While planned navigation can provide a general overview of the terrain along the pre-defined path, it is rather limited because no user interaction is possible. In contrast, the interactive mode gives the user an intuitive way to view a particular region of interest from different positions and orientations during the fly.

- High Quality Visualization: Mapping the digital color photograph onto the elevation map in our terrain modeling stage produces a high-quality photo-realistic view of the terrain. In addition, ray casting of pre-antialised 3D volume terrain and subvoxel sampling in true 3D space performed by our terrain rendering algorithm produce a higher image quality, compared with that reached by other voxel-based terrain rendering methods [7, 8].

- Collaborative Work: The users of our flythrough system often need to work collaboratively, as when planning a mission. Our Responsive Workbench is suitable for a group of people to collectively view, analyze and discuss the 3D terrain model that is projected onto a large table-top. Compared to head-mounted displays, this equipment allows 3D virtual model display for multiple users [11].

- Flexible Manipulations: Our system integrates several functions and tools accessible through the graphical user interface to provide a flexible manipulation environment. Natural 2D and 3D user interfaces have been developed on both SGI workstations and the Responsive Workbench.

Figure 2 shows the overview of the system interface implemented on an SGI workstation. In the upper panel, the photo-realistic 3D terrain image is created according to user-desired viewing parameters; the user can continuously

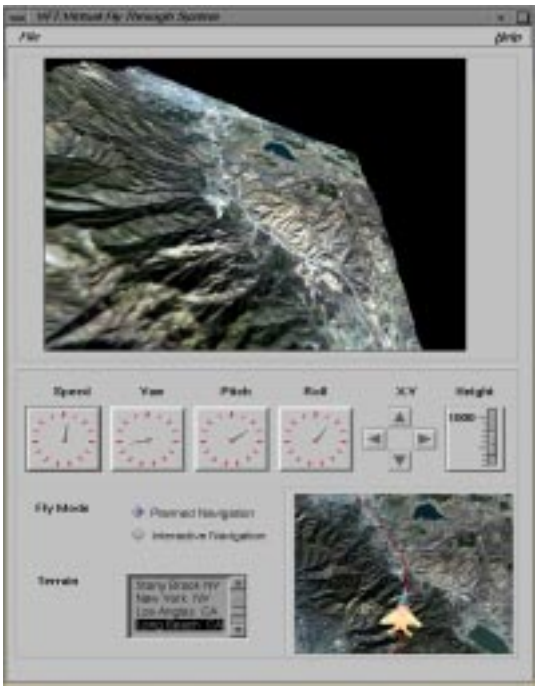

Figure 2. The interactive interface of our visual flythrough system on an SGI workstation.

interact with it by operating a mouse cursor to maneuver in this virtual world. The lower control panel provides several options to control such navigation and visualization operations as navigation speed, six degrees of freedom (yaw, pitch, roll, X and Y coordinates on the ground, and the height from sea level), navigation mode (planned or interactive navigation), and the selected terrain data. In addition, a 2D aerial color photograph is displayed in the lower right corner of the control panel with a tiny aircraft model superimposed, to track course in large terrains. The position of the aircraft indicates the current ground location of the viewpoint.

A superior 3D interaction is provided in our virtual reality environment with the Workbench, where the user can conveniently control the viewing position by moving and rotating the Ascension flock-of-birds trackers with six degrees of freedom. Stereoscopic viewing is provided by shutter glasses for several people simultaneously. Figure 3 shows our Workbench, which consists of an Electrohome 9500 projector, Crystal Eyes shutter glasses, PINCH gloves, 6D Ascension flock-of-birds trackers, and 3D sound.

3. Voxel-Based 3D Terrain Modeling

Terrain model design is an important component of a flythrough system, because it is closely related to the performance of the rendering algorithm with respect to both rendering rate and image quality. Unlike existing systems, in our virtual flythrough system, a pre-antialiased pre-textured 3D voxel-based terrain model has been generated.

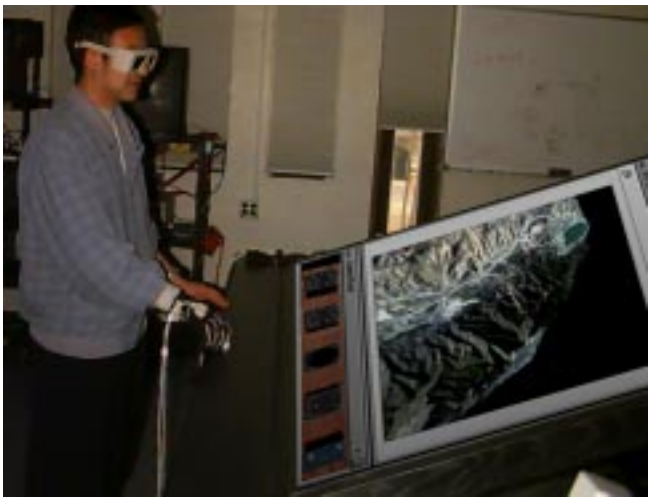

Figure 3. Our visual flythrough system on the Responsive Workbench.

It is well known that the standard ray casting algorithm [12] can produce high quality images from 3D volume data by sampling along each ray cast from the viewpoint through each image pixel into the volume dataset. In fact, both rendering speed and image quality depend on the sampling interval l. Higher values of l produce a faster rendering rate due to fewer sampling points along each ray, but reduce image quality with more missed features. Normally, it is recommended for a reasonable image quality that l be no more than the distance between two adjacent voxels of the volume. In particular, when a terrain dataset is rendered by the ray casting algorithm, there is no explicit intersection calculation, but a sequential search along each ray for the hit position of the ray and the terrain surface [8]. Evidently, both the rendering rate and the image quality of terrain are related to the accuracy of the hit information, which is directly affected by the sampling intervals along each ray.

Most of the published voxel-based terrain rendering algorithms directly use the 2D elevation map to represent 3D values, so that both memory size and accessing time for the terrain model can be greatly reduced [7, 8]. The elevation information of terrain stored in the elevation map describes terrain as a rectilinear grid of points elevated above a *base plane* or sea plane with a 0 elevation. With such a 2D terrain model, the samplings along each ray are performed only on the projection line L of the ray on the base plane of the terrain rather than inside the true 3D terrain space, and are always located on a set of parallel voxel layers. Accordingly, in such a discrete 2D grid traversal, the sampling interval l along the projection line L of the ray is always greater than

the resolution of the 2D elevation map. The corresponding sampling interval along the source ray in the 3D terrain space is even greater, especially when the ray becomes perpendicular to the base plane of the terrain. As we mentioned, the greater the l, the faster the rendering speed and the lower the image quality. That is why the frame generation rate decreases as the pitch angle reduces in [8].

Our terrain model combines a true 3D volumetric terrain dataset and corresponding aerial photograph or satellite image of the terrain. Voxel values of the terrain volume are obtained from an elevation map and filtered by digital filters such as Gaussian filter or the oriented box filter [10] to avoid aliasing in the resampling process. Assume that each voxel above and below the terrain surface has a density value of 0 and 255, respectively. After filtering, those voxels close to the terrain surface have transition density values between 0 and 255. Note that each column of voxels vertical to the base plane of the terrain has a constant color stored in the corresponding position of the terrain model, which we refer to as pre-texturing. These colors, obtained from the aerial photograph, are used to produce a photorealistic image by the rendering algorithm.

Thus, the 3D terrain model used in our virtual flythrough system is represented as a regular grid of voxels with density values between 0 and 255. A grid *cell* is defined as the volume contained within the rectangular box bounded by eight corner voxels from two adjacent parallel slices. The height field $S \subset R^3$ varies within each cell. An interpolation function, such as trilinear interpolation, is commonly used to reconstruct sample values within each cell. The volumetric dataset combined with the interpolation function defines S. The terrain surface in S is defined by a segmentation function Q, which partitions the voxels of volume S into two sets, either below or above the terrain surface.

If a cell contains voxels both below and above the terrain surface, the cell is pierced by the surface and we call it a *boundary cell*. The information of boundary cells is very important since, theoretically, sight rays always stop inside a boundary cell when they are cast towards the terrain. Since the determination of boundary cells is view-independent, we can obtain it by scanning over the volume in a preprocessing stage. In our algorithm, this is implemented by maintaining a flag F for each cell $C(i, j, k)$. The position of a cell $C(i, j, k)$ is determined by the voxel which has the lowest coordinate values i, j and k. Once a boundary cell is found during the volume scan, its flag is set to 1; otherwise, if all eight voxels of a cell are above or below the terrain surface, its flag is set to 0 or 2 respectively.

Note that in a cell column vertical to the base plane of the terrain, cells always appear in the order of below, on, and above the terrain surface from the bottom top. Thus, instead of using a full auxiliary 3D array to maintain a flag for each cell, we establish a 2D *boundary array*, recording the positions of both the lowest and highest boundary cells in each cell column. The use and importance of this flag array in our rendering algorithm is discussed in the next section.

4. Perspective Ray Casting of Voxel-Based Terrain

Based on our 3D terrain model, a fast ray casting algorithm incorporating ray coherence is conducted by traversing each ray inside true 3D terrain volume. High quality perspective images are produced due to more accurate hit information generated by subvoxel sampling in the antialiased 3D terrain volume.

Our algorithm is derived from the standard ray casting algorithm: each ray is cast from the viewpoint through each pixel of the projection plane into the terrain volume dataset. Once the ray enters the volume, a sequence of equidistanced samplings are taken along the ray, until the ray hits the terrain surface or exits the volume. At each sample point, a density value is reconstructed from the eight surrounding voxels by using a trilinear interpolation. If this sample value is greater than the density threshold defined by the segmentation function Q, a terrain color is sampled by bilinear interpolation from the neighboring voxel columns. Bilinear interpolation rather than trilinear interpolation is needed for color sampling, since each vertical column has a constant color. Using a sampling interval l which is lower than the resolution of the terrain volume produces high quality images, but at the same time presents a great challenge to interactive rendering rates.

The detection of the hit position for each ray in our rendering algorithm is accelerated by using the flag array of volume cells to avoid most of the time-consuming trilinear interpolation for each ray sampling. Evidently, if a ray hits the terrain surface, the traversal along it stops only when the ray enters a cell on or below the surface. Therefore, only after a ray enters a boundary cell from the air will the trilinear interpolation begin to be performed at each sampling point to determine the exact hit position.

A key advantage of ray casting over other volume rendering techniques is that ray coherence can be efficiently exploited for acceleration. Assume that there is no roll rotation, so that all pixel columns in the projection plane are perpendicular to the horizon of the terrain. According to ray coherence, a higher ray always hits the terrain at a greater distance than that of the ray below it. Therefore, once the intersection distance along the ray emanating from the bottom of a pixel column is detected, the traversal along the next ray emanating from the adjacent pixel on the same column can start from that intersection distance.

The idea of exploiting ray coherence in our rendering algorithm is similar to that in [8]. However, ray traversal

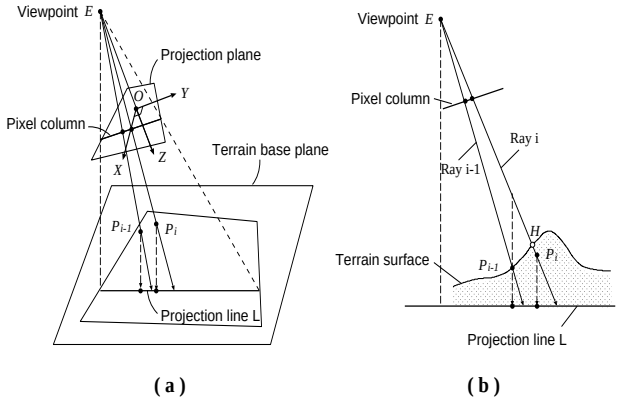

Figure 4. (a) The perspective projection geometry; (b) A cross-section of (a) along the plane defined by the rays from the same pixel column.

in our algorithm is performed with more accurate subvoxel sampling along the ray in 3D terrain space rather than along its projection line L on the terrain base plane (see Figure 4a). Specifically, assume that ray i and ray $i-1$ are cast from two adjacent pixels in the same image column, and d is the distance from the viewpoint E to the hit point P_{i-1} on ray $i-1$. According to ray coherence, the traversal along the higher ray i starts from the sampling point P_i, which has the same distance d from the viewpoint. Note that in our algorithm, the traversal from point P_i along ray i does not always go forward, differing from that in [7, 8]. Note that only when point P_i is located exactly above point P_{i-1} does the traversal go forward from point P_i to find the hit point. However, in our algorithm, the starting point P_i may not be exactly above the stopping point P_{i-1}. Thus, the hit point H along ray i may be already missed at point P_i, as shown in Figure 4b. Therefore, the traversal should retrace from P_i to find the terrain surface which is nearby. Also note that by such modification in exploiting ray coherence, our algorithm rectifies the potential image error reported in [8], when the planes defined by the rays emanating from the same pixel column are not completely perpendicular to the base plane.

The entire ray casting procedure can be further accelerated by completely avoiding the time-consuming trilinear interpolation operation at each sampling point. Note that most of the rays cast from the viewpoint stop at the boundary cells, the remaining rays stop at the cells just below the terrain surface, and no ray stops at the cells above the terrain surface. The lower the density threshold, the greater the chance that rays stop in the first boundary cell they encounter. Accordingly, we can approximately determine the hit point whenever the ray enters a boundary cell. From the experimental results we can see that the image quality is not significantly affected. This approximation not only accelerates rendering time, but also saves a lot of memory, since only the 2D boundary array in our terrain model is required during rendering. Such a feature of the approximation becomes very useful when a large area of the terrain consisting of many big terrain tiles is viewed. Only the terrain tile closest to the viewpoint is rendered on a true 3D terrain dataset with highest image quality. The other tiles are just approximately rendered with their 2D boundary arrays with much smaller memory requirements. A lower resolution of terrain tiles can further be used when they situate farther away from the viewpoint. In fact, the 2D elevation map used by other algorithms [7, 8] can be easily converted into a 2D boundary array. Consequently, both 2D elevation maps and 3D terrain datasets can be rendered by our algorithm with, obviously, different image qualities.

5. Roll Rotation

In a virtual flythrough system, it is important to specify the viewpoint with six degrees of freedom: x, y, z, roll, pitch and yaw. In an image space coordinate system, yaw is the rotation about axis Y (vertical), pitch is about axis X (horizontal), and roll is about axis Z (forward). Before we address this issue, a detailed account of the geometry of the perspective projection as shown in Figure 4a would be helpful. Assume the current viewpoint E is located at position (x, y, z) in object space, and the roll, pitch and yaw angles are respectively α, β, γ. The main direction or boresight of the 3D perspective view over the terrain can be completely determined by pitch angle β and yaw angle γ. The projection plane is oriented orthogonally to the boresight. When there is no rolling, the direction of the image column is always perpendicular to the horizon. Thus, a left-handed image space coordinate system is defined in our system as follows: the origin O is at the intersection point of the boresight and the projection plane, axis Z is along the boresight orthogonal to the projection plane, axis X is along the horizontal direction pointing right, and axis Y is orthogonal to both axes X and Z and pointing upward. The viewpoint thus has coordinates $(0, 0, -f)$ in image space, where f is the effective focal length of the projection. Accordingly, with viewpoint E, pitch angle β, yaw angle γ, and focal length f, the position of the projection plane is determined. In such a projection plane, all the pixel columns have the same upward direction perpendicular to the horizon, so that ray coherence is directly applied to the rays cast from the same column as described in the previous section.

However, when the image is rolling about the Z axis, the image columns are no longer perpendicular to the horizon. Thus, there is no vertical ray coherence between the rays cast from the adjacent pixels in the same column. Instead of using a special-purpose machine to implement the roll rotation [8], we propose an efficient software solution to implement roll rotation using our renderer.

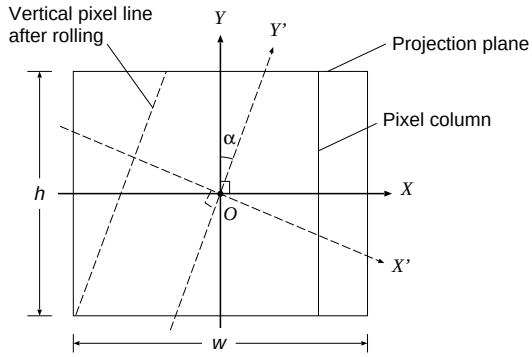

Figure 5. Roll rotation in the projection plane: When the image is rolled, axes X and Y are rotated to X' and Y'.

Assume that roll angle α is between $0°$ and $45°$ (the extension to other angles is straightforward). After rolling (see Figure 5), axes X and Y in image space are rotated to new directions X' and Y' (axis Z is unchange), so that the current axis X is always parallel to the horizon. Thus, the image can be divided into two parts by a line passing through the lower left corner of the image and parallel to axis Y'. Assume the width and height of the image are respectively w and h. w lines parallel to axis Y' can be generated to cover the right part of the image, each of which passes through a pixel at the bottom edge of the image. In our algorithm, these ideal parallel lines are represented by the pixel lines. We adopt a classic Bresenham scan conversion algorithm for lines, which is effective because only integer arithmetic is used. In our implementation, instead of separately performing scan conversion for each line, we only calculate the location of pixels for one line, and obtain the locations of the pixels on the adjacent line by shifting the current line by one pixel along the X axis, since these lines are parallel to each other. Furthermore, we do not even calculate the exact position of pixels on the projection plane. We only establish a 1D bit array B with length h. Each element of array B records the information of whether the corresponding pixel on the line shifts to a right image column or not. Therefore, during the rendering time, every line in the right part of the image is processed from the bottom pixel. The next pixel in the same line must be either the one above it in the same pixel column or the one to the right of the above one, depending on the shifting information saved in the corresponding element of B.

The lines which cover the left part of the image can also be generated by using the same array B, since they are also parallel to the lines in the right part of the image. The starting pixels for these left lines can be determined by scanning the left edge of the image bottom up. Specifically, if the element in B corresponding to the current pixel C on the left image edge indicates no shifting, we know that pixel C belongs to the current line starting from a lower pixel in the left edge of the image. Otherwise, since the current line shifts to the pixel to the right of pixel C, a new line starts from pixel C. Evidently, the number of lines covering the left image is equal to the number of the elements in B indicating shifting action.

Now that the whole image is covered by parallel pixel lines perpendicular to the horizon, ray coherence can be applied to generate the rolled image. Since the amount of computation to establish array B is very small, and is only done once for each image frame, the computational overhead for roll rotation is negligible.

6. Parallelism Acceleration

Our serial voxel-based terrain rendering algorithm has been parallelized on an SGI Power Challenge, a bus-based shared-memory MIMD (Multiple Instruction, Multiple Data) machine. The shared-memory architecture supports straightforward implementation of our algorithm.

In general, there are two main issues that should be considered in designing a parallel algorithm for a shared-memory architecture: dividing the total computation into tasks for each processing element (PE), and choosing appropriate synchronization mechanisms for these tasks. Object-based partitioning and image-based partitioning are two commonly used strategies for parallel volume rendering. In the object-based approaches, the volume data is partitioned into appropriate subvolumes and each PE is assigned a subvolume. Since in this case, the image space is shared by all PEs and the partial results from each PE must be composited together to form the final image, explicit synchronization of the PEs is necessary. However, in the image-based algorithms the partitioning is performed directly in image space. Each PE is responsible for generating a specific portion of the image, and the shared data structure is the volume space. Thus, less synchronization is required for this type of partitioning. We, therefore, adopted the image-based partitioning strategy in our algorithm.

An important consideration for designing a parallel algorithm is how to choose the work unit assigned to PEs, while minimizing load imbalances. In our ray casting algorithm, instead of sampling the whole ray, we skip over most of the empty space above the terrain surface by exploiting ray coherence on a set of rays cast from the same image column (when there is no roll rotation). The total traversal distance along each set of rays cast from the image column is almost the same, so that each set of rays has approximately an equal amount of work to perform for ray traversal. Therefore, we choose to use the static interleave partitioning technique; that is, the image is treated as a pool of columns, and each PE picks and processes a fixed number of image columns in an interleaved order. When the

image rolls, scan-converted pixel lines are assigned to each PE instead of image columns. Since the lengths of the pixel lines in a rolling image are not constant, the amount of work for different pixel lines may differ. However, the load imbalance is alleviated by the interleave partitioning strategy.

7. Experimental Results

To demonstrate the performance of our rendering algorithm which is the core of our virtual flythrough system, we have conducted experiments on an SGI Power Challenge with 16 R10000 processors (194MHZ) and 3GB RAM. The parallel code of the ray casting algorithm is implemented using the SGI multithread support library based on the Sequent Computer System parallel primitives.

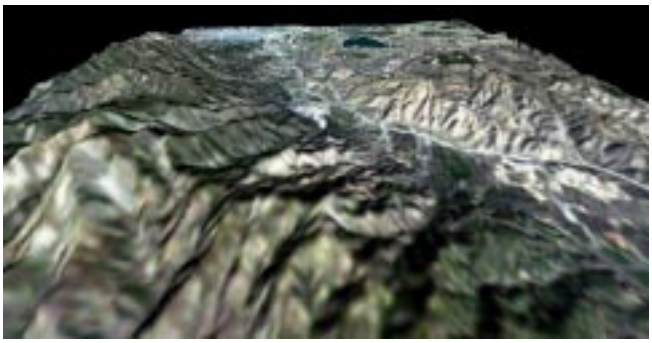

Figure 6. Accurate terrain rendering on an SGI Power Challenge in 0.09 seconds.

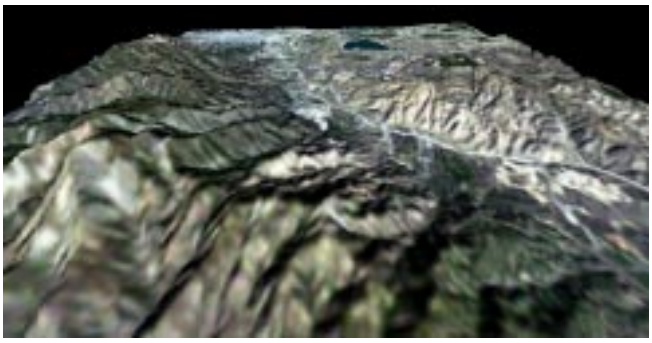

Figure 7. Approximate terrain rendering on an SGI Power Challenge in 0.06 seconds.

Figure 6 shows a view in southern California generated by our rendering algorithm. The size of the pitch, yaw, and roll angles are respectively $30°$, $90°$, and $0°$. The sampling interval is 0.6 compared to the resolution of 3D terrain volume with value 1. The image size is 500×400. Our terrain model consists of a 3D terrain volume with a resolution of $512 \times 512 \times 64$ and a corresponding aerial photo of size 512×512. For comparison, Figure 7 shows the same view generated using only the 2D boundary array instead of the 3D terrain volume. Since there is minimal difference in image quality between these two images, the approximate rendering method should be satisfactory for a terrain tile viewed from far. Table 1 presents the measured rendering times both for the exact and approximate images in figures 6 and 7 with different number of processors. Rendering rates of more than 10 frames per second have been successfully achieved by our rendering algorithm on 16 processors – very competitive to the rendering speeds accelerated by special-purpose hardware or multiprocessors with the same number of processors reported in the literature.

Table 1. Terrain rendering (TR) times (in sec) on SGI Power Challenge.

Processors:	1	4	8	12	16
Accurate TR	0.92	0.28	0.15	0.11	0.09
Approximate TR	0.65	0.21	0.11	0.08	0.06

When the image rolls as shown in Figure 2, the rendering time slightly increases compared to that of the image from the same viewing position without rolling. This is because the number of image columns is the lowest limit of the number of scan-converted pixel lines, and each line needs more calculation time for the first ray cast from the bottom pixel which can not be accelerated by ray coherence. When the roll angle is between $0°$ and $45°$, for example, the closer the roll angle is to $45°$, the more the lines, and therefore the longer the rendering time. Yet, since the total number of pixels in the image – which dominates the rendering time of the ray casting algorithm – is fixed, the time variation is insignificant. From our experimentation, the shortest rendering time for a uniprocessor is 0.92 sec when the roll angle is $0°$; the longest rendering time is 1.04 sec when the roll angle is $45°$, and the average time calculated from 10 continuously changing roll angles from 0 to 45 is 0.96 sec.

In order to show the performance of ray coherence acceleration, we measured the conventional ray casting times by always starting ray sampling and compositing from the viewpoint. We found that the normal ray casting method spent 6.71 sec to render the same image as Figure 6 on a uniprocessor. This speed is nearly 7 times slower than our ray coherence accelerated rendering rate, and about 10 times slower than our approximate rendering rate.

For purposes of comparison, another experiment has been conducted for terrain rendering by using a surface-based model. Each quad element of the 2D elevation grid was covered by two adjacent triangles, and then projected onto the projection plane by a graphics accelerator. To generate a full resolution photo-realistic terrain image as we did with our voxel-based approach, the corresponding

color photograph was employed for hardware texture mapping with no polygon simplification. The resultant surface-based rendering rates showed that about 0.80 sec were required to produce a similar image of Figure 6, which was slower than the volume rendering rates with multiprocessors shown in Table 1. Evidently, the large number of polygons ($512 \times 512 \times 2$) as well as the texture mapping operations limit the interactive surface projection speed.

8. Conclusions and Future Work

We have developed a virtual flythrough system on both graphics workstations and a virtual reality environment with a Workbench. This system is based on a fast terrain rendering algorithm supporting six degrees of freedom viewing during the flythrough with high image quality. The ray casting algorithm has been accelerated by ray coherence. More than 10 frames per second rendering rates have been reached by parallelizing the serial renderer on an SGI Power Challenge with 16 processors. The high quality image is ascribed to both the antialiased 3D terrain model and ray traversal in true 3D space with accurate subvoxel sampling.

Our results reported in this paper are part of the project entitled "Real-Time Retargeting using Volume Graphics", supported by Naval Research Laboratory (NRL) and "Techniques for Volume Graphics", supported by Office of Naval Research (ONR). This project deals with the representation and realistic visualization of large terrain datasets and 3D objects in terrain scenes using a voxel-based approach, coupled with mechanisms for generating perspective views to support real-time flythrough capabilities. The ultimate goal is real-time retargeting, in which the terrain scene can be rapidly modified as new information arrives. Except for the modeling and rendering techniques for terrain datasets presented in this paper, our current project work includes the representation and manipulation of other 3D volumetric objects, such as stationary buildings, trees, clouds, and moving vehicles and planes. We have combined these 3D objects with terrain to generate richer and practical scenes. Also, 3D sound from aircrafts and vehicles has been incorporated to produce a more realistic environment.

Acknowledgments

This work is partially supported by NRL grant N00014961G015, NSF grant MIP9527694, ONR grant N000149710402, and NASA grant NCC25231. Thanks to Edmond Prakash, Lichan Hong, Suya You, Milos Sramek, Kenneth Gordon, Bai Wang, Silvia Piquet, Baoquan Chen, and other members of the virtual flythrough project.

References

[1] P. Lindstrom, D. Koller, W. Ribarsky, L. Hodges, N. Faust, and G. Turner. Real-Time, Continuous Level of Detail Rendering of Height Fields. *Proc. SIGGRAPH' 96*, 109–118, New Orleans, Louisiana, August 1996.

[2] R. Osborne, H. Pfister, H. Lauer, N. McKenzie, S. Gibson, W. Hiatt, and T. Ohkami. EM-Cube: An Architecture for Low-Cost Real-Time Volume Rendering. *1997 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, 131–138, ACM Press, August 1997.

[3] H. Pfister and A. Kaufman. Cube-4: A scalable architecture for real-time volume rendering. *Proc. IEEE Symposium on Volume Visualization' 96*, 47-54, ACM Press, October 1996.

[4] A. Kaufman, D. Cohen and R. Yagel. Volume Graphics. *Computer*, 26(7):51–64, July 1993.

[5] P. Robertson. Fast Perspective Views of Images Using One-Dimensional operations. *IEEE Computer Graphics and Applications*, 7(2):47–56, February 1987.

[6] J. Wright and J. Hsieh. A Voxel-Based, Forward Projection Algorithm for Rendering Surface and Volumetric Data. *Proc. IEEE Visualization '92*, 340–348, Los Alamitos, CA, October 1992.

[7] C. Lee and Y. Shin. An Efficient Ray Tracing Method for Terrain Rendering. *Pacific Graphics '95*, 181–193, 1995.

[8] D. Cohen-Or, E. Rich, U. Lerner, and V. Shenkar. A Real-Time Photo-Realistic Visual Flythrough. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):255-265, September 1996.

[9] G. Wolberg. *Digital Image Warping*, IEEE Computer Society Press Monograph, 1990.

[10] M. Sramek and A. Kaufman. Object Voxelization by Filtering. *Proc. IEEE Symposium on Volume Visualization '98*, 111-118.

[11] E. Lantz, S. Bryson, D. Zeltzer, M. Bolas, B. de La Chapelle, and D. Bennett. The Future of Virtual Reality: Head mounted displays versus Spatially immersive displays. *Proc. SIGGRAPH '96*, 485–486, August 1996.

[12] M. Levoy. Display of Surface from Volume Data. *IEEE Computer Graphics and Applications*, 8(5):29-37, May 1988.