Time-Critical Volume Rendering

Final Report to NASA Ames

Cooperative Agreement No. NCC2-5231

Research Foundation Account No. 431-0979A

Project Director: Arie Kaufman

April 24, 1998

Time-Critical Volume Rendering

Abstract

For the past twelve months, we have conducted and completed a joint research entitled "Time-Critical Volume Rendering" with NASA Ames. As expected, High performance volume rendering algorithms have been developed by exploring some new faster rendering techniques, including object presence acceleration, parallel processing, and hierarchical level-of-detail representation. Using our new techniques, initial experiments have achieved real-time rendering rates of more than 10 frames per second of various 3D data sets with highest resolution. A couple of joint papers and technique reports as well as an interactive real-time demo have been compiled as the result of this project.

1 Introduction

Since March of 1997, we have been supported by NASA Ames to conduct a joint research entitled "Time-Critical Volume Rendering". This study focuses on the control of time in volume rendering of regular gridded data, for coping with real-time constraints and ensuring that the rendering computation does not take more than an allotted time, known as time-critical rendering [1]. The two primary techniques we have explored are fast volume rendering algorithms and hierarchical level-of-detail representation. Our strategy for managing time is initially to select the appropriate level of detail that can be rendered in the allotted time, and then our fast volume rendering algorithm is employed.

An effective approach to achieve high frame rates for volume rendering is to parallelize a fast rendering algorithm that relies on algorithmic optimizations in order to reduce the computational costs [2, 3, 4]. Many acceleration techniques have been proposed to improve the serial volume rendering rate [5, 6, 7, 8]. Our previously proposed PARC (Polygon Assisted Ray Casting) algorithm [9] is one of the most efficient acceleration technique among them. PARC uses pre-calculated subvolumes aligned with the primary axes to bound the object inside the volume. For each particular view, the subvolumes are projected into a pair of front and back z-buffers to obtain closer bounds on the intervals where ray integrals need to be calculated. Projection of the subvolumes can be performed quickly on graphics hardware. However, experimental results have shown that, even with the hardware acceleration, projection time is prohibitively expensive when the subvolumes become smaller for more accurate bounding information. Furthermore, when there is no hardware acceleration available on some workstations, or when the PARC algorithm is to be parallelized on multiprocessors, the projection procedure will be implemented in software, leading to an even worse estimation overhead [3].

In this project, we have proposed a boundary cell-based ray casting algorithm [10]as an improvement of the PARC algorithm. A more accurate estimation of the object boundary is obtained based on the tiny grid cells. Meanwhile, the boundary estimation time has been reduced by employing a fast incremental parallel projection strategy, including use of projection templates. The subsequent ray traversal procedure is further accelerated not only by using existing ray casting optimizations,

such as adaptive image sampling [5] and early ray termination [8], but also by developing a new acceleration technique [11] to reduce the amount of computation of trilinear interpolation during the ray sampling procedure. A high performance parallel rendering algorithm [12] based on this fast serial algorithm has been proposed and implemented on an SGI Power Challenge with 16 processors. The hierarchical level-of-detail representation has been generated from the source 3D data sets by using a suitable 3D filter. Experimental results have shown that our renderer is capable of rendering 256^2 high quality parallel-projection images at rates exceeding 10 frames per second for 3D data sets with highest resolution.

2 Accomplishment

During the research of this project, we have proposed and implemented a serial fast boundary cell-based ray casting algorithm, an optimal interpolation technique for ray sampling procedure, and a high performance parallel ray casting algorithm based on accelerated serial ray casting algorithm. We also established hierarchical level-of-detail representation of 3D data sets and developed a user-interface for interactive exploration of 3D data sets rendered by our fast algorithms.

2.1 Boundary Cell-Based Ray Casting

We have proposed an fast serial ray casting algorithm [10] as an improvement on the PARC algorithm, which accelerates the ray casting procedure by skipping over empty space in volume datasets. A more accurate estimation of the object boundary is obtained based on the boundary cells in the volume with less computational overhead by using incremental parallel projection for each boundary cell.

The original volumetric data set is typically represented as a regular grid of volume elements (*voxels*) with scalar values. A *grid cell* is defined as the volume contained within the rectangular box bounded by eight corner voxels from two adjacent parallel slices. a *boundary cell* is a cell containing the object boundary inside the volume. Our fast rendering algorithm consists of three steps: (1) detect boundary cells in a preprocessing stage; (2) project the boundary cells onto the image plane to produce the intersection distance values for each pixel; and (3) for each viable ray that intersects the object, sample and composite starting from the intersection.

Boundary Cell Detection

Since the information of boundary cells is viewpoint-independent, we can obtain it by scanning the volume in an off-line preprocessing stage. In our algorithm, this is implemented by maintaining a flag F for each cell $C(i,j,k)$. Once a boundary cell is found during the volume scan, its flag is set to 1. Thus, in the subsequent on-line projection procedure, only those cells whose flags are set are projected onto the image plane.

Boundary Cell Projection

Instead of separately projecting individual boundary cells onto the image plane to obtain the intersection distance information, we propose a more efficient software method to accelerate the parallel projection of each boundary cell.

Note that, in parallel projection, the projected area of every cell has the same shape and size in the image plane. Only the projected position and the distance of the cell center to the image plane are different from cell to cell. Therefore, once the first cell $C(0,0,0)$ is projected onto the image plane, the projected positions and the depths of the remaining cells can be quickly calculated by incremental vectors with only addition operations. The depth information of each projected

boundary cell is saved in a pair of front and rear z-buffers Z_n and Z_f, which respectively record the closest and farthest intersection distances to the object boundary along the rays cast from the image pixels.

Ray Traversal

Depending on the intersection distance information in the projection buffers Z_n and Z_f, the ray casting procedure is accelerated by casting rays only from the viable pixels on the image plane, and traversing each ray from the closest intersection to the farthest intersection. Some existing ray casting optimizations, such as adaptive image sampling [5] and early ray termination [8], have been conveniently incorporated to speedup our ray traversal procedure.

2.2 Optimal Interpolation for Ray Casting

To further accelerate the ray traversal procedure in our boundary cell-based ray casting algorithm, we propose another novel acceleration technique [11] to reduce the computation, particularly for the trilinear interpolation operation during the ray traversal procedure, by exploiting the special space distribution of voxels within the regular 3D grid.

A detailed study of the ray traversal procedure has shown that the interpolation operation, such as trilinear interpolation, applied on each sample point along each ray is very time-consuming. However, if we carefully choose the interval value of l between adjacent sampling points according to each specific ray direction, and control the location of the first sample point, we can greatly increase the opportunity to have as many sample points lying exactly on the voxel layers as possible, where only bilinear interpolation rather than trilinear interpolation is needed. Thus, the total amount of computation for ray traversal procedure can be reduced.

Based on above consideration, we have accomplished our acceleration by four steps on each ray: determine the reference axis perpendicular to the parallel voxel layers which are traversed fastest by the current ray; decide the first sampling position when the ray reaches the volume; calculate the value of the sampling interval l to make sure that every alternate sampling point lies on a voxel layer; use bilinear interpolation for each sampling point lying on the voxel layer, and use simplified trilinear interpolation for the other sampling points lying between the voxel layers.

Somewhat similar to Levoy's optimization of adaptive image sampling within the image plane [5], we have applied adaptive depth sampling along each ray to accelerate the rendering rate. This optimization is of particular efficiency when combined with our algorithm. In our algorithm, the amount of interpolation computation for a sample point lying between two parallel voxel layers is greater than that for a sample point lying on a voxel layer. When the adaptive depth sampling strategy is used, most of the time-consuming sampling points between the voxel layers can be skipped. Consequently, our algorithm will achieve better speedup. Experimental results showed that our algorithm saved about 70% ray traversal time by just skipping over every other sample point along each ray.

2.3 High Performance Parallel Ray Casting

Our most significant contribution in this project is the design of a high performance parallel volume rendering algorithm [12] that is based on our fast serial ray casting algorithms.

In the design and implementation of a parallel algorithm, the most important issue by far is to employ an efficient task partitioning scheme with good load balancing. In general, there are two types of task partitionings for parallel volume rendering algorithms: object-based partitionings and image-based partitionings. Most published parallel algorithms face the dilemma of having to select one from either object-based or image-based partitionings and give up the advantages of the

other. Yet, in our algorithm, novel load balancing schemes are designed that benefit from both object-based and image-based partitionings by taking full advantage of the optimizations in our serial algorithm as well as the shared-memory MIMD architecture of the SGI Power Challenge. Specifically, an object-based partitioning is employed to parallelize the boundary cell projection procedure, and an image-based partitioning is adopted for the ray traversal procedure.

Object-Based Partitioning for Boundary Cell Projection

Depending on the boundary cell distribution information obtained in the preprocessing stage, we are able to precisely divide the volume into subsets, each containing an equal number of boundary cells. Therefore, a static contiguous object-based partitioning is employed in our algorithm.

Once the volume is equally divided according to our static contiguous object-based partitioning, each processor concurrently and independently works on one subset of the volume assigned to it by scanning over the subset and projecting every detected boundary cell inside the subset onto the image plane. Within about the same period of time, each processor finishes its work and supplies a pair of partial projection buffers. The complete projection buffers of the whole volume can be obtained by combining all of these partial projection buffers.

Image-Based Partitioning for Ray Traversal

In our algorithm, the projection buffers established in the boundary cell projection procedure not only provides close bounds on the intervals where the ray integrals need to be calculated, but also gives important information about the view-dependent distribution of the image complexity. Thus, a static contiguous image-based partitioning can be employed to divide the whole image into large blocks consisting of consecutive scanlines, one block per processor with roughly an equal amount of work. On each processor, ray integrals are performed within the bounded intervals along those rays cast from the viable pixels inside the assigned section of the image. The empty image scanlines can be efficiently skipped according to the information in the projection buffers.

This parallel ray traversal procedure is also accelerated by the ray casting optimizations used in our serial ray casting algorithm, which fall into two classes according to whether or not there are computational dependencies between different rays. The optimizations in the non ray-dependent class, such as the early ray termination and the adaptive depth sampling, which are performed independently along individual rays, can be directly applied with our image-based partitioning. However, extra attention needs to be paid when incorporating the optimizations in the ray-dependent class, such as the adaptive image sampling, to avoid the cost of replicated ray casting from pixels shared by image sections assigned to different processors.

Nieh and Levoy [2] have proposed a dynamic image-based partitioning scheme which reduces the cost associated with pixel sharing by delaying the evaluation of sample regions whose pixel values are being computed by other processors. In this project, we propose a simpler but more effective solution based on our static contiguous image-based partitioning, which completely avoids such costs without extra synchronization overhead:

1. Divide the image into small square sample regions measuring I_{max} pixels on each side;
2. For P processors, separate the image made up of sample regions into P contiguous large blocks with roughly equal amounts of work;
3. Each processor takes one image block and calculates its sample regions in scanline order from top to bottom using adaptive image sampling.

Note that the pixels shared by image sections processed by different processors are only located in those square sample regions at the top and bottom of each image section. Each processor only need to calculate the shared pixels in the top sample regions. When it reachs the bottom sample regions, the shared pixel values in these regions are already available in the shared-memory, which have been calculated by its lower neighboring processors. Thus, the cost of replicated ray casting associated with pixel sharing in adaptive image sampling is completely avoided by taking advantage of the good load balancing resulting from our static contiguous image-based partitioning.

2.4 Hierarchical Level-of-Detail representation

We have also generated multi-resolution hierarchies for the 3D data volumes with gradual elimination of high-frequency features. Thus, when the rendering of a high resolution data set has exceeded the allotted time, a smaller data set with a lower resolution is to be rendered instead.

In our approach the input dataset is sampled and low-pass filtered into a lower resolution dataset. By establishing the direct correspondence between the size of the filter support and the resulting resolution, a hierarchical level-of-detail object representation is constructed. This topology simplification algorithm is simple, robust, and widely applicable. Unlike, for example, the 3D mip-map approach, in our approach every detail of the dataset hierarchy is created by convolving the original dataset with a low-pass filter of an appropriate support, thus, errors caused by a non-ideal filter do not propagate and accumulate from level to level. Furthermore, the sampling resolution of our dataset hierarchy need not be a power of two. In fact, the resolution difference between any two adajacent levels in our hierarchical representation is small, so that the change of image qualities rendered from adjacent level-of-detail data sets is not significant.

2.5 Interactive user-interface

We have developed interactive user-interfaces both on SGI workstations and a virtual reality environment with a workbench.

The Immersive Workbench is a new generation virtual environments equipment which allows a group of people to collectively view, analyze and discuss the 3D object which is projected onto a large table-top. The workbench we used consists of an Electrohome 9500 projection system, three Crystal Eyes shutter glasses, a pair of PINCH gloves, three six-degrees-of-freedom Ascension flock-of-birds trackers, and a software library. By moving and rotating an Ascension flock-of-birds tracker with six degrees of freedom, the user can conveniently control the viewing positions and directions. High realistic stereo effect produced by compositing two images generated from the left eye and the right eye can be available by wearing the shutter glasses. On SGI workstations, the control of the six-degrees-of-freedom tracker is simulated by using a mouse on the 2D control panel.

3 Experimental Results

To demonstrate the performance of our fast ray casting algorithm, we have conducted experiments on a 16-processor SGI Power Challenge with three data sets: a simulation data of a continuous scalar electric field due to five individual charges with data resolution of 50^3, a negative potential of a high potential iron protein (*neghip*) with resolution of 68^3, and a human brain CT data set with resolution of $256 \times 256 \times 124$. Rendering results are presented in Table 1. These results include the boundary cell projection time and the subsequent ray traversal time, but do not include image display time and the off-line preprocessing time to detect boundary cells. The size of each square sample region for adaptive image sampling optimization was 3×3 pixels, where the thresholds

of color differences for pixel values interpolation were respectively 10, 40, and 30 for the electric field, *neghip*, and the brain data sets. The thresholds of density differences for sample point values interpolation along each ray were respectively 30, 80, and 5. Early ray termination opacity thresholds of 98%, 92%, and 98% were used respectively for each data set. Figure 1, Figure 2, and Figure 3 show the images rendered by our fast ray casting algorithm for the three data sets with image size of 256^2. From Table 1, we can see that more than 10 frames per second rendering rates have been successfully achieved on 16 processors for all these three data sets with the highest resolution. Evidently, the rendering speed for the smaller data set with lower resolution is even faster.

Table 1: *Volume rendering times (in sec) for various data sets.*

Processors	1	4	8	12	16
electric field	0.39	0.11	0.06	0.05	0.04
neghip	1.02	0.28	0.15	0.12	0.09
brain	0.93	0.28	0.17	0.11	0.09

4 Summary

In this project, we have designed a couple of fast volume rendering algorithms and established hierarchical level-of-detail representation for various 3D regular gridded data sets. Real-time rendering rates of more than 10 frames per second have been successfully achieved even for the 3D data sets with the highest resolution by running our fast rendering algorithm on an SGI Power Challenge with 16 processors. An interactive user-interfaces have been developed for users to conveniently exploring the 3D data sets from different views both on SGI workstations and a virtual reality environment with a workbench.

References

[1] S. Bryson and S. Johan(1996). "Time Management, Simultaneity and Time Critical Computation in Interactive Unsteady Visualization Environments". *Proc. IEEE Visualization '96*, 255-261.

[2] J. Nieh and M. Levoy (1992). "Volume Rendering on Scalable Shared-Memory MIMD Architectures". *Proc. 1992 Workshop on Volume Visualization*, Boston, MA, October 1992, 17-24.

[3] C. Silva and A. Kaufman (1994). "Parallel Performance Measures for Volume Ray Casting". *Proc. IEEE Visualization '94*, 196-203.

[4] P. Lacroute (1995). "Real-Time Volume Rendering on Shared Memory Multiprocessors Using the Shear-Warp Factorization". *Proc. 1995 Parallel Rendering Symposium*, Atlanta, GA, 1995, 15-22.

[5] M. Levoy (1990). "Volume Rendering by Adaptive Refinement". *The Visual Computer*, Vol. 6, No. 1, February 1990, 2-7.

[6] J. Danskin and P. Hanrahan (1992). "Fast Algorithms for Volume Ray Tracing". *Proc. 1992 Workshop on Volume Visualization*, Boston, MA, October 1992, 91-105.

[7] S. You, L. Hong, M. Wan, K. Junyaprasert, A. Kaufman, S. Muraki, Y. Zhou, M. Wax, and Z. Liang (1997). "Interactive Volume Rendering for Virtual Colonoscopy". *Proc. IEEE Visualization '97*, Phoenix, AZ, October 1997, 433-436.

[8] M. Levoy (1990). "Efficient Ray Tracing of Volume Data". *ACM Transactions on Graphics*, Vol. 9, No. 3, July 1990, 245-261.

[9] R. Avila, L. Sobierajski, and A. Kaufman (1992). "Towards a Comprehensive Volume Visualization System". *Proc. IEEE Visualization '92*, Boston, MA, October 1992, 13-20.

[10] M. Wan, S. Bryson, and A. Kaufman. "Boundary Cell-Based Acceleration for Volume Ray Casting". (submitted)

[11] M. Wan, S. Bryson, and A. Kaufman. "Optimal Interpolation for Volume Ray Casting". (submitted)

[12] M. Wan, A. Kaufman, and S. Bryson. "High Performance Presence-Accelerated Ray Casting". (submitted)

Inventions: There were no inventions resulting from this project.

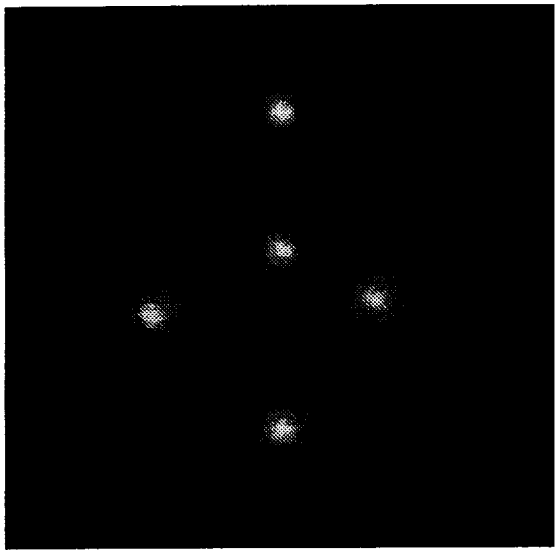


Figure 1: *An electric field simulation.*

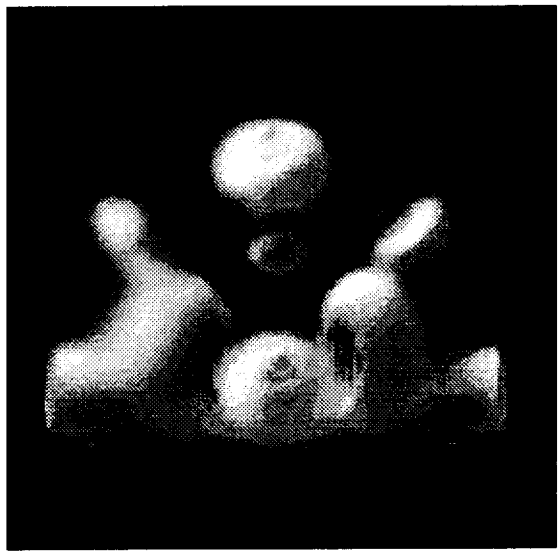
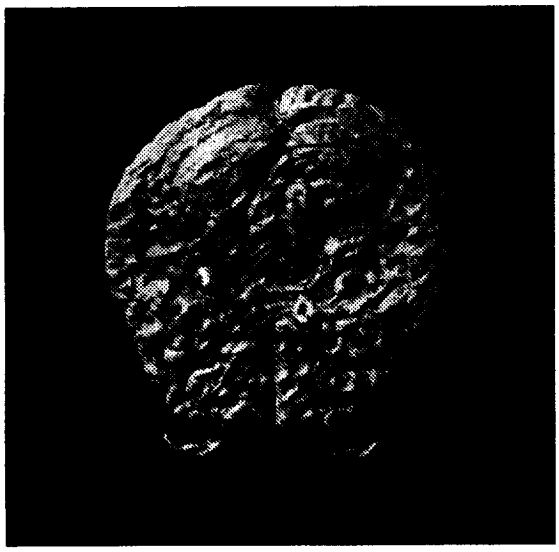
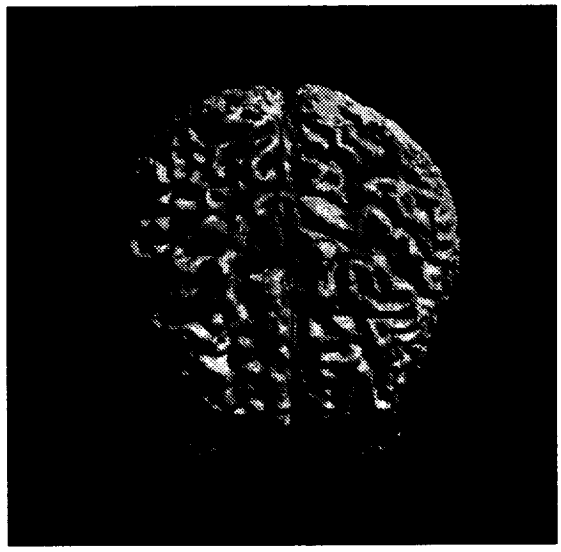


Figure 2: *A negative potential of iron protein.*



(a) front view



(b) rear view

Figure 3: *A human brain.*