Accelerating Volume Reconstruction with 3D Texture Hardware

Timothy J. Cullip[1] and Ulrich Neumann[2]

1 - Radiation Oncology Department
University of North Carolina at Chapel Hill

2 - Department of Computer Science
University of North Carolina at Chapel Hill

Abstract

This paper describes a new method of rendering volumes that leverages the 3D texturing hardware in Silicon Graphics RealityEngine™ workstations. The method defines the volume data as a 3D texture and utilizes the parallel texturing hardware to perform reconstruction and resampling on polygons embedded in the texture. The resampled data on each polygon is transformed into color and opacity values and composited into the frame buffer. A 128×128×64 volume is rendered into a 512^2 window at over 10 frames per-second. Two alternative strategies for embedding the resampling polygons are described and their trade-offs are discussed. This method is easy to implement and we apply it to the production of digitally reconstructed radiographs as well as opacity-based volume rendered images. The generality of this approach is demonstrated by describing its application to the proposed PixelFlow graphics system. PixelFlow overcomes the lighting and volume size limitations imposed by the RealityEngine. It is expected to render 256^3 data sets on a 640×512 screen at over 10 frames per second

1. Introduction

Recently, rendering volume data at truly interactive frame rates has become possible with commercially available graphics systems. Previously, the computing requirements for reconstructing and resampling volumes of data have been too great. With the advent of the Silicon Graphics RealityEngine™ and its 3D texturing hardware [SGI], the rendering method described here may be used to achieve over 10 frames per-second for a one-million point volume size rendered into a 512^2 window. Other data or image sizes are rendered at inversely-proportional frame rates.

2. Reconstruction

Viewing a volume from arbitrary positions requires reconstructing and resampling the volume along rays that extend from the view-point through the image plane pixels. If the resampled points are constrained to lie in planes, a textured polygon can be used to resample volume data that is loaded into the texture memory of the RealityEngine. Polygons are embedded in the texture which is resampled at each point on the polygon projecting to a pixel. There are two alternatives for selecting the orientations of the resampling polygons. Figure 1a illustrates resampling on

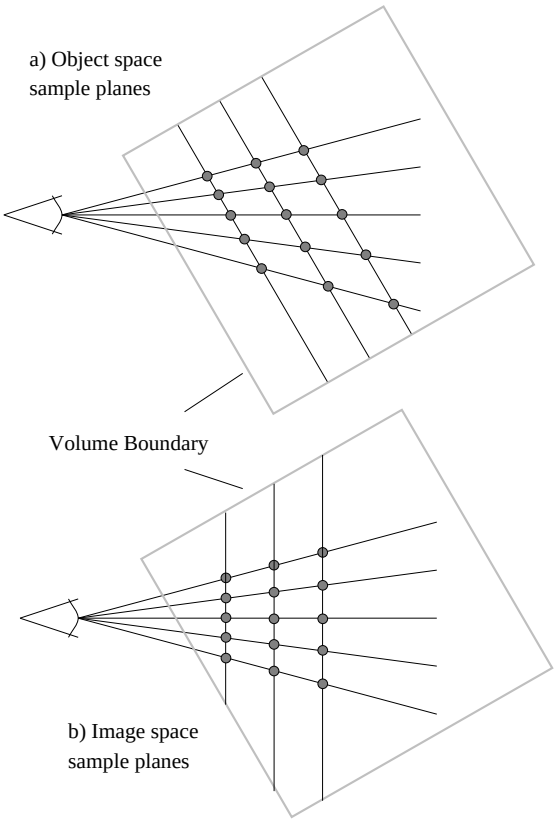

Fig. 1 - Resampling polygon orientations

RealityEngine is a trademark of Silicon Graphics Inc.

polygons aligned with the object-space axes. Figure 1b shows resampling on polygons aligned with the image-space axes. In either case, the resampled values behind each pixel are combined to produce a color for that pixel. The combining method is often a compositing operation, but may be other operations as required by the visualization application.

Polygons aligned in object-space are defined to lie within the volume and rendered with GL library calls. This method is complicated slightly by the need to reorient the sampling polygons in the plane most parallel to the view-plane as the view-point changes. This is accomplished by examining the view matrix and explicitly creating polygons for the six cases that arise [Westover91].

Polygons aligned in image-space must be clipped to the boundaries of the volume to ensure valid texture coordinates. Polygons are defined in image-space and transformed by the inverse viewing matrix into object-space where the clipping occurs. Clipped polygons are then rendered with the usual GL library calls.

In addition to using unclipped polygons, there are other advantages to using the object-space method. The texturing time for any polygon is proportional to the number of pixels it covers. A priori information about the extent of interesting features in each slice of the volume may be used to minimize the polygon size, and thus its texturing time, as a function of its location.

The texture memory of the RealityEngine is limited to 1M 12-bit data points. To render larger volumes, slab subsets are loaded and rendered in succession. Texture memory may be reloaded in about 0.1 seconds. With the object-space method, rendering each slab is simple. The image-space method must render polygons multiple times, clipping them to the currently loaded volume slab.

3. Radiographs

A digitally reconstructed radiograph of medical volume data is produced by combining the resampled values behind each pixel to approximate the attenuation integral

$$\text{pixel intensity} = 1.0 - \exp(-\sum u_i \, d) \qquad (1)$$

where u_i are the resample values behind a pixel and d is the spacing between sample values. Note that d is constant for all samples behind a pixel, but due to perspective, it varies from pixel to pixel. The resampled u_i terms are summed at each pixel, and the d factors are applied by using an additional full-screen polygon with a 2D texture corresponding to the d values required for each pixel. The summation results may be viewed directly or the exponential required to mimic a radiograph may be computed at each pixel by using a lookup table.

The RealityEngine has a maximum precision of 12-bits per frame buffer and texture component. The summation could easily overflow that unless the sample values are properly scaled. Our implementation maintains 12-bit volume data values in the texture memory and scales each resampled value by a user controlled "exposure" value ranging from zero to one. The scaled samples are then summed and clamped if they exceed the 12-bit range. In practice, it has been easy to find suitable exposure control settings for the data sets tested. Figure 2 shows radiograph images of 128×128×64 CT data of a human pelvis made with polygons aligned in object-space.

Fig. 3 - Digitally reconstructed radiographs

4. Opacity-Based Rendering

The summation of samples produces radiograph images. Compositing samples produces images with occlusion. Only one texture component is required for the linear attenuation coefficient used to produce radiographs. Two 8-bit texture components can represent the raw data and a precomputed shading coefficient. The resampled data component values are used as indices into an opacity lookup table. This lookup uses the texture hardware for speed. The shading coefficient is a function of the original data gradient and multiplies the sample opacity to produce images of shaded features as shown in figure 3. This figure shows the human pelvis data set above an image of a 4 mm^3 volume of a chicken embryo acquired by a microscopic MRI scanner. The precomputed shading fixes the light position(s) relative to the volume. For more general lighting by sources fixed in image-space, the shade texture component must be replaced by three components containing the normalized data gradient. Unfortunately, the resampled gradient on the polygons is not normalized and normalization is an expensive process requiring a square root. Lighting without normalization is possible, but this has not yet been tried to see how serious the artifacts are.

Fig. 3 - Shaded volume rendering

5. Performance

We consider two data sizes rendered into a 512^2 window. The smaller data size of 128×128×64 may be rendered at ten frames per-second using 128 polygons aligned in object-space. This equates to a processing rate of 10 million voxels per-second. In our test images we measured about 160 million pixel operations per second, where each pixel operation is a trilinear interpolation of the 3D texture components, a multiplication by a scaling or opacity factor, and a summation or composite into the frame buffer. The larger data size of 256×256×64 requires four 256×256×16 texture slabs and is rendered at 2.5 frames per-second with 256 resampling polygons. Loading texture slabs consumes less than 0.1 seconds per-slab. A progressive refinement approach would allow a user to manipulate the low-resolution data at a high frame rate, and render the high-resolution data as soon as the user allows the motion to stop. The performance is very linear with respect to the number of pixels processed. As the number of screen pixels or resampling polygons is doubled, the frame rate is halved. If more resampling polygons are used, higher quality images are obtained at the expense of lower rendering speed.

6. PixelFlow

Texturing hardware is likely to be a common feature of graphics systems in the future. The PixelFlow graphics system under development at the University of North Carolina at Chapel Hill will have texturing hardware [Molnar+92] that it is suitable for a variant of the polygon resampling approach described above for the RealityEngine. We propose a polygon texturing approach for the PixelFlow system that will overcome the limitations on realistic lighting and data size imposed by the RealityEngine. The texturing hardware in PixelFlow will allow 128^2 pixel processors to access eight arbitrarily-addressed 32-bit values in texture memory in under 500 μs. PixelFlow texturing hardware does not perform any operations on these texture values; rather, they are simply loaded into the pixel processors where a user's program manipulates them as ordinary data. If the 32-bit values are treated as four 8-bit texture components, then three may be

used for the x, y, and z gradient components and one for the raw data. Trilinear interpolation of all four components is performed by software executing concurrently on all 128^2 pixel processors. The interpolated gradient is normalized and used to compute Phong shaded lighting coefficients. Piecewise-linear transformations are computed that translate the interpolated data value into color and opacity. The color is multiplied by the shading coefficient and composited into the frame buffer. Each of the 128^2 pixel processors has an 8-bit ALU, hardware multiply, and hardware-assisted square root and divide instructions. These assets permit the shading and classification to be performed concurrently with the next polygon texture lookups. PixelFlow systems may contain many texture engines whose results are composited over a very high speed network. Unlike the RealityEngine, each PixelFlow processor card has its own memory capable of holding 0.5 M texture values, so the size of the volume that a system can hold may be expanded by adding additional processor cards. A system with 43 processor cards will handle a 256^3 volume size and render it into a 640×512 window using 400 object-space resampling polygons at over 10 frames-per second. As in the case of the RealityEngine, performance is inversely proportional to the number of screen pixels and resampling polygons used.

7. Summary

We have presented a volume rendering approach that leverages the 3D texturing hardware available on high-performance graphics systems. The texturing hardware accelerates the reconstruction and resampling process inherent in all volume visualization algorithms. A working implementation of this approach for the Silicon Graphics RealityEngine is described, and radiographs and opacity-based volume visualizations are demonstrated. This approach is easily implemented and produces the highest performance rendering rates ever reported for a commercially available system. The application of this approach to the proposed PixelFlow graphics system is also presented. PixelFlow will provide scalable volume rendering performance and data capacity beyond that of any systems currently available.

8. References

[Molnar+92] Steven Molnar, John Eyles, and John Poulton. "PixelFlow: High-Speed Rendering Using Image Composition." *Computer Graphics* 26(2):231-240, July 1992. Proceedings of SIGGRAPH'92.

[SGI] Greg Estes and Joshua Mogal. "Its' Time to Get Real." *Iris Universe* (21):15-19, Summer 1992. Silicon Graphics Inc.

[Westover91] Lee Westover. "Splatting - A Parallel, Feed-Forward Volume Rendering Algorithm." *Dept. of Computer Science, UNC at Chapel Hill*, Tech Report TR91-029, July 1991. Ph.D. Dissertation.