Gaze-Directed Volume Rendering

TR89-048

December, 1989

Marc Levoy, Ross Whitaker

The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175

To appear in Proceedings of the 1990 Utah Symposium on Interactive 3D Graphics.

Gaze-Directed Volume Rendering

Marc Levoy
Ross Whitaker

Computer Science Department
University of North Carolina
Chapel Hill, NC 27599

Abstract

We direct our gaze at an object by rotating our eyes or head until the object's projection falls on the fovea, a small region of enhanced spatial acuity near the center of the retina. In this paper, we explore methods for encorporating gaze direction into rendering algorithms. This approach permits generation of images exhibiting continuously varying resolution, and allows these images to be displayed on conventional television monitors. Specifically, we describe a ray tracer for volume data in which the number of rays cast per unit area on the image plane and the number of samples drawn per unit length along each ray are functions of local retinal acuity. We also describe an implementation using 2D and 3D mip maps, an eye tracker, and the Pixel-Planes 5 massively parallel raster display system. Pending completion of Pixel-Planes 5 in the spring of 1990, we have written a simulator on a Stellar graphics supercomputer. Preliminary results indicate that while users are aware of the variable-resolution structure of the image, the high-resolution sweet spot follows their gaze well and promises to be useful in practice.

CR categories and subject descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generation - *display algorithms*; I.3.6 [**Computer Graphics**]: Methodology and Techniques - *Interaction techniques*; E.1 [**Data structures**]: Trees

General terms: Algorithms, Human Factors, Performance

Additional Key Words and Phrases: Volume rendering, ray tracing, eye tracking, head-mounted display

1. Introduction

The spatial acuity of the human eye varies across the surface of the retina. It is highest in the fovea centralis, a region occupying roughly 4 degrees of visual arc, and falls off gradually toward the periphery of the visual field [22]. Directing one's gaze at an object consists of rotating either the eye within its socket or the entire head until the object's retinal projection falls on the fovea.

Researchers in the flight simulator industry have constructed a number of proprietary real-time image generation systems that take advantage of this variation in retinal acuity to reduce rendering costs [20, 7]. These systems track the gaze direction of one or both eyes, generate a high-resolution inset image or sweet spot corresponding to the detected direction, and superimpose it using a servo-controlled mirror over the appropriate portion of a low-resolution background image. Electronic blending of the two images is employed to soften the visual impact of the transition between them. If the inset image is large enough and is moved quickly enough in response to changes in gaze direction, the illusion of a full-field high-resolution image is obtained [19].

This paper explores methods for encorporating gaze direction directly into rendering algorithms. This approach has two advantages over analog superimposition of a separately generated inset image:

- An image of continuously varying resolution can be generated, more closely approximating the falloff in retinal acuity.
- The image can be displayed on a conventional television monitor, obviating the need for specialized display devices.

The algorithm we describe is a spatially adaptive ray tracer for volume data. Previous volume rendering techniques include the slice-by-slice method of Drebin et al. [5], the cell-by-cell method of Upson and Keeler [21], the voxel-by-voxel method of Westover [23], and the ray tracing methods of Levoy [12], Sabella [18], and Upson and Keeler [21]. Ray tracers that modulate the number of rays cast per unit area on the image plane have been reported by Whitted [24], Lee et al. [11], Dippé and Wold [4], Cook [2], and Kajiya [9] for geometrically defined scenes and Levoy [14] for volume data. The modulation criteria in each case is local image complexity. In the present algorithm, we modulate both the number of rays and the number of samples per ray, and the modulation criteria is local retinal acuity.

Spatially adaptive ray tracers yield a low-density nonuniform distribution of samples across the image plane. Methods for reconstructing images from such sampling patterns include mip maps [25], summed-area tables [3], multi-stage filtering [16], and integration over a tiling of rectangular cells [17]. We use a method based on 2D mip maps and their extension into three dimensions - 3D mip maps.

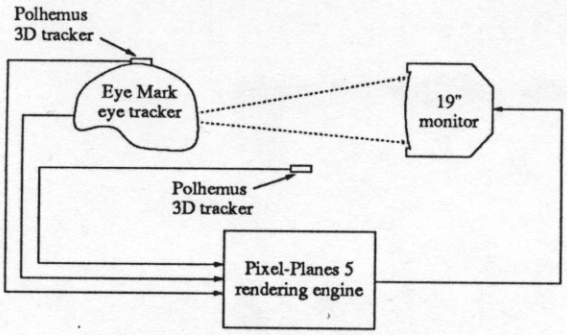

Figure 1: Hardware configuration

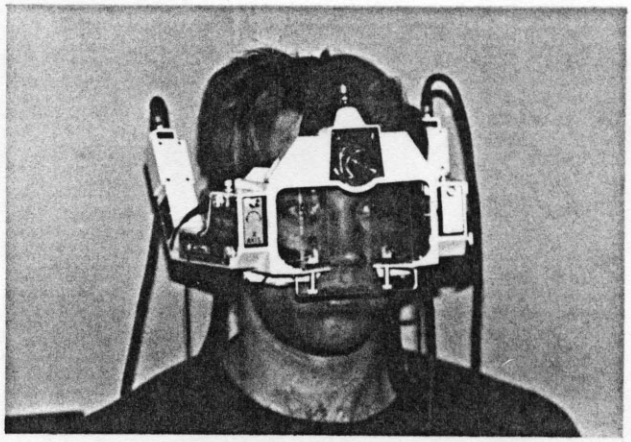

Figure 2: Eye tracker

The goal of this research is to provide users of a planned real-time volume rendering workstation with the illusion of a full-screen high-resolution image at reduced computational cost. Preliminary estimates suggest that for the proposed workstation, tracking gaze direction may reduce image generation time by a factor of up to 5.

2. Hardware configuration

Figure 1 summarizes the proposed hardware configuration. It consists of an NAC Eye Mark eye tracker, two Polhemus 3SPACE trackers, the Pixel-Planes 5 rendering engine, and a conventional 19" television monitor.

The NAC Eye Mark eye tracker is a see-through helmet in which two infrared light emitting diodes have been mounted. Reflections of the two infrared spots from the iris of each eye are tracked in real-time by solid state cameras mounted on the side of the helmet as shown in figure 2. If the helmet is firmly attached to the user's head, this device measures gaze angle relative to the helmet and is accurate to within 3 degrees of visual arc. The position and orientation of the helmet relative to the television monitor is given by mounting one of the Polhemus trackers on the helmet. Combining the information returned by the eye tracker and the Polhemus gives the X and Y coordinates of the image pixel currently centered on the user's fovea. The second Polhemus is held in the user's hand and used to control position and orientation of a volumetrically defined object, a cutting plane, or a light source.

Pixel-Planes 5 is a massively parallel raster display system currently under development at the University of North Carolina [8] and scheduled for completion in the spring of 1990. It consists of 16 independently programmable 40-MFLOP graphics processors, 1/4 million pixel processors organized into 16 independently programmable renderers, a 1024 x 1280 pixel color frame buffer, and a 640 Mb/sec ring network. The implementation on this machine of a near real-time ray tracer for volume data has already been described [13]. The shading calculations for all voxels are performed in the pixel processors, and the ray tracing required to generate an image is divided among the graphics processors. In the present configuration, the combined input of the eye tracker and the two Polhemus trackers is used by Pixel-Planes 5 to generate a variable-resolution volume rendered image for display on the television monitor.

3. Variable-resolution volume rendering

The volume rendering method used in this paper is based on [12]. We begin with a 3D array of voxel data. The array is classified and shaded to yield a color and an opacity for each voxel. Viewing rays are then traced into the array from an observer position. For each ray, samples are drawn along the ray, and a color and opacity is computed at each sample position by trilinearly interpolating from the colors and opacities of the nearest eight voxels. The resampled colors and opacities are then composited from front to back to yield a color for the ray.

To generate a variable-resolution image for a given gaze direction, we modulate both the number of rays cast per unit area on the image plane and the number of samples drawn per unit length along each ray as functions of local retinal acuity. Since less than one ray may be cast per pixel in the visual periphery, care must be taken to avoid undersampling artifacts. We associate with each pixel a 2D convolution mask whose nonzero extent varies as a function of distance on the image plane from the pixel center to the gaze direction as shown in figure 3. A fixed number of rays is cast from each mask and the spacing between samples along a ray is made proportional to the size of the mask. A color is computed for the pixel by integrating the colors returned by all rays cast in the mask weighted by a 2D filter function. A discussion of suitable filter functions is contained in [6].

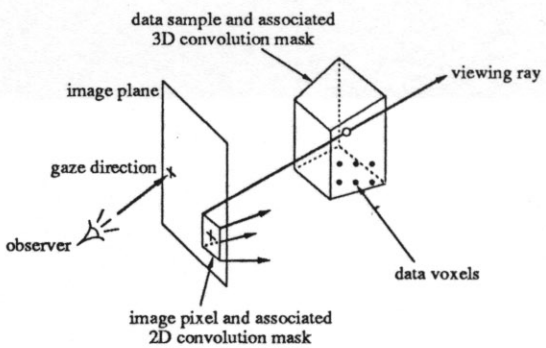

Figure 3: 2D and 3D convolution masks

Since the density of rays and hence of samples along rays decreases as one moves away from the gaze direction, care must also be taken to avoid undersampling the 3D data. Extending the technique described above, we associate with each sample along a ray a 3D convolution mask (see figure 3) whose non-zero extent is proportional to the spacing between samples. A color and opacity is computed for the sample by integrating the colors and opacities of all voxels falling inside the mask weighted by a 3D filter function.

4. Implementation using mip maps

An analysis of numerical error in above algorithm suggests that the size and placement of 2D convolution masks can be quantized to multiples of the pixel spacing without visibly degrading the image. This allows us to share rays among adjacent pixels, substantially reducing the number of rays that must be cast to generate an image. To realize these savings, we define a pyramid of 2D texture maps whose resolutions are binary fractions of the image resolution. Each texture map contains one-fourth as many pixels as the map beneath it in the pyramid. For each pixel, rays are cast from the four corners of each of two convolution masks that enclose the pixel and whose quantized sizes fall just above and just below the desired mask size. As rays are cast, their colors are stored in the pyramid. A boolean flag array is used to insure that rays are cast only once. A single color is then computed for the pixel by bilinearly interpolating between the colors returned by the four rays cast in each mask and linearly interpolating between the two resulting values. In essence, the tracing of rays generates a partially populated 2D mip map [25] from which a variable-resolution image is generated by setting the pyramid's vertical coordinate for each pixel proportional to retinal acuity.

A second observation on the original algorithm is that the large 3D convolution masks required to draw samples of the 3D data in the visual periphery threatens to destroy the computational advantage of employing a lower sampling rate in these areas. To overcome this difficulty, we employ a extension to three dimensions of the quantization technique described above. Specifically, we precompute a hyperpyramid of 3D texture volumes whose resolutions are binary fractions of the data resolution. Each texture volume contains one-eighth as many voxels as the volume beneath it in the hyperpyramid. By placing only viewpoint-independent shading components in this data structure, the cost of computing it is amortized over the duration of an animation sequence. A sample is drawn from the hyperpyramid by selecting the two volumes whose resolutions fall just above and just below the resolution corresponding to the desired 3D convolution mask size, trilinearly interpolating between the nearest eight voxels in each volume, and linearly interpolating between the two resulting values. The viewpoint-dependent portion of the shading calculations are then applied to yield a color and opacity which are composited into the ray. In essence, the hyperpyramid is a 3D mip map that is resampled using an extension to three dimensions of the method described by Williams for 2D mip maps.

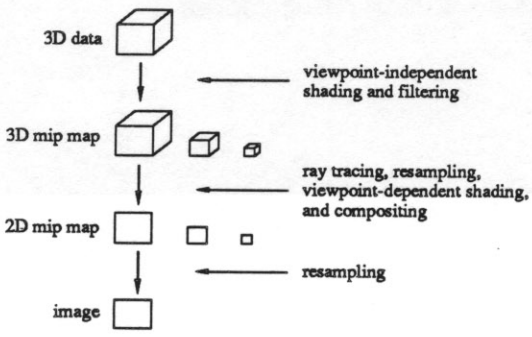

Figure 4: Rendering pipeline

Figure 4 summarizes the rendering pipeline. It begins with a 3D scalar or vector-valued array. In a preprocessing step, viewpoint-independent shading calculations are performed to yield a vector-valued volume of shading components. This volume forms the base of a 3D mip map. Repeated filtering and resampling is applied to the volume, producing successively lower resolution volumes to fill the mip map. For each frame, gaze-directed ray tracing, resampling, viewpoint-dependent shading, and compositing are performed to yield a 2D mip map. This data structure is then resampled to generate an image at the display resolution.

The processing required at each pixel is given by the following pseudocode:

```
procedure RenderPixel(x,y) begin
    {Loop through nearest two 2D mip map levels}
    m_lores = ⌊2DLevel(x,y)⌋, m_hires = ⌈2DLevel(x,y)⌉;
    for m = {m_lores,m_hires} do begin
        {Cast rays from four corners of mask}
        for i = {0,1} do begin
            for j = {0,1} do begin
                if not F_{x/2^m+i,y/2^m+j,m} then begin
                    c_{x/2^m+i,y/2^m+j,m} = TraceRay(x/2^m+i,y/2^m+j);
                    F_{x/2^m+i,y/2^m+j,m} = true;
                end
            end
        end
        {Bilirp to obtain one color for mask}
        c_m = Bilirp(x,y,m);
    end
    {Lirp between resulting values}
    c_pix = Lirp(c_{m_lores},c_{m_hires},2DLevel(x,y) mod 1);
    return (c_pix);
end RenderPixel.
```

```
procedure TraceRay(x,y) begin
    c_ray = 0, α_ray = 0;
    {Loop through all samples along ray}
    for z from Near to Far by 3DLevel(x,y,z) do begin
        {Loop through nearest two 3D mip map levels}
        n_lores = ⌊3DLevel(x,y,z)⌋, n_hires = ⌈3DLevel(x,y,z)⌉;
        for n = {n_lores,n_hires} do begin
            {Trilirp to obtain one value for level}
            S_n = Trilirp(x,y,z,n));
        end
        {Lirp between resulting values}
        S_vox = Lirp(S_n_lores,S_n_hires,3DLevel(x,y,z) mod 1);
        {Perform viewpoint-dependent shading}
        c_vox,α_vox = Shade(S_vox);
        {Composite into ray}
        Composite(c_vox,α_vox,c_ray,α_ray);
    end
    return (c_ray);
end TraceRay.
```

In this pseudocode, c denotes a scalar or vector color, α denotes an opacity, and S denotes a vector of shading components. The 2DLevel procedure accepts a pixel location, determines the distance from the pixel to the gaze direction by referencing a 2D lookup table indexed by X and Y offset, and returns a floating point 2D mip map vertical coordinate by referencing a 1D lookup table indexed by distance. The 3DLevel procedure accepts a voxel location, computes the volume in voxels of the 3D convolution mask whose non-zero extent is proportional to the spacing between samples, and returns a floating point 3D mip map vertical coordinate by referencing a 1D lookup table indexed by mask volume. For parallel projections, the spacing between samples along a ray is a constant; for perspective projections, it rises with increasing distance from the observer. The Bilirp and Trilirp procedures accept pixel and voxel coordinates respectively and an integer 2D or 3D mip map vertical coordinate and return a scalar or vector interpolated from the appropriate texture map or volume. The Lirp procedure interpolates between two scalars or vectors based on a floating point interpolant lying between zero and unity. The Shade procedure accepts a vector of voxel shading components and performs viewpoint-depending shading to yield a color and an opacity for the voxel. The Composite procedure composites a voxel color and opacity into the color and opacity accumulated along a ray. Further details on compositing and shading calculations are given in [12].

On Pixel-Planes 5, viewpoint-independent shading and filtering will be performed on the 16 graphics processors (GP's) at the start of an animation sequence. The resulting 3D mip map will be transferred across the ring network and distributed among the 1/4 million pixel processors (PP's). For each frame, viewpoint-dependent shading will be performed in parallel by the PP's on the entire mip map, followed by gaze-directed ray tracing, resampling, and compositing on the GP's. The resulting 2D mip map will then be resampled in sections by the GP's and transmitted to the frame buffer for display.

5. Discussion

Shading issues. The values stored in the 3D mip map depend on what shading model is employed and what rendering parameters change from frame to frame. If a Lambertian (diffuse) shading model is used and objects and light sources are fixed and only the observer moves, then the entire shading calculation is viewpoint-independent. In this case, final voxel color and opacity may be computed and stored in the mip map. If specular reflection is included in the shading model, then a viewpoint-dependent component must be computed on every frame and added to the values obtained from the mip map. If the lighting and observer are fixed and the object moves, then diffuse and specular components must be evaluated on every frame. In this case, only normalized voxel gradients and voxel reflectance coefficients can be stored in the mip map. A generalization of the distinction between viewpoint-independent and viewpoint-dependent components for shading of volume data is given by the texel model of Kajiya [10].

Sampling issues. Since the 3D mip map is intended to be independent of observer position, an isotropic convolution filter is used during its construction. For perspective projections, the convolution filter required at sample positions along a ray are generally nonisotropic as shown in figure 3. This mismatch introduces errors into the resampled values. The errors are minor for typical projections, and less severe than errors arising when surfaces textured using a 2D mip map are turned nearly on edge [25]. Since shading is a non-linear process, calculating colors from blurred normals stored in a 3D mip map is not equivalent to calculating colors from high-resolution normals and subsequently blurring them for storage in the mip map. This introduces additional errors into the computed colors. These errors are not visually objectionable, however, as noted by Blinn [1] for the case of bump mapping.

Rendering efficiency. Using a 3D mip map to represent volume data, the cost of drawing a sample is independent of the distance between samples. Using a 2D mip map to represent ray colors, the cost of computing a pixel color is independent of the distance between rays. Rendering cost per unit area on the image plane is therefore linearly related to the density of rays and samples and independent of the data resolution. If one of the shading components stored in the 3D mip map is voxel opacity, and if the non-zero extent of the convolution filter used during construction of the mip map measures 2 voxels on a side (i.e. if each voxel contains contributions made by exactly eight voxels from the volume beneath it in the hyperpyramid), the mip map can also be used as a hierarchical spatial occupancy enumeration of the data - an octree. Each voxel tells us whether a particular region of space is occupied or empty. By descending the mip map from top to bottom for each ray, occupied leaf voxels can be found in approximately logarithmic time relative to the length of the ray. This technique substantially reduces rendering time for many useful datasets [15]. In summary, *a 3D mip map provides an efficient solution to both the visibility problem and the resampling problem for volume data!*

6. Simulation results

Pending the completion of Pixel-Planes 5, we have implemented our rendering algorithm on a Dec 3100 workstation. The figures in this paper were generated from a 256 × 256 × 109 voxel magnetic resonance (MR) scan of a live human subject. Using a diffuse shading model and a 2 × 2 × 2 box

filter, we constructed a 3D mip map containing voxel color and opacity at varying resolutions. This preprocessing step required 5 minutes.

We then cast rays into the 3D mip map using a parallel viewing projection and a sweet spot whose structure is shown in figure 5. We assume a 19" television monitor (measured diagonally) viewed from a distance of 20". A horizontal line through the middle of the displayed image subtends 37° of visual arc. Our target resolution (shown as a bell-shaped curve in figure 5b) falls off smoothly (a Gaussian was used) from one ray per pixel (corresponding to one 3D sample per voxel) inside a circle 4.2" in diameter (12° of arc) to one ray per 16 pixels (one 3D sample per 64 voxels) outside a circle 7" in diameter (20° of arc). (For comparison, the high-resolution area-of-interest inset image employed in the CAE-Link system has a diameter of 18° including a blending annulus 3° in radius [7].)

Our quantized implementation approximates the target resolution by casting one ray per pixel inside a circle roughly 5" in diameter (span [1] in figure 5b), one ray per 16 pixels outside a circle 5" in diameter (spans [3]), and one ray per 4 pixels within an annulus having an inner diameter of 4.2" and an outer diameter of 7" (spans [2]). The resulting partially populated 2D mip map is shown in figure 6.

Finally, we interpolate between the images in the 2D mip map based on the target resolution at each pixel as described earlier, producing the variable-resolution image shown in figure 7. For comparison, a full-resolution image is shown in figure 8.

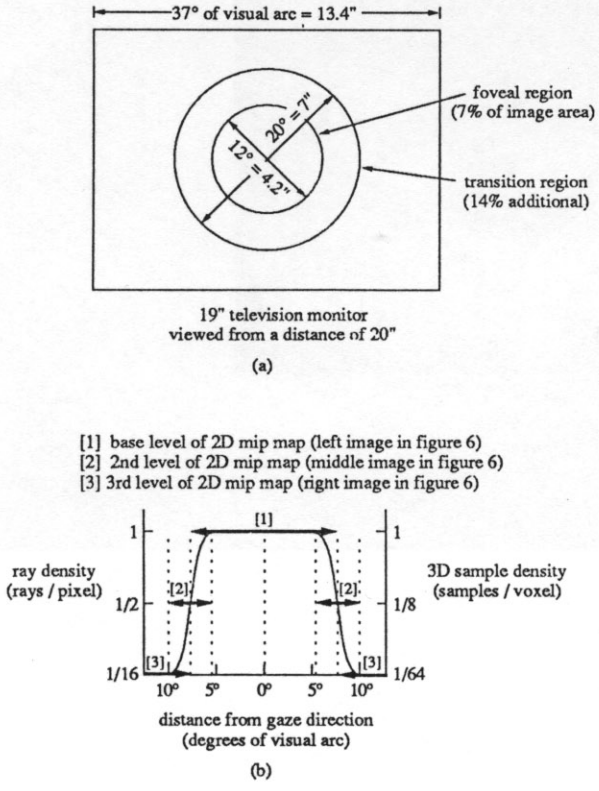

Figure 5: Structure of sweet spot

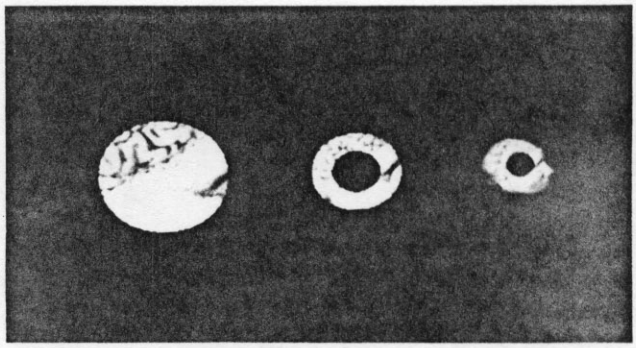

Figure 6: Partially populated 2D mip map

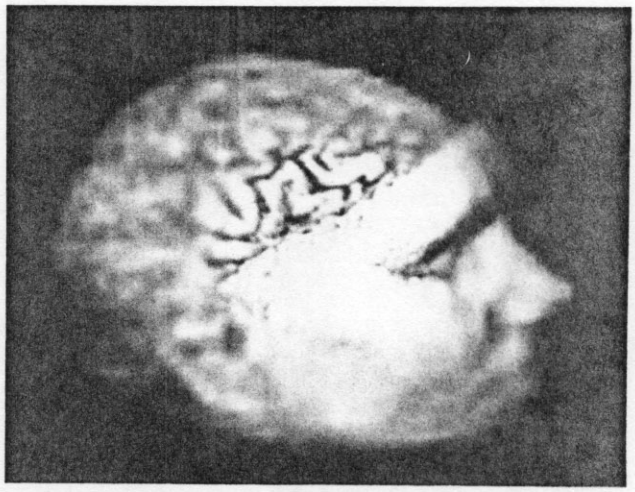

Figure 7: Variable-resolution image

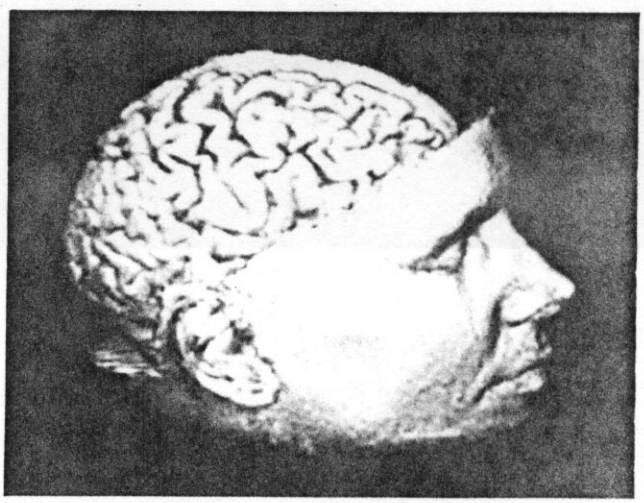

Figure 8: Full-resolution image

	fig.	# of rays	# of 3D samples	rendering time
variable-resolution	7	9,657	55,438	13.0 secs
full-resolution	8	44,100	316,500	59.6 secs

Table I: Rendering performance

Table I compares rendering performance for the two images. Timings include all per-frame processing. The opacity component of the 3D mip map was used as an octree to speed up ray tracing of both images. As the table shows, the variable-resolution image required roughly 1/4 as many rays, 1/6 as many voxels (this would be 1/8 for data of uniform complexity), and 1/5 as much rendering time as the full-resolution image.

To obtain an early evaluation of the performance of our eye tracker and the behavioral response of users to our variable-resolution imagery, we have written a simulator on a Stellar GS-1000 graphics supercomputer. In the simulator, we precompute a fully populated 2D mip map for one view of a volume dataset. The Stellar is fast enough to read gaze direction from the eye tracker and generate variable-resolution images from the precomputed mip map at 15 frames per second. System latency is estimated at between 100 and 150 ms. To eliminate the need for tracking head motion in the simulator, users are mechanically constrained by a chin rest and immobilization strap. We have subjectively evaluated users in both tracking mode (up to 200°/sec) by asking the user to follow the motion of a cursor superimposed on the image and saccading mode (may exceed 700°/sec). Users report that the high-resolution sweet spot follows their gaze flawlessly in tracking mode and adequately although with a perceptible delay in saccading mode. Users are generally aware of the variable-resolution structure of the image.

Based on an expected speedup of 32:1 moving from a Dec 3100 to Pixel-Planes 5 (as reported in [13]) and working from a 128 × 128 × 109 voxel dataset, we expect to be capable of generating variable-resolution images slightly cruder than figure 7 at about 10 frames per second. Once Pixel-Planes 5 is operational, we intend to perform a series of formal psychophysical experiments to test the viability of our approach. Ideally, we could simply measure the user's ability to detect the presence of a low-resolution periphery in a forced-choice experiment in which variable-resolution sequences are randomly interspersed with full-resolution sequences. Realistically, we will probably be compelled to evaluate ease-of-use and utility for performing specific tasks such as feature recognition or image matching. We also intend to investigate the effect of varying the size and shape of the high-resolution sweet spot, the relative resolutions of the sweet spot and visual periphery, the structure of the blending region between them, and the filter functions employed at each stage of the rendering pipeline.

7. Conclusions

A hardware configuration and rendering algorithm have been presented for generating and displaying sequences of images whose resolution varies locally in response to changes in the user's direction of gaze. Encorporation of gaze information into the rendering algorithm allows images of continuously varying resolution to be generated, and produces images that can be displayed on conventional television monitors. Our rendering algorithm is a spatially adaptive ray tracer in which the number of rays and the number of samples per ray are modulated by local retinal acuity. For a 19" television monitor viewed at 20", a 7" high-resolution sweet spot, and a disparity in sample spacing between the spot and the surround of about 4:1 in each of X,Y, and Z, we obtain a cost savings of a factor of 5 over generating a full-screen high-resolution image.

The proposed system can be extended in a number of ways. If lag time proves problematic, we can employ predictive tracking. We expect such techniques to work well when the user is following an object rotated under joystick control due to the mechanical inertia of the joystick, but not as well when the user's gaze is wandering over a static image. By modifying the criteria for selecting 2D convolution mask sizes to encorporate measures of local image complexity as well as retinal acuity, images of subjectively equal quality can be generated up to an order of magnitude faster [14]. In the context of our Pixel-Planes implementation, this should allow us to render 256 × 256 × 128 voxel datasets at 30 frames per second. By not clearing the 2D mip map between frames when the object is stationary, progressive refinement can be supported. As the user's gaze wanders across the image, a trail of sweet spots is left behind. If the user fixates on one spot, the sweet spot grows in size until it encompasses the entire image. In the Pixel-Planes 5 implementation, progressive refinement should result in a full-screen high-resolution image in less than 1 second.

By adding a Z-component to each pixel in a 2D mip map, we create a variable-resolution Z-buffer. Preliminary analysis suggests that such a data structure could be used to implement gaze-directed polygon rendering. Polygon scanlines would be subdivided into segments corresponding to the boundaries of the populated portions of the various resolution Z-buffers required for a particular gaze direction. Each segment would then be scan-converted, Z-compared, and shaded. When all polygons have been processed, the partially populated mip map is converted into a variable-resolution image as described earlier. This approach would reduce the per-pixel costs of scan-conversion, hidden-surface removal, and shading.

In conclusion, we note that although our proposed approach permits only one user per television monitor, it is ideally suited for personal head-mounted displays. In that context, the eye tracker constitutes a unintrusive addition to the portable hardware and promises better rendering performance for a given image generation system. Before this goal can be realized, the resolution and angle of view of current head-mounted displays must be improved. Our approach may also prove useful for reducing image generation costs in non-gaze-directed environments by having the user attach a 3D cursor specifying an area-of-interest to some object in the scene.

Acknowledgements

The authors wish to thank Prof. Stephen M. Pizer of the Computer Science Department and Profs. R. Eugene Johnston and David Beard of the Radiology Department for their encouragement and support. Thanks are also due to Ned Greene of Apple Computer for an enlightening discussion concerning 3D mip maps. The MR scan used in this paper was provided by Siemens AG and edited by Dr. Julian Rosenman of the Radiation Oncology Department. This work was supported by NCI grant P01-CA47982.

References

[1] Blinn, J.F., "Light Reflection Functions for Simulation of Clouds and Dusty Surfaces," *Computer Graphics,* Vol. 16, No. 3, July, 1982, pp. 21-29.

[2] Cook, R.L., "Stochastic Sampling in Computer Graphics," *ACM Transactions on Graphics,* Vol. 5, No. 1, January, 1986, pp. 51-72.

[3] Crow, F.C., "Summed-Area Tables for Texture Mapping," *Computer Graphics,* Vol. 18, No. 3, July, 1984, pp. 207-212.

[4] Dippé, M.A.Z. and Wold, E.H., "Antialiasing Through Stochastic Sampling," *Computer Graphics,* Vol. 19, No. 3, July, 1985, pp. 69-78.

[5] Drebin, R.A., Carpenter, L., and Hanrahan, P., "Volume Rendering," *Computer Graphics,* Vol. 22, No. 4, August, 1988, pp. 65-74.

[6] Feibush, E., Levoy, M., and Cook, R., "Synthetic Texturing using Digital Filters," *Computer Graphics,* Vol. 14, No. 3, July, 1980, pp. 294-301.

[7] Fisher, R.A. and Tong, H.M., "A Full-Field-of-View Dome Visual Display for Tactical Combat Training," *Proc. Image Conference IV,* Phoenix, Arizona, June, 1987.

[8] Fuchs, H., Poulton, J., Eyles, J., Greer, T., Goldfeather, J., Ellsworth, D., Molnar, S., Turk, G., Tebbs, B., and Israel, L., "A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories," *Computer Graphics,* Vol. 23, No. 3, July, 1989, pp. 79-88.

[9] Kajiya, J.T., "The Rendering Equation," *Computer Graphics,* Vol. 20, No. 4, August, 1986, pp. 143-150.

[10] Kajiya, J.T., Kay, T.L., "Rendering Fur with Three Dimensional Textures," *Computer Graphics,* Vol. 23, No. 3, July, 1989, pp. 271-280.

[11] Lee, M.E., Redner, R.A., and Uselton, S.P., "Statistically Optimized Sampling for Distributed Ray Tracing," *Computer Graphics,* Vol. 19, No. 3, July, 1985, pp. 61-67.

[12] Levoy, M., "Display of Surfaces from Volume Data," *IEEE Computer Graphics and Applications,* Vol. 8, No. 3, May, 1988, pp. 29-37.

[13] Levoy, M., "Design for a Real-Time High-Quality Volume Rendering Workstation," *Proc. Chapel Hill Workshop on Volume Visualization,* ed. C. Upson, University of North Carolina, 1989, pp. 85-92.

[14] Levoy, M., "Volume Rendering by Adaptive Refinement," *The Visual Computer,* Vol. 6, No. 1, January, 1990. In press.

[15] Levoy, M., "Efficient Ray Tracing of Volume Data," *ACM Transactions on Graphics,* 1990. In press.

[16] Mitchell, D.P., "Generating Anti-Aliased Images at Low Sampling Densities," *Computer Graphics,* Vol. 21, No. 4, July, 1987, pp. 65-72.

[17] Painter, J. and Sloan, K., "Antialiased Ray Tracing by Adaptive Progressive Refinement," *Computer Graphics,* Vol. 23, No. 3, July, 1989, pp. 281-288.

[18] Sabella, P., "A Rendering Algorithm for Visualizing 3D Scalar Fields," *Computer Graphics,* Vol. 22, No. 4, August 1988, pp. 51-58.

[19] Peters, D., CAE-Link Corp., Personal communication, September, 1989.

[20] Tong, H.M. and Fisher, R.A., "Progress Report on an Eye-Slaved Area-of-Interest Visual Display," *Proc. Image Conference III,* Phoenix, Arizona, May, 1984.

[21] Upson, C. and Keeler, M., "VBUFFER: Visible Volume Rendering," *Computer Graphics,* Vol. 22, No. 4, August 1988, pp. 59-64.

[22] Uttal, W.R., *The Psychobiology of Sensory Coding,* Harper & Row, 1973.

[23] Westover, L., "Interactive Volume Rendering," *Chapel Hill Workshop on Volume Visualization,* Chapel Hill, North Carolina, May, 1989, pp. 9-16.

[24] Whitted, T., "An Improved Illumination Model for Shaded Display," *Communications of the ACM,* Vol. 23., No. 6, June, 1980, pp. 343-349.

[25] Williams, L., "Pyramidal Parametrics," *Computer Graphics,* Vol. 17, No. 3, July, 1983, pp. 1-11.