Hierarchically Accelerated Ray Casting for Volume Rendering with Controlled Error

Allen Van Gelder Kwansik Kim Jane Wilhelms
Baskin Center for Computer Engineering and Information Sciences
University of California, Santa Cruz 95064
UCSC-CRL-95-31
avg@cs.ucsc.edu ksk@cs.ucsc.edu wilhelms@cs.ucsc.edu

March 31, 1995

Abstract

Ray casting for volume rendering can be accelerated by taking large steps over regions where data need not be processed at a fine resolution. A new implementation is described that utilizes both a user-specified data importance and a high-level data model in appropriate regions to achieve acceleration. Previously reported work exploited high accumulated opacity, transparency, and nearly constant valued regions. This work generalizes homogeneity from nearly constant values to least-squares fits with a set of basis functions. The implemented set supports trilinear functions, and the framework supports other functions.

Experience is reported with sequential and small-scale parallel runs. Comparisons are made with other methods on the bases of time, image quality, and sensitivity to error tolerance. Ray casting with zero error provides a reference image with which others are compared quantitatively. The implementation lives within a general system for multi-dimensional trees. For a given error tolerance, this ensures that the same resolution is used in all regions, by all hierarchical rendering methods, whether based on projection or ray-casting. Data-importnace functions were found to be a significant source of acceleration.

Image comparisons led to identification of the main source of image degradation in projection methods as being color interpolation rather than data interpolation. An improvement to projection methods is suggested, based on hardware texture maps.
Keywords: computer graphics, scientific visualization, ray casting, direct volume rendering.

1 Introduction

This paper further explores visualization using an error-controlled hierarchy, as initially described earlier [WVG94]. The particular emphasis here is the addition of a ray-caster to the direct volume rendering methods already available in the program (coherent projection [WVG91] and hardware 3D texture maps [WVGW94]), and the comparison of the three methods.

The higher speeds of modern graphics workstations make ray-casting in 3D volumes practical for moderately sized volumes, although still not interactive in most cases. Various acceleration techniques can usually improve the speed substantially, as reviewed in Section 3. The implementation reported here provides several such accelerations.

The primary acceleration tool is a generalization of data homogeneity to regions that are adequately modeled by a set of "basis functions". Previous work (see Section 3) exploited nearly constant-valued regions; essentially the only basis function was the constant function. Other recent work uses wavelets as basis functions. Our ray-cast methodology is described in Section 4.

The crux of these approximation methods is the decision on whether the data is adequately approximated in the current region, or needs to be evaluated at finer resolution. Simple error metrics, such as unweighted RMS error, are not necessarily good indicators. Our implementation permits the user to supply a *data importance* weighting function. This allows small errors to be considered serious in the critical range of the data, while substantial errors are acceptable in uninteresting ranges.

The primary motivation of our work was not to build the world's fastest ray tracer, but rather to provide a reference image for quantitative image comparisons with faster methods. The program interface gives the user many degrees of control of the image parameters, and permits switching between rendering methods, and saving framebuffers, without introducing extraneous variations among images. Framebuffers can be compared quantitatively, pixel by pixel, by several metrics, including RMS error and maximum error. The hierarchical design is reviewed briefly in Section 2. Experimental results are reported in Section 7.

2 The Hierarchy

Our hierarchical visualization software *mdh* creates an n-dimensional hierarchy over an n-dimensional rectilinear volume. (In practice, n is usually three for spatial volumes or four for spatio-temporal volumes.) The hierarchy uses a space-saving strategy known as a BON tree [WVG92]. Each node in the tree contains a model of the data in the region covered by the node as well as evaluation information useful for traversing the tree for imaging.

The standard data model is a trilinear cell model (voxel and voxel trilinear models are also available). A sample data set of resolution (x, y, z) is assumed to represent corner points of $(x-1)*(y-1)*(z-1)$ cells. A trilinear function models the behavior of the sampled function, providing $C0$ continuity at shared faces in the original data. At interior nodes of the tree, the trilinear function represents a best fit to the trilinear functions of the cells covered by the node.

Evaluation information includes nodal error and importance. Nodal error is the average deviation of the data model stored with the node from the data within the region covered. Importance is a user-defined transfer function describing the importance of data values within the volume. Regions, such as the air around medical data, that do not provide useful information can be given small importance.

For rendering, the tree is traversed selectively based on the amount of error the user will allow in the image. For a quick image, large amounts of error may be tolerated. For a careful rendering, no error may be desirable. Even if no error is allowed, volumes with homogeneous regions can be rendered without descending through the entire tree to the cells. Therefore, the hierarchy and its selective traversal provide a fast and flexible method of imaging large volumes.

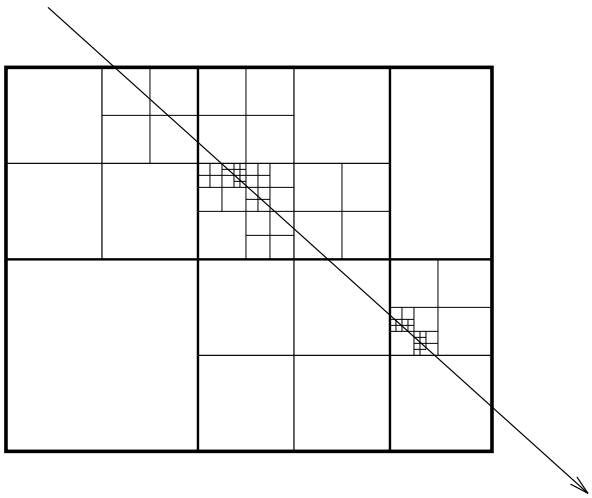


Figure 1: Raycasting through an *mdh* tree at varying levels.

This section is only a very brief summary of the methods developed in *mdh*. Previous publications describe the algorithms in more detail [WVG91, WVG92, WVG94, WVGW94].

3 Background and Related Work

Much work has been done on accelerating raycasting algorithms by exploiting empty or homogeneous subregions of volume data. Many other ray tracing and ray casting approaches have been reported, but this section reviews mainly those that rely on hierarchical representation.

Levoy [Lev90] introduced basic acceleration techniques based on the presence information and α-termination, in which the ray is no longer integrated when accumulated opacity is near 1.0. The volume data is preprocessed into an octree of bits, so an empty region can be traversed in one step. By casting a ray front to back, computing color integration can be terminated if it reaches the maximum opacity.

Laur and Hanrahan [LH91] extend this idea to take advantage of relatively constant-valued subregions to apply the hierarchical approach with back-to-front splatting. Depending on the error tolerance given, a large block of relatively constant subregions can be projected at the higher level of octree. Each node in the octree stores its average values and standard deviations (used as error estimates). The procedure performs progressive refinement until the given error level is satisfied.

Daskin and Hanrahan [DH92] extended the above work, using additional acceleration techniques for raycasting. They added β-acceleration, in which fewer samples are taken as the opacity along the ray accumulates.

Westermann [Wes94] provides a general multiresolution framework for volume rendering using a wavelet representation to sample data along the ray. Homogeneity and β-acceleration techniques are implemented on raycasting and wavelet coefficients are directly used to hierarchically sample data along the ray.

Wilhelms and Van Gelder extended the work of Laur and Hanrahan by incorporating additional basis functions, permitting the data to be fit by a trilinear function [WVG94]. Rendering was accomplished with projection of each region, giving more accurate images than splatting.

```
rayCastRegion(ray, region, entryPt, exitPt)
    {
    if (bottomLevel or withinTolerance)
        renderRay(ray, entryPt, exitPt)
    else
        subEntry = entryPt
        while (subEntry not = exitPt and not interrupted)
            {
            subRegion = findRegion(region, subEntry)
            subExit = findExit(ray, subRegion, subEntry)
            rayCastRegion(ray, subRegion, subEntry, subExit)
            subEntry = subExit
            }
    }
```

Figure 2: Overview of ray-casting within tree traversal. Tree node "`region`" is either bottom-level, or has children (typically 8) as found by `findRegion`.

This paper describes a method by which the ray is computed adaptively at different levels of the tree (see Figure 1), in the same vein as Danskin and Hanrahan, and also Westermann. However, the criteria for choosing the appropriate level differ substantially among the methods.

4 Ray-Casting Algorithm

The ray-casting method reported here was integrated into the multi-dimensional hierarchical system reported previously [WVG94]. This modular approach gave us numerous capabilities, essentially free of charge, such as depth selection based on error metrics and data importance, interrupt checking, choice of display dimensions on higher dimensional data. In particular, for a given error tolerance, the ray-caster will visit exactly the same cells as the previously implemented projection methods, permitting time comparisons and quantitative image comparisons without extraneous differences.

A logical overview of the ray-casting method appears in Figure 2. Observe that `rayCastRegion` is recursive. Its job is to compute the contribution of one ray through the specified region, which it either does directly or by calling itself on the necessary subregions.

For orthographic projection (parallel rays), a single permutation gives a front-to-back topological order for all subregions, making `findRegion` simple and efficient. For perspective, rays may require varying permutations, but the permutation is fixed within an individual ray.

The function `findExit` can use a generalization of the Snyder and Barr method for regularly tesselated grids [SB87], which can be thought of as merging three ordered lists of intersection points. Three ray parameter values t_x, t_y, and t_z, are maintained, representing the most imminent intersections of the ray with x, y, and z grid planes. The minimum is selected as the next intersection parameter. In the uniform case, the selected parameter can be immediately updated by a fixed amount. For example, if t_x is selected, it is updated by a Δt_x that jumps to the next x plane of the grid.

In our extension to the adaptive case, "updates" of the same value (`subEntry`) may occur at several tree levels, so they are done nondestructively by `findExit`. Moreover the update amount is $w_x \cdot \Delta t_x$ (if t_x was selected), where w_x is the x width (measured in grid points) of the subregion being "jumped over".

Most previous works on ray-casting acceleration techniques [Lev90, DH92, Wes94] use variations of regular sampling. Upson and Keeler in their seminal work intersect a cell face but then step along the ray through the cell to integrate [UK88]. Levoy skips transparent regions (based on an octree), then steps at equal intervals through regions to be colored, evaluating the function by interpolation at the full resolution of the data [Lev90]. Danskin and Hanrahan adaptively vary the step size while integrating color and opacity along the ray, and approximate the data by an average [DH92]. Westermann varies step size with the multiresolution scale, and interpolate the data with wavelets [Wes94].

Instead of using a regular sampling technique, we integrate directly from the entry point to the exit point, approximately solving the underlying differential equation. The approximation is the same as used by the coherent projection method [WVG91], to find the color at the thickest point of a cell's projection. This gives the "emission" contribution of the region for this ray.

We have also experimented with reflective shading and depth attenuation to convey a sense of shape in the image (Section 5). These effects are subjective and *ad hoc*, and are optionally invoked by the user. However, lack of shape cues is often an obstacle in understanding volume rendering, and deserves further investigation. Sobierajski and Kaufman [SK94] have described methods for shape cues based on introduction of familiar geometric objects.

As mentioned at the beginning of the section, we already have a flexible data structure and algorithm for traversing the hierarchy selectively. The selective traversal examines only the part of the tree that the ray will pass through. The selective traversal algorithm takes care of all the restrictions defined by user parameters, including data importance, error tolerance, being within clipping planes, etc. The hierarchical raycasting algorithm determines through which subregions of the current region the ray will pass, as outlined above, and selectively traverses over those subregions. The *mdh* tree stores the trilinear coefficients of the node that embraces a region in the volume data. These coefficients are the least-squares trilinear approximation of the data region. If the algorithm decides stops the refinement at the node that is higher than bottom level which is original data, it evaluates the data, color and opacity at the intersection points of the bounding box of the region using these trilinear coefficients. If the algorithm goes down to the data level, which means the ray is passing through a relatively less homogeneous region, it looks up the eight corner data values of the cell or voxel, and computes colors and opacity at the intersection points of the cell boundary.

Raycasting is easily parallelized, because each ray's computations are nearly independent. Our implementation has adopted the simple `m_fork` facilities for multiprocessing on SGI workstations.

5 Shading

We implemented diffuse and specular shading within the ray casting volume renderer. Parameters for shading include amount of diffuse shading, amount of specular shading, specular shininess, and opacity of the shading contribution. Diffuse and specular shading are calculated according to the usual model. Diffuse reflection is assumed to be the color of the transfer function at the location being shaded, while specular reflection is white. Parameters limiting the use of shading include the range of data that is to be shaded and the minimum gradient magnitude for shading contributions. Data outside the shading range or with a gradient less than a specified threshold isn't shaded.

Shading occurs along a given ray through a region if either the entry data value or exit data value are within the shading range, and if the entry gradient or exit gradient magnitudes are above the threshold. In that case, the data value used for shading (which provides the diffuse reflectivity) is taken to be the average data value of the front and back ray-region intersections. The gradient is the average of the front and back gradients, unless one gradient magnitude is out of range. In that case, the gradient that is within range is used.

We use the derivative of the trilinear function as the gradient of the data. There are advantages and disadvantages to this. The advantage is that the function is already being used to model data at interior node of the tree, so is readily available. It is also used to calculate interpolated data values within the data itself. The disadvantage is that the gradient at a shared cell face may not be continuous. We are presently exploring the amount of error this may introduce. An alternative is to use finite differences for the gradient within the data.

6 Comparison of Rendering Methods

Our hierarchical visualization software provides three methods of direct volume rendering: coherent projection [WVG91, WVG94], hardware 3D textures [WVGW94], and ray-casting.

Coherent projection uses hardware-assisted Gouraud shading to provide a fast and reasonably good quality image [WVG91, LH91, ST90]. Assuming parallel projection, a projection of one region (the template) is always a uniformly scaled version of the projection of any other. Color and opacity contributions are calculated only at the projected region's vertices, and hardware Gouraud shading is used to find values across it. Because perspective projection makes each region's projection different, coherent projection is much slower if it is used. Because there is limited control over color and opacity across the region, shading is usable, but not of the best quality. The method does fit well into a hierarchical visualization system because each region is rendered independently and in visibility ordering. It runs on standard graphics hardware. While parallelization is not as straight-forward as with a ray-caster, images could be created in parallel from different regions of the tree and composited.

Hardware 3D texture mapping is a very fast method available on machines that have this feature [CCF94, GL94, WVGW94, CN93]. (An example of such a machine is the SGI Reality Engine.) In this case, a version of the data taking into account distance between data points is stored as a 3D texture map. The volume is created by drawing polygonal texture-mapped slices through the texture map. The specialized texture-mapping hardware does most of the work of creating the image. Because the size of a 3D texture map is usually small, several texture maps may be required to create the entire volume. The image quality can be quite good (as least as good as coherent projection) if sufficient planar slices are used. The relative speed we suggest is assuming trilinear interpolation, 12-bit textures, and about twice the number of planar slices as there are data points along the long diagonal of the volume. However, shading is only possible if the lighting effects are precalculated (thus the light must move with the volume and specular effects are inaccurate). Because specialized hardware is used, it cannot be parallelized. Flexible error-controlled traversal would be difficult because the method uses precalculated rectilinear chunks of the volume for imaging.

We have only recently added ray-casting to our hierarchical system. As is well known, ray-casting provides the best quality images and most control. Even without shading, the images appear more crisp and less blurry than with coherent projection or texture mapping. It is relatively easy to add shading and to insert geometric objects into the volume. Perspective does not incur any significant extra penalty. Ray casting can be used with an error-controlled hierarchy, though it is noticeably slower than ray casting a non-hierarchical rectilinear volume. The method runs on standard hardware and is easily parallelizable. However, it is slow, compared to the above approaches.

It is certainly desirable to have a range of methods available, to provide for the whole range from excellent image quality to interactive speeds. The comparative abilities of the various methods are summarized in Table 1.

	Coherent Projection	Ray Casting	Hardware 3D Textures
Shading	-	+	0
Perspective	0	+	+
Image Quality	-	+	-
Error Control	+	+	-
Standard Hardware	+	+	0
Parallelizable	-	+	0
Relative Speed (1-proc)	100	2500	3

Table 1: Comparison of Three Basic Methods: Coherent Projection, Ray Casting, and Hardware 3D Textures. This table summarizes the abilities of the three methods for different characteristics related to image quality (above) and speed (below). (The relative speed of 3D textures is very approximate, depending on the type of map and size of volume.) A "+" in the table indicates the method is related to this characteristic; a "-" means it is adequate; a "0" means it is poor. See text for discussion.

7 Experimental Results

We compared the hierarchical volume rendering using our three methods and allowing different amounts of error, examining in particular speed and image quality.

Table 2 shows timing results on two data sets.[1] It is clear that for fast rendering hardware 3D texture mapping is the best method, but machines with this capability are still rare. We observe a rather shocking speed degradation in ray casting with the hierarchy rather than on the original data, and are pursuing improved traversal methods. However, on volumes where importance can be used to limit traversal, hierarchical ray casting usually is faster than regular grid ray casting.

Error tolerance is measured as a fraction of the standard deviation of the data, but also scaled to the size of the region. For example, a region comprising 1/64-th of the volume is expected to have 1/8 as large a standard deviation as the entire volume, by the square-root rule, so a threshold of .10 means that such a region is adequately approximated if its own RMS error is 1/10-th of that expected quantity. While this is a bit complicated to explain, it makes the tolerance a non-dimensional quantity that has about the same significance across a variety of datasets.

The slides demonstrating these methods are described below. The hipip volume is imaged without any importance function. Red denotes positive high potential, blue denotes negative high potential, and white denotes very low potential. No visual differences could be observed using an importance function that reduced weight on very low potential data. Quantitative comparisons are discussed later. The CT head images used an importance function that reduced weight in regions containing soft tissue, and gave zero weight to air.

1. *Slide 1* compares ray casting and coherent projection on the hipip data set. The lower left image shows coherent projection with no error and the lower right image shows coherent projection with 10% error. The upper left image shows ray casting with no error and the upper right image shows ray casting with 10% error. Notice the sharper outline of the red and blue features in the ray cast image, and the smaller amount of image degradation that occurs using ray casting with error.

2. *Slide 2* compares ray casting with and without shading on the hipip data set. The lower left image

[1] Hipip (High Potential Iron Protein) is from L. Noodleman and D. Case, Scripps Clinic, La Jolla, Ca. The CTHead and its down-sample (the same data set) was from UNC.

Data Set	error	Impor--tance	Coherent Projection	Regular Grid Raycasting	Hierarchical Raycasting	Parallel Hierarchical Raycasting	Volume Texture
Hipip	0	no	18.9	121.00	337.0	97.0	0.0
	10 %	no	0.89	-	103.0	27.0	-
	30 %	no	0.23	-	52.0	13.0	-
	70 %	no	0.07	-	44.0	11.0	-
	0	yes	5.66	-	183.0	49.0	-
CT Head 128	0	no	54.15	181.0	409.0	126.0	1.5
	1.0 %	no	44.48	-	360.0	103.0	-
	15 %	no	15.65	-	199.0	58.0	-
	0	yes	29.32	-	255.0	83.0	-
	1.0 %	yes	21.79	-	216.0	67.0	-
	15 %	yes	3.25	-	174.0	50.0	-
CTHead 256	0	no	247.58	380.0	704.00	200.0	2.06
	1.0 %	no	214.0	-	608.0	170.0	-
	15 %	no	43.5	-	269.0	74.0	-
	0	yes	55.84	-	205.0	60.0	-
	1.0 %	yes	51.84	-	201.05	59.0	-
	15 %	yes	27.98	-	175.0	56.0	-

Table 2: CPU time comparisons (in seconds) on SGI Reality Engine II with four 150 MHz processors, 64MB memory. The parallel time denotes the maximum CPU time of the four processors. Error tolerances are explained in the beginning of Section 7
. Volume sizes are: Hipip 64^3; CT head, $256 \times 256 \times 113$; down-smapled version, $128 \times 128 \times 113$.

shows ray casting with no error and the lower right image ray casting with 10% error. The upper left image shows ray casting with diffuse and specular shading and no error, and the upper right image shows ray casting with shading and 10% error. Shading helps considerably in bringing out features in the data and establishing orientations.

3. *Slide 3* compares ray casting (right) with no error and 3D texture mapping (left). Again, the main difference is the crispness of edges around features. 3D texture mapping generally produces a picture very close to that seen with coherent projection. However, it does not easily accommodate shading.

4. *Slide 4* compares ray casting and coherent projection on a down-sampled version of the CT head (128x128x56) and restricted to the front of the skull. The transfer function is selecting for bone only. The lower images, left to right, show coherent projection, ray casting, and ray casting with shading and no error. The upper images show the same methods with 15% error. The shading uses a colored light in this case. Again ray cast images are crisper and more tolerant of error, and shading helps bring out features.

5. *Slide 5* shows a ray cast image of the whole CT head data set (256x256x113) without error or shading.

6. *Slide 6* shows a ray cast image of the whole CT head with shading and no error. We find that shading on this volume is not very smooth, as the data is not filtered and small variations in gradient across

error	importance	rms	max
0	yes	0.0028	0.015
10 %	no	0.0085	0.078
10 %	yes	0.0161	0.122
30 %	no	0.0284	0.341
30 %	yes	0.0291	0.341

Table 3: Image comparisons with raycast reference image of Hipip. Subject images are also raycast.

error	importance	rms	max
0	yes	0.007	0.216
1 %	no	0.137	0.078
1 %	yes	0.133	0.122
15 %	no	0.137	0.341
15 %	yes	0.133	0.341

Table 4: Image comparisons with raycast reference image of CTHead-128. Subject images are also raycast.

the volume cause shading variations. This phenomenon may also be due to our use of the trilinear function to calculate gradients; we are exploring these issues.

7.1 Degradation with Error Tolerance

We observed that image quality degraded less with error tolerance when using raycasting, as opposed to projection with hardware interpolation. This occurred even though both methods used exactly the same approximations to the field function while traversing the volume.

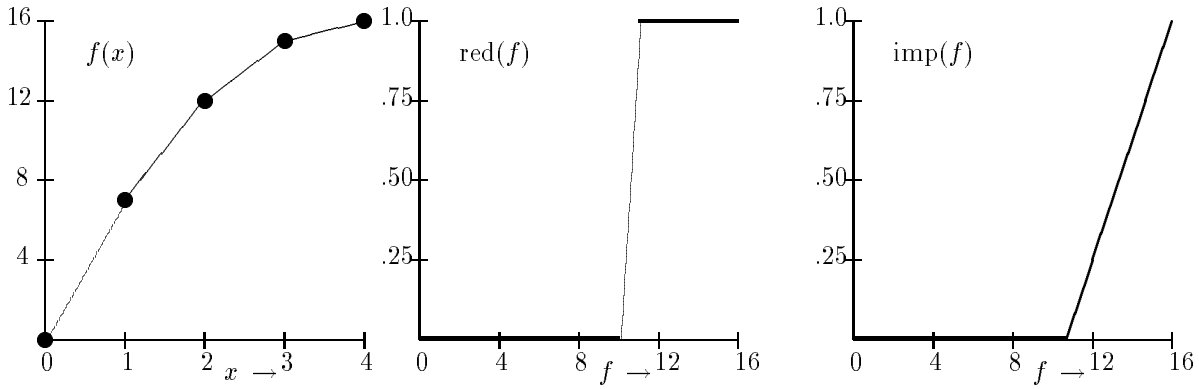
The explanation for this observation lies in the nonlinearity of the transfer function that maps data to color and opacity. Figure 3 demonstrates on a simplified example. Intuitively, both methods interpolate the field function in all three dimensions, and map those values to color exactly, and finally perform approximate color integrations in "screen z" (along the sight line). However, raycasting computes a new color mapping and approximate integral at each pixel, while the projection method does so only at polygonal vertices. Consequently, the projection method effectively approximates color in "screen x" and "screen y", as well as "screen z".

In the one-dimensional example, cell boundaries are at the integral values of x and we assume the field function f is piecewise linear as shown at the upper left. Solid dots denote sample data values at cell boundaries.

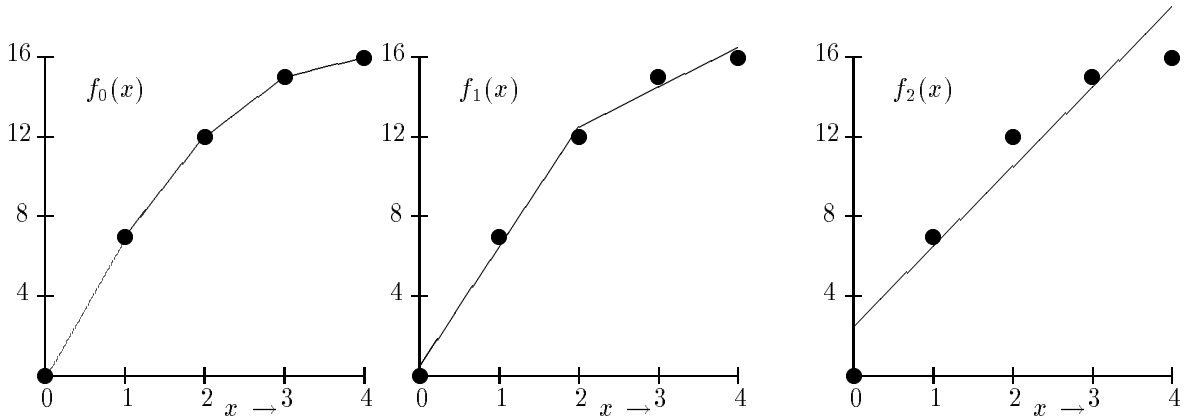
The color rises sharply just below $x = 2$, even though $f(x)$ is linear across the cell. Raycasting produces an accurate picture, as it maps f to color repeatedly and detects the nonlinearity. The projection method maps f to color on boundaries only, and therefore must interpolate color across the interior, introducing inaccuracy, as shown in the lower left diagram. In all image diagrams the dotted line represents the "true" image.

For this example, an unweighted error tolerance might permit the right half of the line to be approximated at height 1, while the left half is refined, as the right half has lower RMS error at height 1. This happens to introduce no additional error for either method, as shown in the middle diagrams of the third and fourth rows.

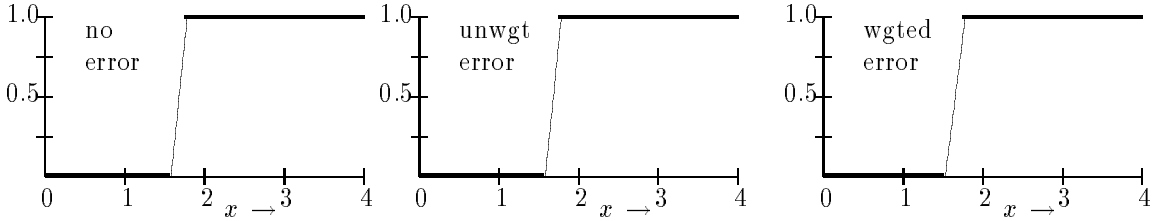
An importance-weighted error causes the left half to have lower error than the right. For certain thresholds, the left half will be approximated at height 1 and the right half will be refined to height 0. Since the nonlinearity of the transfer function occurs in the left half, the error is even more exacerbated in the projection image, as shown in the lower right diagram. Error is introduced into the raycast image also, as shown in the third row, right, but it is less pronounced.



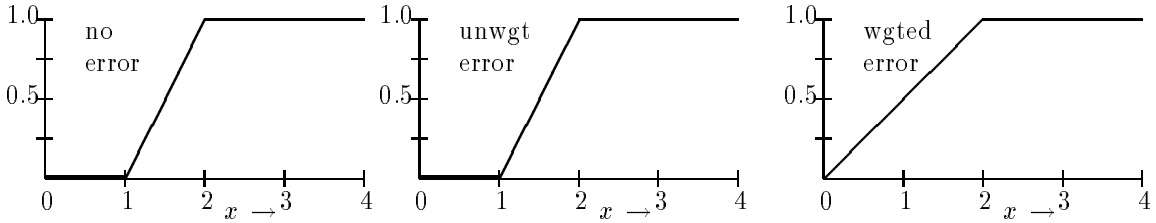
Example field function f, color function red(f), and importance function imp(f)



Approximations of f at heights 0 (full resolution), 1 and 2



Raycast of f at full resolution, with unweighted error, and with importance-weighted error



Projection of f at full resolution, with unweighted error, and importance-weighted error

Figure 3: Nonlinearities in transfer function (red(f)) cause larger errors with projection methods, relative to raycasting, even though both use the same approximation to the field function f as discussed in Section 7.1.

7.2 Quantitative Image Comparisons

Raycast images made with various acceleration methods were compared with reference images.[2] Tables 3 and 4 show the results. The framebuffers of the reference image and the subject image were compared pixel by pixel. Each difference lies in the range -1.0 to 1.0. The maximum absolute error and the root-mean-square (RMS) error are reported for each subject image.

The main observation about these data is that the user-supplied importance functions have much less effect on the image quality than the choice of error threshold, yet they still achieve substantial accelerations, as shown in Table 2. This is not very surprising for the CT head, because the zero-importance range corresponds to air, which is transparent in the transfer function. However, the Hipip importance function assigns varying importance to different potentials (the field function) that are not transparent in the transfer function. It achieves accelerations by factors of two and more with imperceptible degradation of the image.

8 Conclusions

We have presented a method of ray casting error-controlled hierarchical volumes using cell face intersection rather than sampling along the ray. We have integrated a shading model into the ray caster. We have compared ray casting to coherent projection and hardware 3D texture maps.

We conclude that ray casting is a considerable improvement to faster projection-based volume rendering methods in terms of image quality, though it incurs a major speed penality. A range of rendering methods within a single visualization system is desirable, and that shading makes images more comprehensible.

User-supplied data-importance functions were found to be a simple and highly effective way to cut rendering times by half and more with little effect on image quality in most cases. These can be added to the toolkit of acceleration methods for both ray-casting and projection methods.

The analysis of projection errors relative to ray-casting showed that they are attributable to the nonlinearity of the color transfer function. That is, color interpolation errors are more serious than data interpolation errors. This suggests an avenue for improvement of projection methods on workstations with hardware texture maps. Max *et al.* have made a step in this direction for cells of uniform color and varying thickness [MBC93]. We plan to investigate an extension to handle nonuniformly colored cells, by employing texture lookup tables in conjunction with texture maps.

References

[CCF94] Brian Cabral, Nancy Cam, and Jim Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *1994 Symposium on Volume Visualization*, pages 91–98, Washington, D.C., October 1994.

[CN93] T. J. Cullip and U. Newman. Accelerating volume reconstruction with 3d texture hardware. Technical Report TR93-027, University of North Carolina, Chapel Hill, N. C., 1993.

[DH92] John Danskin and Pat Hanrahan. Fast algorithms for volume ray tracing. In *1992 Workshop on Volume Visualization*, pages 91–98, Boston, Mass., October 1992. ACM.

[GL94] S. Guan and R. G. Lipes. Innovative volume rendering using 3d texture mapping. In *SPIE: Medical Imaging 1994: Images Captures, Formatting and Display.* SPIE 2164, 1994.

[Lev90] Marc Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, July 1990.

[2] Due to an alignment bug, we were unable to compare raycast and projection framebuffers, or raycast and 3D-texture framebuffers.

[LH91] David Laur and Pat Hanrahan. Hierarchical splatting: A progressive refinement algorithm for volume rendering. *Computer Graphics (ACM Siggraph Proceedings)*, 25(4):285–288, July 1991.

[MBC93] Nelson Max, Barry Becker, and Roger Crawfis. Flow volumes for interactive vector field visualization. In Nielson and Bergeron, editors, *Visualization '93*, pages 19–24, San Jose, Ca, October 1993. IEEE.

[SB87] J. Snyder and A. Barr. Ray tracing complex models containing surface tessellations. *Computer Graphics (ACM Siggraph Proceedings)*, 21(4):119–128, July 1987.

[SK94] L. M. Sobierajski and A. E. Kaufman. Volumetric ray tracing. In *ACM Workshop on Volume Visualization 1994*, pages 11–18, Washington, D.C., October 1994.

[ST90] Peter Shirley and Allan Tuchman. A polygonal approximation to direct scalar volume rendering. *Computer Graphics*, 24(5):63–70, December 1990.

[UK88] Craig Upson and Michael Keeler. The v-buffer: Visible volume rendering. *Computer Graphics (ACM Siggraph Proceedings)*, 22(4):59–64, July 1988.

[Wes94] Ruediger Westermann. A multiresolution framework for volume rendering. In Arie Kaufmann and Wolfgang Krueger, editors, *1994 Symposium on Volume Visualization*, Washington, D.C., October 1994. ACM.

[WVG91] Jane Wilhelms and Allen Van Gelder. A coherent projection approach for direct volume rendering. *Computer Graphics (ACM Siggraph Proceedings)*, 25(4):275–284, 1991.

[WVG92] Jane Wilhelms and Allen Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992. Extended abstract in ACM Computer Graphics 24(5) 57–62; also UCSC technical report UCSC-CRL-90-28.

[WVG94] Jane Wilhelms and Allen Van Gelder. Multi-dimensional trees for controlled volume rendering and compression. In *ACM Workshop on Volume Visualization 1994*, Washington, D.C., October 1994. See also technical report UCSC-CRL-94-02.

[WVGW94] Orion Wilson, Allen Van Gelder, and Jane Wilhelms. Direct volume rendering via 3d textures. Technical Report UCSC-CRL-94-19, CIS Board, University of California, Santa Cruz, 1994.