A Coherent Projection Approach for Direct Volume Rendering

Jane Wilhelms and Allen Van Gelder
Computer and Information Sciences
University of California, Santa Cruz 95064

Abstract

Direct volume rendering offers the opportunity to visualize all of a three-dimensional sample volume in one image. However, processing such images can be very expensive and good quality high-resolution images are far from interactive. Projection approaches to direct volume rendering process the volume region by region as opposed to ray-casting methods that process it ray by ray. Projection approaches have generated interest because they use coherence to provide greater speed than ray casting and generate the image in a layered, informative fashion. This paper discusses two topics: First, it introduces a projection approach for directly rendering rectilinear, parallel-projected sample volumes that takes advantage of coherence across cells and the identical shape of their projection. Second, it considers the repercussions of various methods of integration in depth and interpolation across the scan plane. Some of these methods take advantage of Gouraud-shading hardware, with advantages in speed but potential disadvantages in image quality.

1 Introduction

The two main approaches for rendering three-dimensional scalar sample volumes are extraction of isosurfaces [LC87] and direct volume rendering [DCH88, Lev88, Sab88, UK88, Wes90, MHC90, ST90]. While extraction of isosurfaces produces clearcut delineation of features, the binary decision made about the location of surfaces means that only a limited amount of the total information contained in the volume can be presented in one image. Direct volume rendering can use semi-transparency to visualize much more of the volume contents, and all cells become capable of contributing to the image. The use of independent transfer functions to map the volume's original scalar values to color and opacity makes it possible to combine continuous variations with feature extraction in a very flexible fashion.

Because the amount of information presented in one directly rendered image and its sometimes blurry appearance can make it difficult to fully understand, the ability to view the volume interactively from various positions is of considerable importance. Unfortunately, direct volume rendering is very expensive and most animations depend upon precalculated images which are replayed as film loops. Techniques that do provide fast rendering tend to do so at the expense of image quality.

This paper presents an approach to direct volume rendering using projection of individual volume cells [DCH88, UK88, Wes90]. Processing is cell by cell, not pixel by pixel as in the alternative ray-casting approach [Lev88, Sab88, UK88]. Projection seems in many ways preferable to ray casting. It can take advantage of coherence when a cell projects onto many neighboring pixels. It can avoid some of the aliasing inherent in point-sampling approaches. It can process the image plane by plane, so that if the total rendering time is considerable, the viewer can gain useful information during the rendering process by watching the image being created. This is particularly useful if the image is drawn back to front, because regions of the image that might be obscured later are all visible at some point during rendering. It has been suggested that projection is more amenable to parallel processing [UK88].

The coherent projection approach presented here takes advantage of the regularity of rectilinear volumes in two ways. First, the projection of each cell is an exact but translated geometrical copy of every other cell, so an analysis of the geometry of a single cell can be used to hasten processing of every other cell. This assumes parallel projection, though perspective projection is possible if cells are individually projected at some extra cost. Second, a cell projects as one to seven polygons. The intensity and opacity values of the pixels involved can be found by interpolating (in one of several ways) between their vertices.

Several methods are presented for calculating cumulative intensity and opacity by integration in depth and interpolation across the scan plane. They are investigated in terms of their speed and image quality.

2 Coherent Projection Algorithm

A *cell* refers to the rectilinear region bounded by eight neighboring sample points, the *cell corners.* Cells are modeled as containing a semi-transparent material that both emits light and occludes it in amounts dependent upon the scalar data values of the cell corners and their mapping to intensity (red, green, and blue) and opacity values using transfer functions. The coherent projection method depends upon three observations concerning the parallel projection of rectilinear cells. (For simplicity, cells are assumed to be identical in size. The algorithm also works for hierarchical volumes where cells are uniformly scaled versions of each other [WCA+90].)

1. The projection of each of these cells is geometrically a translated copy of the projection of any one cell.
2. From any eyepoint, a generic cell can be simplified into at most seven subcells with the same front and back face.
3. The projection of these subcells is either a triangular or a quadrilateral *projected polygon.*

The first step of coherent projection is to determine an appropriate template for the particular shape and orientation of the cells making up the volume (Section 2.1). The volume is then traversed, relying on the template for geometry, but taking into account the unique data values of each cell (Section 2.2). For each cell, intensity and opacity values for the subcell vertices are determined (Section 2.3), and then intensity and opacity values of the pixels that the subcells project onto are determined using interpolation (Section 2.4). These pixels are composited with the accumulating image in the frame buffer (Section 2.5).

2.1 Determining the Generic Cell Type

Depending upon orientation, one, two, or three faces will form the front of the rectilinear cell (Figure 1). If the projection of the cell is divided into regions having the same front and back face, up to seven *subcells* result from each cell. (Use of coherence in regions with the same front and back cell faces has also been investigated elsewhere [UK88, MHC90, ST90].) Subcells are polyhedra whose front and back faces project to the same screen location and form *projected polygons.* Thus, each projected polygon vertex can be thought of as a *vertex pair*,consisting of a front vertex and a back vertex which may have no distance between them along silhouette edges. Some subcell vertices are original cell corners. Others are *intersection points* that must be calculated. The distance from front to back between the vertex pairs is the same for all pairs with a non-zero distance, and can be precalculated.

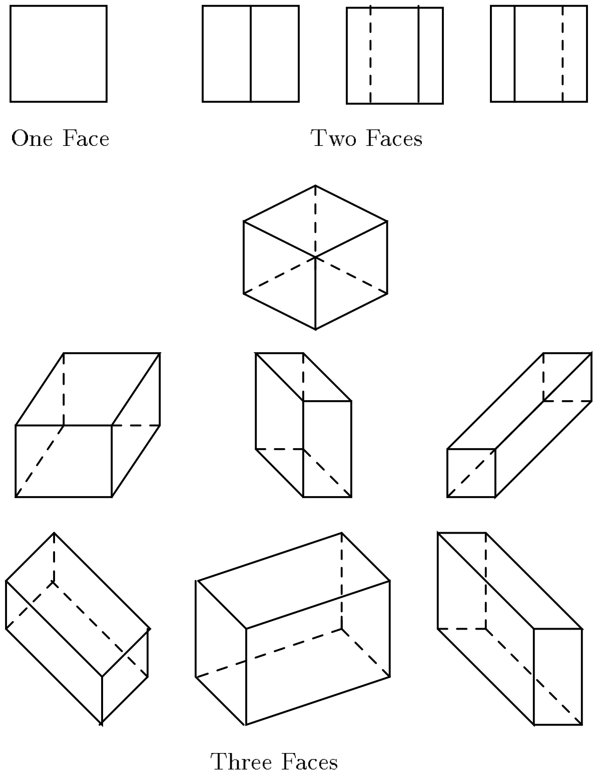

Figure 1: Cell Projections

When only one face is visible (see Figure 1), there is only one projected quadrilateral polygon, and its vertices are projections of original cell corners.

When two faces are visible (e.g., front and left), two or three quadrilateral projected polygons face the front. Two corners of the original cell are nearest the front, and two are farthest away. If the two farthest vertices project exactly onto the two nearest, no new intersection points are needed and two projected polygons result. Otherwise, a face is split and four new intersection points located on cell edges must be found. In this case three projected polygons result.

If three faces are visible, six or seven polygons are needed to represent the cell. There are seven possible subcases, determined by the location of the farthest cell corner relative to the nearest. If the farthest cell corner projects onto a face (three cases), four new edge intersections and two new face intersections must be calculated. If the farthest cell corner projects onto a line (three cases), only two new edge intersections are needed. If the farthest cell corner projects onto the nearest, no new intersections are needed.

In this implementation, a table was generated to specify each of these cases, assuming they represent front, front-left, and front-left-bottom faces. All other cases are mapped to these. The table describes which edges and faces contain intersections, which cell corners and intersection points form vertex pairs, and which front vertices form polygons. Once a generic cell type is determined the table is used to find appropriate properties from one particular generic cell. These values are stored and reused for all other cells.

2.2 Traversal of the Volume

The algorithm works either by traversing the volume back to front, or front to back. There are advantages to both traversals. Back-to-front traversal avoids the need for an opacity buffer to store accumulated opacity values. Furthermore, this traversal shows each layer of the image in front of the last, so all cells at some time appear in front. If the traversal is front to back, an opacity buffer is needed. A potential advantage of front-to-back rendering is that rendering can be bypassed for the new cells that lie behind fully opaque cells. This requires a suitable saving strategy, which is being investigated.

Because of the regularity of the volume, an appropriate traversal order can be determined from its orientation. (The transformation matrix can provide this.) More than one traversal order is possible. For example, if only the front face is visible, the traversal can be XYZ, YXZ, $-XYZ$, etc. The current implementation traverses the volume in the order that accesses the data most efficiently: if X varies fastest, an XYZ traversal is used when possible. This means that sometimes the slice whose projection covers the smallest area is projected, making the image less understandable as it is developing. A more visually appealing alternative would be to project the slice with the largest projected area first.

2.3 Integration in Depth

Each cell is assumed to consist of a semi-transparent material which emits its own light, transmits some light coming from behind, and occludes some light both from behind and from within the cell. Such a model is related to previous work in computer graphics for modeling semi-transparent media [Bli82, KH84, Max86, Sab88]. Luminosity and occlusion are represented as intensity and opacity.

To create the image, the scalar values from the original data are converted to intensity and opacity values using transfer functions stored in red, green, blue, and opacity tables. Estimated data values for edge intersections are found by linear interpolation between adjacent corner vertices, and for face intersections by bilinear interpolation of face vertices; these values are then used for mapping.

The cumulative intensity and opacity contribution of the medium between the front and back cell face must be determined. This process will be referred to as *integration in depth*, and involves solving (approximately) a linear differential equation. The discussion that follows is general and is applicable to any method that seeks the intensity and opacity contribution along the line of sight through the volume, such as ray casting approaches that trace between entry and exit points of cells [UK88]. Three approaches are described, providing a trade-off between image quality and speed. For coherent projection, one of these methods is used to find intensity and opacity at projected polygon vertices.

2.3.1 Some Definitions

The intensity and opacity contribution of a cell at each pixel (or along each line of sight ray) should take into account the emission and occlusion properties of the material being modeled. We define *material opacity* as the fraction of light entering from behind that would be occluded *if that material were present for a depth distance of 1*. For example, if the material opacity value is 0.5, 50% of any light coming from behind the material would be removed if the material were present for a depth of one unit distance. Our implementation treats the value returned by the opacity transfer function as the material opacity (in the range 0 to 1) and converts it into the differential opacity (in the range 0 to ∞) as described below. (It would also be reasonable to specify that the transfer function return the differential opacity directly, but then there is some awkwardness about representing ∞.)

The *differential opacity* of a material, denoted by Ω, is defined as the rate at which light is occluded as it travels through the material. That is, in an infinitesimal distance dz, a fraction $\Omega(z)dz$ of light is occluded, and $(1-\Omega(z)dz)$ of the light is transmitted. From these definitions it follows (see Equation 7) that a differential opacity Ω acting over a unit distance gives a material opacity

$$O = 1 - e^{-\Omega} \tag{1}$$

Inverting this function,

$$\Omega \quad = \quad \log\left(\frac{1}{1-O}\right) \tag{2}$$

The color transfer functions return *material intensity*, which is defined as the amount of light emitted by the material if present for a depth distance of 1. However, the material intensity value is the same as the differential intensity value.

The intensity and opacity contributed by the material between the front and the back of a cell along the line of sight are called *cumulative intensity* and *cumulative opacity*. These quantities are defined as solutions of differential equations; they are found and used in compositing.

2.3.2 Integration Methods

Throughout this discussion we use *integration* somewhat liberally to mean (exact or approximate) solution of a differential equation. Three approaches have been implemented. In all cases the underlying differential equations are the same, as described next.

Let z represent distance through the cell from the back vertex. Let $\Omega(z)$ be differential opacity and let $E_c(z)$ be the differential intensity of light of color c (which, as mentioned before, is the same as material intensity). Let $T(z)$ denote the fraction of light entering the cell that is transmitted through distance z (needed for compositing). Let $I_c(z)$ denote the intensity of light of color c that is emitted within the cell and reaches z. Then:

$$\frac{dT}{dz} = -\Omega(z)T(z) \quad (3)$$

$$\frac{dI_c}{dz} = -\Omega(z)I_c(z) + E_c(z) \quad (4)$$

The boundary conditions are that $T(0) = 1$ and $I_c(0) = 0$. Letting d be the back-to-front distance between vertices, the cumulative intensity of color c is $I_c(d)$ and the cumulative opacity is $O_{cum} = (1 - T(d))$.

Assuming $\Omega(z)$ has a closed form integral (it is normally a simple interpolation function of some kind), Equation 3 has the standard closed form solution,

$$T(z) = e^{-\int_0^z \Omega(u)\,du} \quad (5)$$

The solution of Equation 4 can also be expressed as an integral:

$$I_c(z) = T(z)\int_0^z \left(\frac{E_c(v)}{T(v)}\right) dv \quad (6)$$

However, this integral has a closed form expression only in special cases. For example, if there is a constant μ_c such that $E_c(z) = \mu_c\Omega(z)$, then the substitution $U_c(z) = I_c(z) - \mu_c$ in Equation 4 gives a differential equation for U_c that is the same as Equation 3 (except for the boundary condition), and $I(z)$ is "as solvable as" as $T(z)$; this method is employed by Max *et al.* [MHC90]. The conditions $E_c(z) = \mu_c\Omega(z)$ effectively restrict the mapping from scalar values to a single transfer function plus different multipliers μ_c for different colors. It is not employed here because we wished to permit the specification of independent transfer functions for all four quantities.

For the rest of the discussion, we shall drop the subscript c on intensities E and I, with the understanding that intensity calculations are done independently for each color.

An important case that has closed form solutions is used in several of the methods, sometimes as an approximation: that is when $\Omega(z)$ and $E(z)$ are constant within the cell. Then we have

$$T(z) = e^{-\Omega z} \quad (7)$$

$$I(z) = \frac{E}{\Omega}\left(1 - e^{-\Omega z}\right) \quad (8)$$

Substituting d for z gives the cumulative values, $O_{cum} = (1 - T(d))$ and $I_{cum} = I(d)$.

1. Average C*D Integration: A simple way to determine cumulative intensity and opacity is to average the front and back intensity and differential opacity values, and to approximate $(1 - e^{-\Omega d})$ by $\min(1, \Omega d)$ in Equations 7 and 8. This gives

$$O_{cum} = \min(1, \Omega_{ave} d) \quad (9)$$

$$I_{cum} = \left(\frac{E_{ave}}{\Omega_{ave}}\right) O_{cum} \quad (10)$$

The name "C*D" is based on the fact that in the common case that $O_{cum} < 1$, the cumulative intensity becomes "color times distance".

This is not always a desirable choice: for example, the intensity and opacity calculated for a ray through a homogeneous volume (constant data values throughout) will generally not be the same if the volume is treated as one large cell compared to the same volume treated as many small cells composited together (see Section 3.1). However, this approach is fast, as no transcendental functions are used, and for large volumes may be visually indistinguishable from more expensive methods (see Section 3.4). This method tends to overestimate intensity and opacity.

2. Exponential Homogeneous Integration: A more complex and yet computationally acceptable method is to assume the region between the front and back of the cell is *homogeneous.* The front and back material intensity and opacity are averaged to provide the average material intensity E_{ave} and average material opacity O_{ave}. Then O_{ave} is converted to Ω_{ave} by Equation 2, and the closed form solutions in Equations 7 and 8 are used with E_{ave} for E and Ω_{ave} for Ω. Related exponential methods have been used by other visualization researchers [MHC90, ST90, Sab88]. The distance d between front and back vertices can be scaled by a user-defined factor for flexibility.

3. Exponential Linear Integration: Here the material intensity and opacity are assumed to vary *linearly* (but independently) between front and back cell faces. A closed form solution does not exist in general, and numerical solution is used. The user defines a number of divisions between the front and back faces, and linear interpolation is used to estimate the material intensity and opacity values at those points. Then exponential homogeneous integration is applied to each of the subregions, and these are composited as described in Section 2.5. As mentioned before, Max *et al.* [MHC90] require that intensity is some constant multiple of opacity, and use the faster analytical solution based on Equation 5 and the discussion following it.

2.4 Interpolating to Pixel Values

Once the intensity and opacity of the polygon vertices have been determined, it is necessary to find the values of the pixels that lie between them and are projected onto. These pixels are located by the usual process of polygon scan conversion, either in hardware or software. A major motivation of this method is that this job can be relegated to hardware. The cumulative intensity and opacity associated with the pixels projected onto are found by one of the three methods described below. These will be referred to as *interpolation* methods, though, more correctly, they are a choice of which integration to apply to each pixel. Again, there is a tradeoff of cost and accuracy. Three approaches have been implemented:

1. Gouraud Shading Interpolation: This is an extension of the common Gouraud shading model [Gou71] which first linearly interpolates cumulative color and opacity along edges from scanline to scanline, and, within each scanline, linearly interpolates between the values for edge pairs. Shirley and Tuchman [ST90] recently published a method using Gouraud-shaded tetrahedrons for volume rendering.

The assumption of a linear variation between vertices is not completely in keeping with the basic model of a semi-transparent gas. However, the method often works quite well in practice (see Section 3). Many graphics workstations have hardware-assisted Gouraud shading and this step can done efficiently and in parallel with the rest of the volume renderer. It is essential, however, that the Gouraud shader interpolates in the opacity channel, as well as red, green, and blue. The limited precision of interpolation in hardware can also cause aliasing, so a software version which works in floating point has also been implemented.

2. Exponential Homogeneous Interpolation: This uses exponential homogeneous integration to calculate the cumulative intensity and opacity at each pixel. First, the average material intensity and opacity are found for each projected polygon vertex. Then, polygon scan conversion is used to linearly interpolate across the projected polygon face and find the average material intensity and opacity at each pixel.[1] Finally, exponential homogeneous integration is used between front and back faces to find the cumulative intensity and opacity at each pixel.

3. Exponential Linear Interpolation: This approach uses exponential linear integration at each pixel projected onto. In this case, the material intensity and opacity values of the front and back face polygons are linearly interpolated separately, again using scan conversion to find these values at each pixel. Then exponential linear integration is applied at each pixel. This is most closely related to the work of Max et al.[MHC90], but permits separate independent transfer functions for different colors and opacity.

2.5 Compositing

Compositing is used at each pixel to combine the effects of cells that project there [PD84]. For back-to-front traversal

$$C_{acc} = (1 - O_{new})C_{acc} + C_{new} \tag{11}$$

$$O_{acc} = (1 - O_{new})O_{acc} + O_{new} \tag{12}$$

[1] This facilitates comparison of methods. More consistent here and in the next method would be to interpolate on data, then apply the transfer functions.

	Exponential		Average C*D	
	Intensity	Opacity	Intensity	Opacity
1 Cell	91.97	0.500	127.50	0.693
2 Cells	91.97	0.500	105.41	0.573
5 Cells	91.97	0.500	96.72	0.526
10 Cells	91.97	0.500	94.26	0.512
50 Cells	91.97	0.500	92.42	0.502
100 Cells	91.97	0.500	92.19	0.501
10000 Cells	91.97	0.500	91.97	0.500

Table 1: Combining Compositing and Integration

where C_{new} and O_{new} are the color and opacity values of the newly calculated cell at a particular pixel, and C_{acc} and O_{acc} are the accumulated color and opacity values of the pixel projected onto before and after projection. However, it is not necessary to calculate O_{acc} or store it.

For front-to-back traversal,

$$C_{acc} = (1 - O_{acc})C_{new} + C_{acc} \tag{13}$$

$$O_{acc} = (1 - O_{acc})O_{new} + O_{acc} \tag{14}$$

and O_{acc} must be stored.

3 Experimental Results

Coherent projection methods have been explored on a number of data sets. This section will explore: 1) integration approaches; 2) interpolation approaches; 3) compositing; and 4) cost and quality of final images.

3.1 Results Concerning Integration in Depth

Three issues to consider in choosing an integration method are: behavior within a single cell; behavior when compositing many cells; and time costs. After some illustrative examples are used to explore these issues, values are compared for two real data volumes.

Behavior within a Single Cell: Consider integration of one front-back vertex pair. Generally, cumulative intensity and cumulative opacity are higher using average C*D integration, compared to the "exponential" integration methods. The exponential homogeneous approach averages out differences between front and back faces, while the exponential linear approach interpolates. Consider three cases with the same cell depth:

	Front Vertex		Back Vertex	
	opacity	intensity	opacity	intensity
(A)	0.4	100	0.4	100
(B)	0.8	200	0.0	0
(C)	0.0	200	0.8	0

Exponential homogeneous integration will find the same cumulative intensity and opacity for all three because of averaging. Exponential linear integration will not. In fact, as the number of subdivisions increases, cell B will become increasingly dark, because the high opacity at the front will occlude both light from behind and light being emitted within the cell itself. (In cases where opacity is very high, even cells with high material intensity values will become very dark. This is a rather undesirable though not unrealistic property of high opacity regions in this approach.)

If, instead of material opacity, the *differential* opacity were linearly interpolated through the cell in the exponential linear integration method, then cases A and B would give identical results because intensity is a linear multiple of opacity and the opacities have the same integrals, so the discussion following Equation 6 applies. Case C, however, would still differ.

Compositing Between Cells Combined with Integration within Cells: Compositing brings out an inadequacy of Gouraud interpolation methods. Inter-cell compositing is itself an exponential process, comparable to the exponential integration methods described above. Table 1 shows the problem with a simple example: a constant value region with intensity 127.5, material opacity 0.5, and distance 1. (Compositing was done in floating point.) Using exponential integration methods, the same intensity and opacity result from treating the region as one cell or many cells, as should occur in reality. This is not the case when average C*D integration is used.

If a hierarchy or progressive refinement is used, the resultant image will vary in intensity depending upon the number of subdivisions. Intensity variations can also occur when viewing the volume at an angle, because the line of sight rays through pixels pass through different numbers and depths of cells.

Time Costs: The exponential integration methods are clearly more expensive. Considering only the subroutines involved in integration in depth, the average C*D method was from 25% to 50% faster than exponential homogeneous integration, and this latter was about three times as fast as exponential linear integration with five subdivisions. The relative costs of these routines in the whole program depend upon volume size and orientations. On tested volumes (40x32x32 and 256x256x51 resolution), exponential homogeneous integration routines took from 35% to 60% of the total running time.

Integration on Two Larger Volumes: The effect of the three approaches on two real data sets was examined. One was a 256x256x50 section of an MR brain data set.[2] The transfer functions for this were a mostly linear ramp with increased red and opacity in the middle ranges. (See Figure 5). The other volume was the pressure scalar field from a computational fluid dynamics simulation of air flow around a blunt fin.[3] The original curvilinear-gridded data was resampled to a regular grid with dimensions 115x100x51 [WCA+90]. Transfer functions mapped high pressure regions to red and medium pressure regions to blue. (See Figure 2 for a smaller 40x32x32 version of this volume.)

The cumulative intensity and opacity values found by the three integration methods for each front/back vertex pair were compared. In summary, comparing mean values and standard deviations, differences were at most 2%. Occasionally, C*D integration differed by 8%.

3.2 Interpolation Between Projected Vertices

Next, methods for finding cumulative intensity and opacity at pixels were explored (see Section 2.3). When Gouraud interpolation was used, all of the three integration methods were used to find cumulative intensity and opacity at the subcell vertices. When exponential interpolation methods were used, both subcell vertices and interpolated pixels used the same method.

Interpolation within One Cell: When viewing a single cell, all methods appear visually reasonable, though Gouraud-interpolated cells are somewhat darker around the borders when viewed with more than one face visible. This is because the cumulative intensity and opacity at some silhouette vertices is zero, due to the distance between front and back subcells vertices being zero. For example, as we move horizontally across a cell of constant opacity Ω, say from $x = 0$ at a silhouette vertex of cell-depth 0 to an interior vertex at $x = a$, where the cell-depth is d, intensity should increase according to the nonlinear function $(1-e^{-\Omega x d/a})/\Omega$. However, linear interpolation yields only the smaller function $\frac{x}{a}\left(1-e^{-\Omega d}\right)/\Omega$. "Exponential" interpolation methods use the correct nonlinear function for pixel calculations and do not have this problem.

It is possible to somewhat alleviate linear interpolation artifacts and still use hardware Gouraud interpolation through the use of blend functions and multiple blendings. We have implemented a "three-pass" method inspired by a suggestion of Pat Hanrahan and Peter Shirley. It requires back-to-front traversal. The essential step is to compute a "half-way" opacity such that $(1-O_{half})^2 = (1-O_{cum})$. Blending the back twice with this O_{half} causes the hardware (effectively) to multiply two linear interpolations yielding a net quadratic interpolation that more closely approximates the desired exponential function. In the third pass the hardware blends the intensity of the new cell into the background. Additional details are omitted for lack of

[2] MR data was from a Siemens Magneton and provided courtesy of Siemens Medical Systems, Inc., Iselin, MJ. Data edited by Dr. Julian Rosenman, North Carolina Memorial Hospital. Data is from UNC 1989 Volume Visualization Workshop dataset.

[3] CFD data courtesy of NAS/NASA Ames Research Center.

Subcells	1		2		10		50		100	
	Int.	Opac.	Int.	Opac.	Int.	Opac.	Int.	Opac.	Int.	Opac.
Double Real	92	128	92	128	92	128	92	128	92	128
Float Real	92	128	92	128	92	128	92	128	92	128
16-bit Integer	92	128	92	128	92	127	92	127	90	127
8-bit Integer	92	128	92	127	87	124	83	122	1	101
Iris 4D-50GT	92	128	91	127	87	119	34	32	1	1

Table 2: Comparison of Compositing Using Integer or Real Arithmetic (Intensity and Opacity)

space. This method is inaccurate when $\Omega d << 1$ due to limited hardware precision, but may be useful when the scene consists of a few large cells, as shown in Figure 3.B.

Interpolation with Layers of Cells: Problems with Gouraud interpolation methods become more obvious when layers of cells are composited. The inaccurate interaction of integration and compositing described in Section 3.1 will occur when linear methods are used, because the integration method simulated at each pixel is inaccurate. However, this problem is not very obvious unless the volume consists of a few large cells. Figure 3 illustrates this effect on a 4x4x4 constant value volume rendered using four methods: the upper left image used exponential homogeneous integration at subcell vertices and hardware Gouraud interpolation; the upper right used the same methods but with the multiple blending mentioned above; the lower left used average C*D integration at subcells vertices and hardware Gouraud interpolation; and the lower right used exponential homogeneous integration and interpolation. It is possible to see some artifacts between cells in all images except the lower right.

Precision Problems with Hardware Shading: A much greater problem may occur when using hardware Gouraud shading as opposed to software Gouraud shading, because if the individual cells of the volume are *very dim*, precision and truncation errors become a major consideration. Figure 4 shows the problems due to hardware interpolation that show up when many layers are composited together. These images show a 20x20x20 scalar field with material opacity values decreasing radially from the center. Left images use exponential homogeneous integration at subcell vertices and right images uses average C*D integration at subcell vertices. Upper images use hardware Gouraud shading and lower images use software Gouraud shading. All images use hardware compositing, so the compositing behavior itself is not the problem here. Rather the accumulation of tiny (one bit) errors from each dim cell layer (average cumulative intensity was 7 and average opacity 0.1) when many layers are composited together produce this result. Further, the blocky nature of the pictures is also seen on a single layer of cells when the monitor brightness is enhanced. These problems become much less noticeable if cells are bright, because truncation is less significant.

3.3 Compositing

A related problem can occur due to hardware compositing that uses only eight bits for the opacity channel. To explore this, a single cell of homogeneous material intensity and opacity (both 128 from a maximum of 255) was used. As described in Section 2.3, using exponential methods, integration in depth through this cell should provide the same intensity and opacity whether it is treated as one or many cells. This is the case using floating point arithmetic. Compositing in software on a Sun 4 provides the results in Table 2 for double- and single-precision floating point, eight-bit integer, and sixteen-bit integer arithmetic. The cell depth is 1 and the cell is divided into 1, 2, 10, 50, and 100 subcells. The final row shows results gained from compositing on an Iris 4D-50GT and reading the frame buffer.

Of course, most graphics systems are designed to render fairly bright objects and composite a few layers. This was what they were designed for, and they are very good at it. However, some of us insist on trying to extend their uses in new directions, and encounter the problems described above. For us, it would certainly be desirable that the machines use more bits per pixel for interpolation and compositing.

3.4 Timings and Image Quality

In this section the various approaches are compared in terms of the images produced and time taken on few data sets (See Table 3). The machine used for timings and still images was a Silicon Graphics Iris 4D50-GT, and images for timings were drawn into a 500x500 pixel window.

As a brief comment, coherent projection takes advantage of the identical parallel projection of all cells. If perspective projection were used, each cell would have to be independently projected. This would approximately double the cost of rendering when using hardware Gouraud interpolation.

Considering subcell vertex integration methods, average C*D integration was only slightly less expensive than exponential homogeneous integration considering the overall time

Integration Method Interpolation Method	Pixel Coverage	Ave. C*D Hrd.Gour.	Exp.Homog. Hrd.Gour.	Exp.Homog. Sft.Gour.	Exp.Homog. Exp.Homog.	Exp.Lin.(5) Exp.Lin.(5)
Sphere 20x20x20						
Front	13x13	3	4	48	103	359
2xZoom	25x25	3	4	155	373	1395
Rot. 30,30,30		5	6	96	193	623
Blunt Fin 40x32x32						
Front	7x8	16	21	62	90	-
2xZoom	13x16	16	21	138	274	-
Rot. 30,30,30		27	32	132	177	-
SOD 97x97x116						
Front	3x3	495	542	1095	-	-
2xZoom	5x5	516	545	1771	-	-
Rot. 30,30,30		830	833	2161	-	-

Table 3: Timings on Projection Methods (seconds)

of image creation. This is particularly true on large volumes with small dim cells, where the integration approximates by multiplying color times distance. Exponential linear integration did incur a large time cost and provided minimal differences in image quality compared to exponential homogeneous integration on most volumes viewed anyway.

Considering methods of determining pixel values, software Gouraud interpolation was considerably more expensive than hardware Gouraud interpolation, though how much is dependent upon the cell size. The extra cost of software interpolation decreases with cell size. Also, when cells cover only a few pixels, artifacts due to hardware interpolation also become less significant. But then, advantages of any type of coherent projection disappear when cells are only a pixel or a few pixels in area. A considerable advantage of hardware interpolation is seen when volumes are zoomed. Hardware interpolation is little affected by cell size, while software methods become far more expensive. Applying either of the exponential integration methods at each pixel was prohibitively expensive for medium to large volumes. When a few large cells are visible, however, these methods do produce better images.

The figures show some illustrative images done using exponential homogeneous integration at polygon vertices and hardware Gouraud interpolation. Figure 2 is a 40x32x32 resampled volume based on a CFD simulation of the blunt fin. Figure 5 shows two views of the MR Brain data set. Both volumes were cited in Section 3.1. Figure 6 shows part (zoomed) of the SOD enzyme volume (resolution 97x97x116).[4] Figure 7 is an image of a sampled function, which shows the present implementation's rudimentary isosurface extraction (green) and gradient shading (red and blue) abilities.

[4]Electron density map of active site of superoxide dismutase [SOD] enzyme as determined by X-ray crystallography, Duncan McRee, Scripps Clinic, La Jolla, California. Data is from UNC 1989 Volume Visualization Workshop datset.

4 Future Directions

There are several improvements that can be implemented. Exponential linear integration is important only when a large variation exists between front and back vertex values. To avoid the expense of the calculation when unnecessary, the system could compare values and choose between the two exponential methods based on their variation.

The advantages of projection disappear when cells are very small. When this occurs, it would be intelligent to automatically reduce the calculations on each cell, perhaps treating each as one projected polygon with values that are an average of corner vertices.

When imaging front to back, it would be useful to stop rendering cells that project to already opaque regions. The implementation for this is preliminary.

Most images shown here assume the only light is that emitted by the volume. Better gradient shading and isosurface extraction should be implemented. Preliminary tests suggest this should be feasible.

It has become obvious in exploring volume rendering on real data sets that speed is only a small part of the problem. The laborious process of finding an appropriate mapping from data values to color and opacity deserves much future attention. The ability to shade according to gradients and isolate surfaces would also improve the program's usefulness.

5 Conclusions

Coherent projection offers rapid imaging, good quality images, and the ability to watch the whole image appear in layers as it is rendered. It is most advantageous when cells cover a number of pixels, as with medium-sized volumes or large volumes zoomed.

The images produced by the various methods explored are generally close as long as cells are the same size. Indeed, particularly when cells are fairly small, it is remarkable how little difference even the inexpensive simple average method makes in the final image. The choice of integration and interpolation should be based on the tradeoffs of quality versus speed. For initial exploration and manipulation of a volume, the cheapest possible method might be desirable. When a desired view is found, more accurate renderings using exponential methods and software interpolation can be used.

In considering the varied approaches to direct volume rendering and the cost involved, it is worth keeping in mind the level of abstraction with which we are often dealing. The model of a semi-luminous material is itself generally an abstraction with no reality: what is the luminosity of a pressure field? The transfer functions are completely under user control: how transparent is bone? The underlying function is often not known: is a linear variation at all accurate? The method that conveys desired information to the viewer without grossly ignoring or adding volume contents and works quickly enough to encourage its use will be most desirable.

Acknowledgements

Particular thanks go to Pat Hanrahan, Nelson Max, Marc Levoy, and Peter Shirley for useful discussions, to the careful reviewers whose suggestions have hopefully clarified the paper, and to Sam Uselton for his initial user-interface. Funds for the support of this study have been allocated by the NASA-Ames Research Center, Moffett Field, California, under Interchange No. NCA2-430.

Figure 2: A 40x32x32 Blunt Fin

References

[Bli82] Jim F. Blinn. Light reflection functions for simulation for clouds and dusty surfaces. *Computer Graphics*, 16(3), July 1982.

[DCH88] Robert A. Drebin, Loren Carpenter, and Pat Hanrahan. Volume rendering. *Computer Graphics (ACM Siggraph Proceedings)*, 22(4):65–74, July 1988.

[Gou71] H. Gouraud. Continuous shading of curved surfaces. *IEEE Transactions on Computer*, 20(6):623–628, 1971.

[KH84] James T. Kajiya and B. P. Von Herzen. Ray tracing volume densities. *Computer Graphics*, 18(4):165–174, July 1984.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics (ACM Siggraph Proceedings)*, 21(4):163–169, July 1987.

[Lev88] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, March 1988.

[Max86] Nelson Max. Light diffusion through clouds and haze. *Computer Vision, Graphics, and Image Processing*, 33:280–292, 1986.

[MHC90] Nelson Max, Pat Hanrahan, and Roger Crawfis. Area and volume coherence for efficient visualization of 3d scalar functions. *Computer Graphics (ACM Workshop on Volume Visualization)*, 24(5):27–33, December 1990.

[PD84] Thomas Porter and Tom Duff. Compositing digital images. *Computer Graphics*, 18(3):253–260, July 1984.

[Sab88] Paolo Sabella. A rendering algorithm for visualizing 3D scalar fields. *Computer Graphics*, 22(4):51–58, July 1988.

[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.

[WCA+90] Jane Wilhelms, Judy Challinger, Naim Alper, Shankar Ramamoorthy, and Arsi Vaziri. Direct volume rendering of curvilinear volumes. *Computer Graphics*, 24(5):41–47, December 1990. Special Issue on San Diego Workshop on Volume Visualization.

[Wes90] Lee Westover. Footprint evaluation for volume rendering. *Computer Graphics (ACM Siggraph Proceedings)*, 24(4):367–76, August 1990.

Figure 3: A 4x4x4 Constant Value Volume
(Integration at Vertex/Interpolation at Pixel)
A. Exp.Hom./Hard.Gour. B. With Multiple Blends
C. C*D/Hard.Gour. D. Exp.Hom./Exp.Hom.

Figure 5: MR Brain

Figure 6: SOD Enzyme

Figure 4: A 20x20x20 Radially Decreasing Volume
(Integration at Vertex/Interpolation at Pixel)
A. Exp.Hom./Hard.Gour. B. C*D/Hard.Gour.
C. Exp.Hom./Soft.Gour. D. C*D/Soft.Gour.

Figure 7: Abstract