Interactive Volume Rendering

Lee Westover

Department of Computer Science
The University of North Carolina at Chapel Hill
Chapel Hill, North Carolina, 27599-3175
USA

ABSTRACT

Volume rendering is the display of data sampled in three dimensions. Traditionally, visualization of such data has been through conventional computer graphics line or surface drawing methods preceded by processes that coerce the sampled data into a form suitable for display. This approach is being replaced by new techniques which operate directly on the three dimensional samples to avoid the artifacts introduced by the use of conventional graphics primitives.

Volume rendering is a compute-intensive operation. This paper discusses an approach for volume rendering in which interactive speed is achieved through a parallelizable forward mapping algorithm, successive refinement, table driven mappings for shading and filtering, and the avoidance of complex machine classification of the data.

Since the renderer is interactive, users are able to specify application specific mapping functions on-the-fly. Current applications include molecular modeling, geology, computed tomography, and astronomy.

KEYWORDS: 3D Image, Volume Rendering, Algorithms.

INTRODUCTION

A common data type in scientific computing is a regular three-dimensional grid of sample points. A single data point may be a scalar, as in electron density maps, or a vector, as in the results of fluid flow simulation. Other examples include seismic studies for oil exploration, light wavelength data for galaxies, and stacks of computed tomography scans for medical imaging. Direct display of volume data is called volume rendering. Current techniques are too slow to be interactive because renderers either tri-linearly interpolate many points along each sight ray or perform complicated polygon fitting for each point neighborhood. Since typical data range from 64 by 64 by 64 data points to 256 by 256 by 256 data points, the sheer number of samples amplifies any inefficiency of the display algorithm. In a non-interactive mode a user guesses at viewing and shading parameters and renders an image that can take anywhere from 5 minutes to many hours to compute. When the image is completed, the user may change the input parameters and try again, iterating until a satisfactory image is generated. While this batch method of volume rendering is satisfactory for final presentation image generation, it does not lend itself to data exploration. The length of time between iterations makes experimentation painful and breaks idea continuity.

The goal of this work is to design a system for interactive exploration of volume data with enough flexibility to encourage the user to try numerous and possibly unusual mappings. The renderer uses only table driven interpretation and classification so the user can easily understand how the data is being mapped to the final image. The user must be able to control each and every step in the generation process. When the user changes an input viewing parameter, he should immediately (with in 10 seconds) see the change. The altered image may not be the final image, but it should be adequate for the user to quickly steer through his data [Brooks 86] [Greenberg 86].

PREVIOUS WORK

Early attempts to visualize volume data often failed because the large data size and the massive amounts of calculations needed. Because of the availability of polygon renderers and line drawing displays, the volume rendering problem was often coerced into one of these problems by preprocessing the data. Contour maps would display a line drawing connection of equal valued data points [Wright 72] [Williams 82]. Alternatively, these contours were triangulated to form polygons that

were then fed into polygon engines [Fuchs 77] [Ganapathy 82]. Other algorithms were developed for special classes of point data such as the rings of Saturn [Blinn 82], clouds [Kajiya 84], meteorological data [Hibbard 86], and points as primitives [Reeves 83] [Levoy 85].

Recently, the advances in machine speed and memory capacity have allowed researchers to eliminate the intermediate surface model and directly display volume data. These approaches typically fall into one of two categories, backward mapping and forward mapping.

Backward mapping algorithms are those algorithms that map the image plane into the data, commonly called ray tracing. For each pixel in the final image, the renderer shoots rays from the pixel into the data and intersects that ray with each data point until either the ray exits the volume or the opacity accumulates enough density to become opaque [VanHook 86] [Levoy 88] [Sabella 88] [Upson 88].

Forward mapping algorithms are those algorithms that directly map the data onto the image plane. Examples in surface graphics include the Z-buffer and the painter's algorithm. For each data point, the renderer maps the point onto the image plane and then adds its contribution to the accumulating image. This accumulation can be either back-to-front or front-to-back. The image is complete when each data point is added to the screen or the image becomes opaque and new samples can have no further effect on the final result [Drebin 88] [Upson 88].

DESIGN TRADEOFFS

The main goal of this work is to find an algorithm suitable for interactive volume rendering. A parallel algorithm is desired because of the large amount of computation required in the volume rendering process. Similarly, as much of the rendering process as possible should be table driven.

For an arbitrary view, a naive parallel backward mapping algorithm requires that the entire data volume be replicated at each parallel computation node. More sophisticated methods may relieve some of this replication but will not eliminate it. Since a forward mapping algorithm can treat each data point in isolation, there is no need to replicate any of the input volume in a parallel system.

There are two places where discrete samples may be reconstructed in the volume rendering process. First, the renderer does volume space reconstruction where the three-dimensional individual input samples are reconstructed into a three-dimensional continuous function. Second, the renderer does image space reconstruction where individual image space intermediate samples are reconstructed into viewable samples that make the final image.

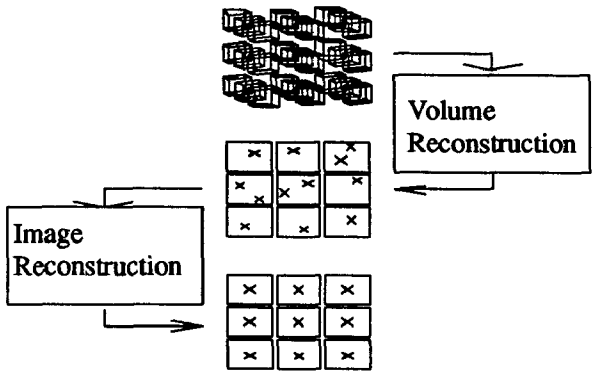


Figure 1. Reconstruction

In backward mapping algorithms input values rarely fall exactly along a ray. Consequently, a continuous approximation of the volume data is generated near a sample point using tri-linear interpolation. Image reconstruction is required when the intermediate samples do not fall exactly on output pixel centers. This occurs when anti-aliasing is done by uniform or stochastic supersampling. (If all rays originate at exact pixel centers, the second reconstruction step becomes the identity mapping.) Anti-aliasing is required in a backward mapping algorithm because tri-linear interpolation is a poor reconstruction kernel [Levoy 88].

A forward mapping algorithm can perform the volume space reconstruction in two-space because the data points themselves are fed through the rendering pipeline and only an image space footprint of the data point need be considered. Furthermore, the image space reconstruction is not required because the footprint function is a continuous function and needs only to be sampled. Detail of this process appear later in the paper.

A problem for both forward and backward mapping algorithms is perspective. For a perspective view, the sampling rate of the input data with respect to the screen changes with depth. However, orthographic views of volume data are useful in their own right and often desired. Since perspective views are not critical for a wide range of applications, the renderer described in this paper only generates orthographic views at this time.

For maximum speed the renderer uses table driven operations as often as possible. For example, the reconstruction process is driven by filter tables and the shading process is driven by four shading tables. The renderer should not make any binary classification or shading decisions. All decisions should be probabilistic or weighted [Levoy 88] [Drebin 88]. The use of shading tables does not enforce this rule but certainly supports it.

Since a forward mapping algorithm allows parallel implementation without input data replication, allows volume reconstruction to occur in two-dimensional image space, and provides speed and flexibility through table driven shading and reconstruction, the renderer presented here uses a table driven forward mapping algorithm.

ALGORITHM

The algorithm consists of three main parts: the viewing transformation, signal reconstruction, and converting an input sample into a shaded intermediate sample.

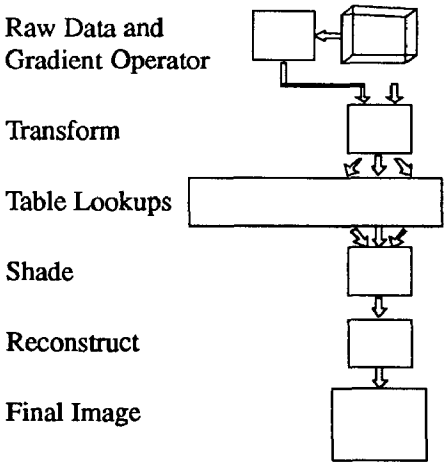


Figure 2. Block Diagram

Pipeline Structure

The path through the rendering pipeline is to take an input sample packet,

$$\begin{bmatrix} \textit{density value} \\ \textit{gradient strength} \\ \textit{gradient <i, j, k> direction} \\ \textit{grid <i, j, k>} \end{bmatrix}$$

perform the grid space to screen space mapping which converts the grid <i, j, k> into screen <x, y, z>, and forms a packet that consists of:

$$\begin{bmatrix} \textit{density value} \\ \textit{gradient strength} \\ \textit{gradient <i, j, k> direction} \\ \textit{screen <x, y, z>} \end{bmatrix}$$

The packet is then shaded, converting the density value and gradient information into red, green, blue and alpha values forming a packet that consists of:

$$\begin{bmatrix} \textit{red} \\ \textit{green} \\ \textit{blue} \\ \textit{alpha} \\ \textit{screen <x, y, z>} \end{bmatrix}$$

This packet is then passed through a reconstruction step and combined into the image buffer. When all input samples have been processed the image is complete.

The following sections describe these processing steps in detail.

TRANSFORMATIONS AND RECONSTRUCTION

Screen Mapping

Since the input volume is a regular three dimensional grid and the renderer only generates orthographic views, a digital differential analyzer (DDA) can incrementally map each data point from grid space to the screen space.

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} dx/di & dx/dj & dx/dk \\ dy/di & dy/dj & dy/dk \\ dz/di & dz/dj & dz/dk \end{bmatrix} \begin{bmatrix} \Delta i \\ \Delta j \\ \Delta k \end{bmatrix}$$

where dA/dB denotes the change in the A direction in image space for each step in the B direction in grid space.

Thus, the step sizes in each of x, y, and z for each input i, j, and k can be read directly from the transformation matrix since it is made up of only rotations and scaling. By inspecting the sign and magnitude of these values, the renderer can determine a traversal that guarantees a back-to-front ordering or a front-to-back ordering. Each ordering has its advantages. The back-to-front ordering allows the user to watch the image as it is formed and see features which later may be obscured. Once these deltas are known and the ordering is known, the first input point is transformed with a full matrix multiplication. This is the origin for the DDA. The renderer then just adds the appropriate delta as it walks through the input volume. The image <x, y, z> for each intermediate sample are used to build the image at the end of the rendering pipeline.

Reconstruction

There is one stage in the volume rendering process where the renderer must reconstruct a continuous signal from discrete samples. Image plane samples are generated by sampling the discrete input volume. Since resampling a sampled signal is undefined, a continuous signal needs to be reconstructed from the original data samples prior to resampling. The sampled signal can be thought of the product of a comb function and some continuous signal. The ideal way to reconstruct such a signal is by convolving a sinc function with the sampled data. For a band limited original signal, the convolution can exactly reconstruct the original signal. Similar to the way surface graphics image reconstruction can be either pixel driven or polygon fragment driven [Carpenter 84] [Abram 85], volume graphics reconstruction can either be pixel driven or data sample driven.

A backward mapping algorithm is pixel driven and can have problems with the volume signal reconstruction. The volume convolution between a three-dimensional sinc function and the input volume is an expensive operation. For speed considerations, most forward mapping algorithms use tri-linear interpolation to generate points that lie exactly along a ray. However tri-linear interpolation is an ineffective reconstruction kernel which necessitates over sampling and a second low pass filtering step of the generated samples.

Instead of trying to determine which part of the input volume affects a given pixel, the forward mapping algorithm turns the problem "inside out" and determines what output pixels a given sample can effect. Since the data points themselves are the input samples there is no need to generate interpolated values. This does not alleviate the need for volume reconstruction, it just moves the problem to a later stage of the rendering pipeline when a sample is added to the final image.

Reconstruction is the process of convolving the reconstruction kernel with the sampled signal. The volume reconstruction equation is:

$$signal_{3D} =$$

$$\iiint h_V(u-x,v-y,w-z) \sum \delta(x,y,z)\, \rho(x,y,z)\, du dv dw$$

where $h_V()$ denotes the volume reconstruction kernel, ρ denotes the density function, and $\sum\delta$ denotes the comb function.

Moving the summation outside the integral and evaluating the integral at point <x, y, z> results in:

$$signal_{3D}(x,y,z) = \sum_{i \in Vol} h_V\,(i_x-x,i_y-y,i_z-z)\, \rho\,(i)$$

where i ranges over the input samples that lie within the kernel centered at <x, y, z>.

Instead of considering how multiple samples effect a point, consider how a sample can affect many points in space. The effect at a point <x, y, z> by a data sample <i> is:

$$effect_i(x,y,z) = \rho\,(i)\, h_V(i_x-x,i_y-y,i_z-z)$$

Therefore, the renderer can treat each data sample individually and spread its effect to the output pixels.

Combining

Visibility in and intensity from density functions is often modeled by a scattering equation which integrates brightness along the view direction [Blinn 82] [Kajiya 84] [Sabella 88]. An effective approximation to a scattering equation is to use image composition functions [Porter 84]. Since the composition functions require discrete layers, the renderer needs to integrate each sample along the view direction. Therefore the sample is projected onto the image plane. Projecting the sample onto the image plane at pixel <x, y> is:

$$effect_i(x,y) = \int_{-\infty}^{\infty} h_V(i_x-x,i_y-y,w)\, \rho\,(i)\, dw$$

Since ρ is independent of z, ρ can be moved outside the integral:

$$effect_i(x,y) = \rho\,(i) \int_{-\infty}^{\infty} h_V(x-i_x,y-i_y,w)\, dw$$

Notice that the integral is independent of the sample's density. Since it only depends on the sample's <x, y> projected location, the function footprint can be defined as follows:

$$footprint\,(x,y) = \int_{-\infty}^{\infty} h_V(x,y,w)\, dw$$

where <x,y> denotes the displacement of an image sample from the center of the volume reconstruction kernel's <x, y> projection.

Now the renderer adds the point to the image buffer. Since the footprint function is continuous in <x, y>, there is no need to reconstruct this function. It only needs to be sampled at pixel centers to determine the footprint's contribution to a pixel,

$$weight\,(u,v)_P = footprint\,(P_x-u,P_y-v)$$

where <u,v> denotes the <x, y> of a sample's image plane projection and *P* denotes the pixel in question.

If the three-dimensional volume kernel is rotationally symmetric, the footprint function can be precomputed for all viewpoints and stored in a table in a preprocessing step. If not, the footprint function can be computed once per view and stored in a table. In either case the table is indexed by the fractional offset of a sample's projection from the pixel center and returns the weight for each pixel in the neighborhood of a sample's projection. The sample's <red, green, blue, alpha> (determined by the shader) is then weighted by the table value and added to the image buffer. The process of table lookup, weighting, and combining is called *splatting*.

Once the point's image plane footprint is determined, it is added to the image buffer. The combining rules are different for a front-to-back and a back-to-front traversal, but these rules are equivalent (in the absence of roundoff errors).

For a front-to-back traversal the formula are:

$$I_o = I_c + ((1-A_c) * (I_n * A_n))$$

$$A_o = A_c + ((1-A_c) * A_n)$$

For a back-to-front traversal the formula are:

$$I_o = ((1-A_n)*I_c) + (I_n*A_n)$$

$$A_o = ((1-A_n)*A_c) + A_n$$

where I denotes the intensity, A denotes the opacity, o denotes the output, c denotes what is already in the image buffer, and n denotes the new point.

SHADING

Provided the renderer's shader uses only information that is either part of the sample item or can be generated once in a preprocessing step, any shader can fit in the forward mapping algorithm. For speed considerations, the renderer in the current system uses a table driven shader that is made up of four parts: emittance, diffuse reflection, specular reflection, and opacity calculations. Conceptually a shaded object is a reflective light emitting semi-transparent blob. This shader requires that the density, gradient strength and gradient direction be known for each input sample.

Since the gradient operator requires knowledge of neighboring samples, gradients are generated in a preprocessing step. The result of the preprocessing step is a 32 bit packet made up of 8 bits for density, 8 bits for gradient strength, and 5 bits each for gradient x, y, and z direction (there is 1 bit leftover). While many gradient operators are used in similar applications [Horn 81] [VanHook 86] [Drebin 88] the renderer uses the following:

$$gradient_i(i,j,k) = data(i+1,j,k) - data(i-1,j,k)$$

$$gradient_j(i,j,k) = data(i,j+1,k) - data(i,j-1,k)$$

$$gradient_k(i,j,k) = data(i,j,k+1) - data(i,j,k-1)$$

The shading model uses four tables: a table to determine emitted color, a table to determine reflected color, a table to determine opacity and a table to modulate the opacity. Each table has 256 entries and the table can be indexed by any value available in a sample packet. The indices select the corresponding shading value in the following shading rules. In addition, the emitted, diffuse, or specular component can be set to zero and the modulation component can be set to one, effectively turning off that component.

The emittance rule for shading is:

$$I_{emit} = Table_{emit}[index_{emit}]$$

The diffuse rule for shading is:

$$I_{diff} = Table_{refl}[index_{refl}] * DOT(L,G)$$

The specular rule for shading is:

$$I_{spec} = Table_{refl}[index_{refl}] * DOT(H,G)^n$$

The opacity rule for shading is:

$$A_{rslt} = Table_{opac}[index_{opac}] * Table_{modu}[index_{modu}]$$

The final intensity is:

$$I_{rslt} = I_{emit} + I_{diff} + I_{spec}$$

where I denotes the intensity, A denotes the opacity, L denotes the light vector, G denotes the gradient direction, H denotes the vector half way between the eye vector and the light vector, and n denotes the specular power. Intensity has three components: red, green and blue. Each intensity component is clamped to fall between 0 and 255.

Since the color specified in the tables is the color for a fully opaque sample, the color is attenuated by the opacity value which occurs during the combining stage.

An example of a use for the opacity modulation table is surface enhancement. If the table is loaded with a ramp and the gradient strength is used to select a value, the effect is to increase the opacity of samples that lie between two samples that are drastically different, thus bringing out pseudo surfaces. Another example is to index the opacity modulation table with the packet's z value for pseudo depth queuing. Other interesting effects are achieved by selecting emitted and reflected color with different packet elements. For example, choosing emitted light based on gradient strength and reflected light based on density value is a useful way to view molecular electron density volumes.

INTERACTION

User Interface

The user has many interactive controls for image generation.

For shading, the user selects the emittance and reflectance tables which are full color tables and opacity and the modulation tables which are scalar tables. The contents of the four tables are displayed on the user's screen. The user also has control over which parts of the sample packet are used to index each table by a shading cross-bar selector. For lighting, the user has the option of having the light direction fixed to the world coordinate system or fixed to the grid coordinate system.

For viewing parameters, the user uses a virtual track-ball to select the view direction. In addition, the user selects the high and low <i, j, k> bounds for his data. This allows clipping in grid space. The user specifies which grid <i, j, k> appears in the center of the final image.

The user can also zoom into or out of the data. The user selects whether to generate the images in a back-to-front or a front-to-back ordering.

Successive Refinement

A way to improve the update rate of image generation is to display partial images during image generation. This allows the user to view the data with the current parameters as quickly as possible. If the user does not change the viewing parameters, the image continues to improve while the user watches the display [Bergman 86]. The successive refinement takes two forms.

First, the splat buffer is visible to the user at all times, so he can see any partial image. If the view displayed is not to his liking, he can change a view parameter without waiting for the image to complete and image generation starts over.

Second, different features of the rendering pipeline take different amounts of time to compute. The most important item effecting rendering time is the number of sample packets that are passed down the rendering pipeline. Another place to gain speed is to skip formal reconstruction and simply map each output sample packet to its nearest pixel. In addition, the different components of the shading equation take different amounts of time to compute, and can be turned on in succession. Therefore, there are three axes along which to refine an image: input resolution, reconstruction and shading. Currently, while input parameters remain unchanged the renderer starts with a low resolution copy of the input, moves to a middle resolution copy, then moves to the full resolution copy. It then turns on reconstruction. Once this image completes the diffuse and specular part of the shading model are turned on.

The low resolution version of the data is achieved by visiting only every 4th input sample in each grid direction. With a three-dimensional input volume this reduces the number of samples for the pipeline by a factor of 64. The middle resolution version is achieved by visiting every other input sample. This reduces the number of input points by a factor of 8.

PARALLEL IMPLEMENTATION

A parallel version of the forward mapping algorithm has been implemented. The system consists of a client, a splat server, and a set of map/shade servers. The splat server is currently a TAAC-1. It receives shaded packets from the client and reconstructs and combines the packets into the image buffer. The client is currently a SUN-3/180C with 16 MB of main memory. It runs the user interface, controls each map/shade server's actions, collects shaded packets from each map/shade server and down loads these packets into the splat server. The map/shade servers are a run-time configurable collection of SUN-3s and SUN-4s with anywhere from 4 to 32 MB of main memory each. At the command of the client, each map/shade server reads a sub-cube of the input volume off disk. When the client tells each map/shade server to render an image, the map/shade server renders its sub-cube, generating shaded packets that it sends to the client in groups.

The only complicated part of the parallel version of the forward mapping algorithm is that the splat server must guarantee a back-to-front or a front-to-back traversal of the shaded packets from the multiple map/shade servers. This is done with a sorted linked-list of the map/shade server packet buffers.

RESULTS

Timing Tests

Both the single processor version and the parallel version of the algorithm were used to generate six images, one for each step in successive refinement.

IMAGE	STEP IN REFINEMENT
A	low resolution version of the data
B	middle resolution version of the data
C	full resolution version of the data
D	full reconstruction
E	diffuse portion of the shading model
F	specular portion of the shading model

The timing volume data was a 96 by 128 by 113 computed tomography study of a human head. Color plates 1 and 2 contain the six result images. The single Sun-3 was a Sun-3/60M with 8 MB of main memory. The single Sun-4 was a Sun-4/280S with 64 MB of main memory. The single TAAC-1 was in a Sun-3/180 with 16 MB of main memory. The four Sun-3s were all Sun-3/60M with 8 MB of main memory. The four Sun-4s were all Sun-4/280S with 32 MB of main memory. All times are in seconds.

MACHINES	A	B	C	D	E	F
TAAC-1	3	9	65	165	179	182
1 Sun-3	20	60	363	733	843	898
4 Sun-3	3	21	141	172	205	223
1 Sun-4	8	28	181	481	484	522
4 Sun-4	3	11	89	108	112	116

Sample Images

Included are 16 sample images that display the results at each step in the refinement process as well as the output of the algorithm on various applications. On all image, the boundaries of the input volume is displayed as white edges.

DATA	SIZE	DIMENSIONS
CT of head	96 128 113	Three Spatial
P-orbital	64 64 64	Three Spatial
Transfer RNA	64 64 64	Three Spatial
Hemoglobin	64 64 64	Three Spatial
Super-oxide	64 64 64	Three Frequency
Galaxy	100 100 80	Two Spatial and Light Wavelength
Seismic	64 64 64	Three Spatial

Color plate 1 and color plate 2 each contain four images of the computed tomography data set used in the timing tests. The four images of color plate 1 are the first four steps in the refinement process. The upper left image is from the low resolution version of the data. The upper right image is from the middle resolution version of the data. The lower left image is from the full resolution version of the data. The lower right image is from the full resolution version of the data with full reconstruction. The upper two images of color plate 2 are also from the timing tests. The left image is with the diffuse portion of the shading model. The right image is with both the diffuse and specular portion of the shading model. All these images were generated with the same shading tables, chosen to bring out the skin surface. The bottom two images are other views of the same data set. The left image was made with tables that were chosen to display bone. In addition, the reflection table was loaded with green to emphasize the reflected light. The right image has the right part of the head clipped away and was generated with tables that emphasize surfaces by using the opacity modulation table. Note the detail in both the bone surfaces and the skin-to-air interface including the nasal sinuses and throat regions.

Color plate 3 contains four images of electron density data. The upper left image is an image of the p-orbitals of copper chloride. Note the soft bonding between the lobes where the electrons are shared. In the upper right image of transfer RNA, the volume is colored base on gradient strength to bring out the atom boundaries. The lower left image is an image of a synthetic hemoglobin density map. The map was generated from known atom centers, with bonds added to the volume data in a separate process. Since electron density is directly related to atom type and in the synthetic case these density can be determined for each atom type, the shading tables were chosen to color the density by atom type: green for carbon, blue for nitrogen, red for oxygen, and yellow for sulfur. The lower right image is an image of the structure factors for super-oxide dismutase. The points were colored by phase angle and the opacity depends the intensity.

Color plate 4 contains four images: two of a Fabry-Perot map of a galaxy and two of a synthetic seismic study. Both the top images and both bottom images were generated with the same shading parameters and tables. The only difference in the right and left image in each case is a rotation.

ACKNOWLEDGEMENTS

I would like to thank Apple Computer Inc., Schlumberger-Doll Research, and Sun Microsystems for supporting this research. In addition, I would like to thank my advisor, Turner Whitted, for our numerous discussions and his helpful insight. I would also like to thank Mark Harris for his willingness to be a guinea pig on many versions of the system. The seismic data is courtesy of Schlumberger-Doll Research. The super-oxide dismutase data and the copper chloride p-orbitals data courtesy of Michael Pique of the Scripts Institute. The transfer RNA data and the hemoglobin data courtesy of Frank Hage of the Biochemistry Department of the University of North Carolina at Chapel Hill. The computed tomography data courtesy of Radiation Oncology of the the University of North Carolina at Chapel Hill. The Fabry-Perot data courtesy of Gerald Cecil of the Institute for Advanced Study at Princeton.

REFERENCES

Abram, G.D., L.A. Westover, J.T. Whitted, [1985] "Efficient Alias-Free Rendering Using Bit-Masks and Look-Up Tables", Computer Graphics, vol. 19, no. 3, July 1985.

Bergman, L., H. Fuchs, E. Grant, S. Spach, [1986] "Image Rendering by Adaptive Refinement", Computer Graphics, vol. 20, no. 4, August 1986.

Blinn, J.F., [1982] "Light Reflection Functions for Simulation for Clouds and Dusty Surfaces", Computer Graphics, vol. 16, no. 3, July 1982.

Brooks Jr., F.P., [1986] "Interactive Graphics and Supercomputer Debate", 1986 Workshop on Interactive 3D Graphics, October 1986.

Carpenter, L., [1984] "The A-buffer, and Antialiased Hidden Surface Method", Computer Graphics, vol. 18, no. 3, July 1984.

Drebin, R.A., L. Carpenter, P. Hanrahan, [1988] "Volume Rendering", Computer Graphics, vol. 22, no. 4, August 1988.

Fuchs, H., Z.M. Kedem, S.P. Uselton, [1977] "Optimal Surface Reconstruction from Planar Contours", Communication of the ACM, vol. 20, no. 10, October 1977.

Ganapathy, S., T.G. Dennehy, [1982] "A New General Triangulation Method for Planar Contours", Computer Graphics, vol. 16, no. 3, July 1982.

Greenberg, D.P., [1986] "Scientist Wants a Window into the Database", Panel on Graphics, Image

Processing, and Workstations, October, 1986.

Hibbard, W.L., [1986] "4-D Display of Meteorological Data", 1986 Workshop on Interactive 3D Graphics, October 1986.

Horn, B.K.P, [1981] "Hill Shading and the Reflectance Map" Proceedings of the IEEE, vol. 69, no. 1, January 1981.

Kajiya, J.T., B. Von Herzen, [1984] "Ray Tracing Volume Densities", Computer Graphics, vol. 18, no. 3, July 1984.

Lenz, R., B. Gudnumdsson, B. Lindskog, P.E. Danielsson, [1986] "Display of Density Volumes", IEEE Computer Graphics and Applications, vol. 6, no. 7, July 1986.

Levoy, M.S., J.T. Whitted, [1985] "The Use of Points as a Display Primitive", Technical Report #85-022, University of North Carolina, Chapel Hill, NC, 1985.

Levoy, M.S., [1988] "Volume Rendering: Display of Surfaces from Volume Data", IEEE Computer Graphics and Applications, vol. 8, no. 3, May 1988.

Reeves, W.T., [1983] "Particle Systems -- A Technique for Modeling a Class of Fuzzy Objects", Computer Graphics, vol. 17, no. 3, July, 1983.

Porter, T., T Duff, [1984] "Compositing Digital Images" Computer Graphics, vol. 18, no. 3, July 1984.

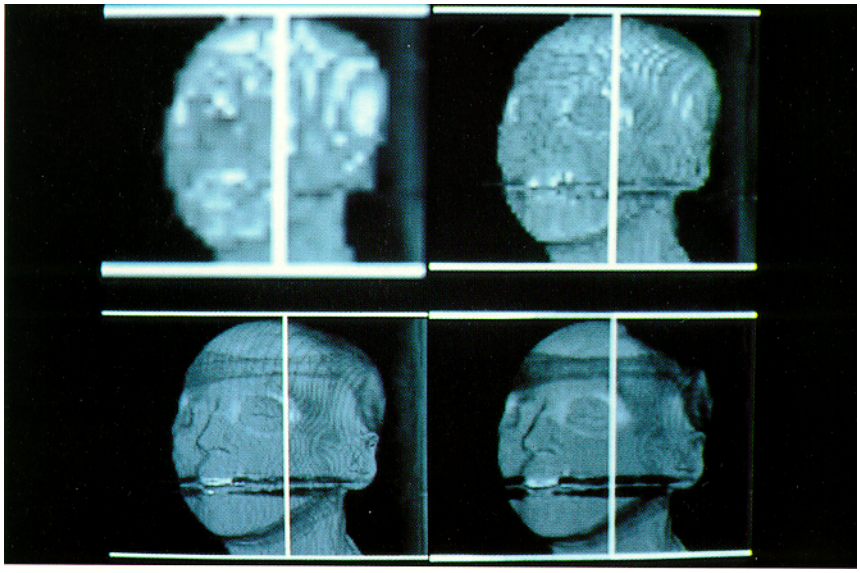
Sabella, P., [1988] "A Rendering Algorithm for Visualizing 3D Scalar Data" Computer Graphics, vol. 22, no. 4, August 1988.

Upson, C., K Keller, [1988] "VBUFFER: Visible Volume Rendering" Computer Graphics, vol. 22, no. 4, August 1988.

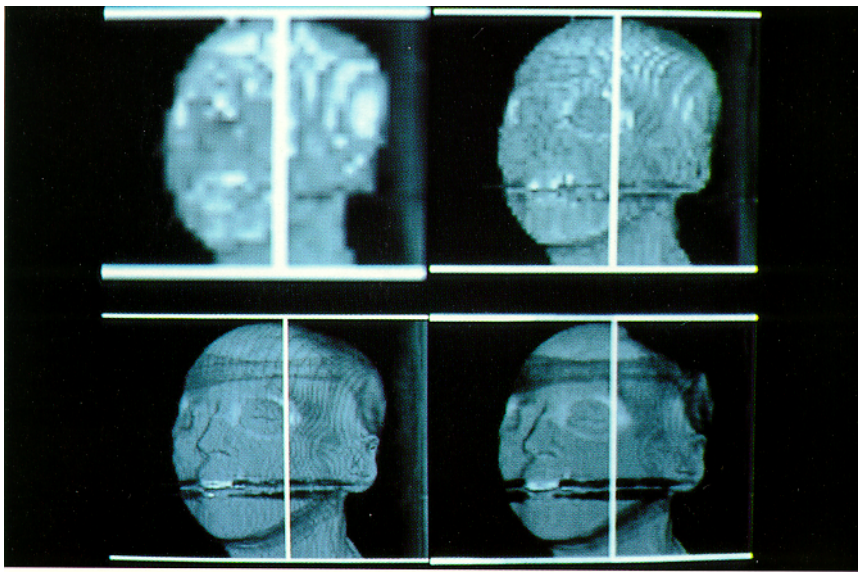
VanHook, T., [1986] Personal Communication. September 1986.

Williams, T.V., [1982] "A Man-Machine Interface for Interpreting Electron Density Maps", Ph.D. dissertation, University of North Carolina, Chapel Hill, NC, 1982.

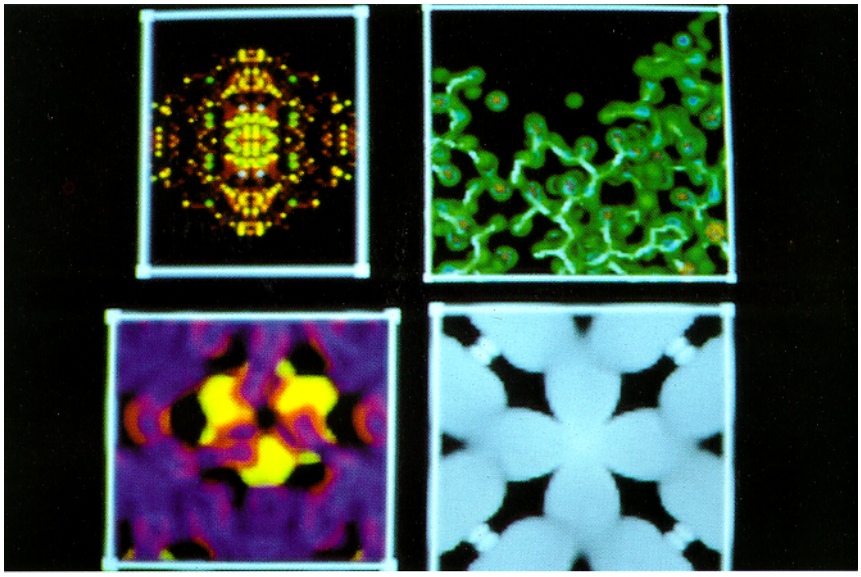
Wright, W.V., [1972] "An Interactive Computer Graphics System for Molecular Studies", Ph.D. dissertation, University of North Carolina, Chapel Hill, NC, 1972.



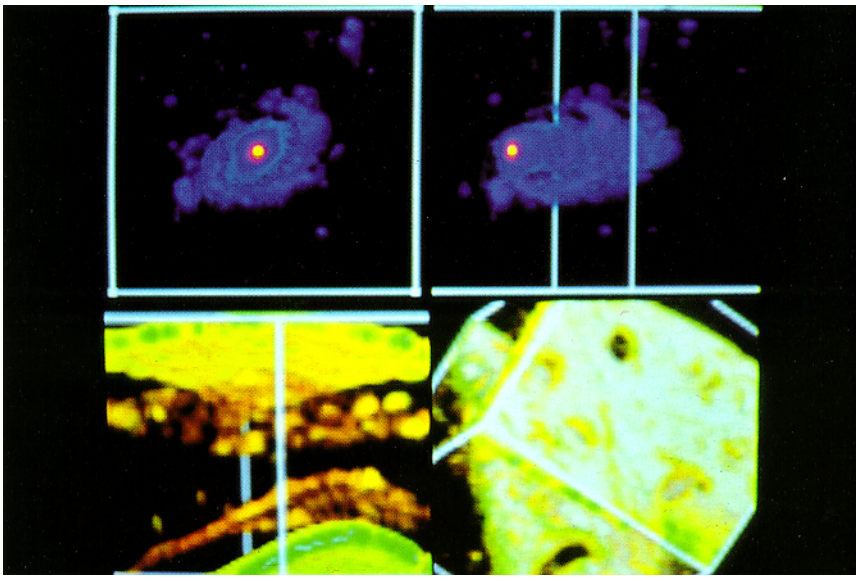
Westover. Color Plate 1.



Westover. Color Plate 2.



Westover. Color Plate 3.



Westover. Color Plate 4.