Direct Volume Rendering on Multicomputers

Dissertation Proposal by
Ulrich Neumann

Department of Computer Science
University of North Carolina, Chapel Hill
August 20, 1991

1. Introduction

Direct volume rendering is the common name that describes the viewing of volume data as a semi-transparent cloud of material. Its advantages are that much or all of the volume may be visible to the observer at one time and there is no need to introduce intermediate geometry that doesn't really exist in the data. We assume the input data is a scalar field sampled at the vertices of a 3D rectilinear lattice, a situation often encountered in medical image and simulation data. *Distributed-data volume rendering* denotes those approaches that distribute the input data array over multiple processing nodes which compute concurrently with the goal of achieving a linear speedup. This research focuses on the efficiency of the direct volume rendering process and its mapping onto parallel systems.

In the interest of space, a representative example of the volume rendering process is presented here without derivation. Many derivations of similar processes have been published [Blinn82], [Kajiya+84]. Much of this example comes from Wilhelms and Gelder [Wilhelms+91]. The input data is assumed band-limited.

1 - Reconstruct the continuous 3D scalar function **F** by convolving each sample point **f** with a reconstruction filter kernel **K**.

$$\mathbf{F}(x,y,z) = \sum_{x,y,z} \mathbf{f}_{x,y,z} * \mathbf{K}$$

2 - Apply an opacity **O** and shading **S** function to the continuous scalar field. These user definable transfer functions yield a differential opacity Ω and color emittance **E** as a function of the scalar field properties. The Ω and **E** fields must be low-pass filtered for resampling in the next step.

$$\Omega = \mathbf{O}(\mathbf{F}) \qquad \mathbf{E} = \mathbf{S}(\mathbf{F})$$

3 - Integrate an intensity function along sample view paths through the volume (Fig. 1). The integrals must be taken monotonically toward or away from the viewer. For the back to front case the accumulated intensity **I** along the sample ray is

$$\mathbf{I}(p) = \mathbf{T}(p)\int_0^p \frac{\mathbf{E}(v)}{\mathbf{T}(v)}dv \qquad \text{where } \mathbf{T}(p) = e^{-\int_0^p \Omega(v)\,dv}$$

The above three steps are approximated in actual implementations. Their accuracy determines the final image quality. To decide which approximations are appropriate to use, we must have a basis for comparison. Image error **Q**, a function of several image error measures described later, is such a basis. An algorithm's efficiency **V** is a function of the computational cost **C** and the image error.

$$\mathbf{V} = 1 / (\mathbf{C}\ \mathbf{Q})$$

For parallel systems, we must account for the costs of communication between nodes as well as computation. The MIMD message passing class of parallel systems is considered here. Communication costs may vary for different routing architectures but in all cases minimizing the required bandwidth is desirable. Communication arises from the need to compute the integral along the view path. The distribution of data among the nodes strongly influences the character of this problem. Each node may integrate its portion(s) of the integral along each path. These fragments are then *composited* in path sequence to produce the final integral. The composited fragments yield exactly the same result as a single full path integration would have. For back to front compositing,

$$\mathbf{I} = (\mathbf{T}_{\text{front}})\, \mathbf{I}_{\text{back}} + \mathbf{I}_{\text{front}}$$

where $\mathbf{I}$ is the result color intensity, $\mathbf{I}_{\text{front}}$ is the intensity of the front fragment being added in, $\mathbf{I}_{\text{back}}$ is the intensity of the back fragment, and $\mathbf{T}_{\text{front}}$ is the transmissivity of the front fragment. Fig. 1(b,c) illustrates the path fragmentation that arises from two different data distributions among four nodes.

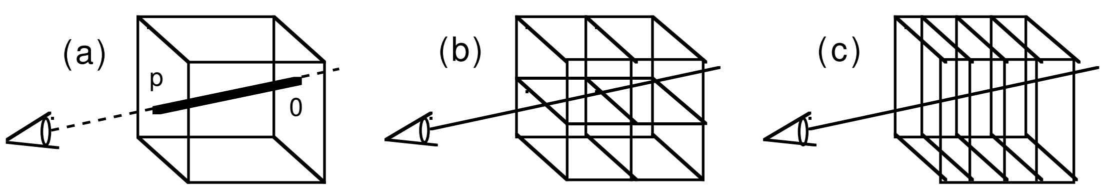

Fig. 1 - Integral path - (a)unfragmented, and fragmentation for (b)block and (c)slab data distributions.

We refer to the communication required to compute the line integrals as *redistribution*. It performs a view dependent screen and depth-order sort of data. Data expands and contracts as it passes through the computation pipeline so the choice of when to redistribute is also important. Redistribution costs are a function of

1 - N, the number of nodes in the system

2 - the data distribution among the nodes

3 - the point in the pipeline at which redistribution occurs

4 - the view parameters.

To evaluate the effects of these parameters, a sequence of views is run through a simulation to determine at each node the number and size of redistribution messages required to generate an image. The results of a sequence of views are actually a vector field over three dimensions - N, node#, and view. It may be useful to view this data as a volume.

2. Thesis

Direct volume rendering algorithms are distinguished by their reconstruction and integration methods. These are the numeric approximations that to a great extent determine the cost and image quality of an algorithm. Parallelizing any algorithm adds the complexities of data distribution and sorting. As a basis for comparison , we can define the efficiency of an algorithm as a function of the computation and communication cost required to achieve a desired image quality. The most efficient algorithm for distributed-data direct volume rendering on MIMD multicomputers will use

1 - a serial-transformation method using a separable reconstruction filter

2 - an exponential homogeneous integration method

2 - a slab data distribution

3 - a "sort-last" redistribution phase.

2.1. Research Summary

1 - Through experimental means I will determine the most efficient reconstruction and integration method among a candidate set. These methods will be compared on the basis of equal image quality. Several analytically generated test data sets will be used - each stressing a different rendering capability. Image error will be computed by comparing the image properties of the test cases and the analytically computed images.

2 - Through simulation the redistribution cost will be characterized for a range of node numbers, data distributions, and view directions. The output data will show the message traffic to and from each node. The goal will be to find the data distribution that minimizes the peak and average node traffic, thus providing an even and minimal communications load among the nodes.

3 - Given the reconstruction and integration method selected and the data distribution results, I will determine the point in the processing pipeline where the data is smallest but still appropriate for redistribution. This approach will then be implemented on at least one parallel system. Performance measurements will be made and compared to expectations.

2.2. Contributions to Knowledge

1 - Quality and cost comparisons between candidate 3D reconstruction and integration methods. This result has equal bearing on uni-processor and parallel implementations.

2 - Redistribution cost analysis for different data distributions and system sizes. This result should have application to any distributed data approach, particularly algorithms for message passing architectures.

3 - Demonstration and characterization of a distributed algorithm on a multicomputer.

3. Related Work

The following sections describe some of the more relevant work that has been done in the areas of parallel volume rendering algorithms, 3D reconstruction, and integration approximations.

3.1. Parallel Algorithms - Most previously published parallel approaches to volume rendering are not designed for general-purpose MIMD multicomputers. They often target special hardware and are demonstrative as opposed to comparative in nature. Drebin uses a four-channel SIMD processor to project a volume by serial transformation and compositing [Drebin+88]. Each channel operates on a separate element of a voxel's data vector and has simultaneous access to a portion of a shared memory [Levinthal+84]. Westover's multiple voxel shaders send their results to a central "splat-server" where they are composited into a final image [Westover89]. The problem of the splat-server becoming a bottleneck as the number of shaders is increased is addressed in later work were he describes a distributed splatting approach where each node splats onto a set of scan lines [Westover91]. Several systems utilize multiple projection processors, each responsible for a sub-block of data [Goldwasser+85], [Yazdy+90]. However, these schemes rely on a single compositing processor which becomes a bottleneck as more projection processors are added. The method outlined by Levoy for Pixel-Planes 5 uses multiple SIMD shaders and MIMD projection processors, all interconnected by a ring network [Levoy89]. I have used the Pixel-Planes 5 SIMD array to generate the splat kernel for the MIMD projection processors [Neumann91]. A data-parallel SIMD approach has been successfully implemented on the Connection Machine CM-2 using serial transformations [Shroder+91]. This algorithm scales well on the CM-2 class of machines but it is limited to affine projections and rotation-only transformations.

3.2. Reconstruction and Integration - Mitchell and Netravali have published a comparative study of 2D separable cubic spline filters [Mitchell+88]. A range of spline parameters were subjectively evaluated. The results of error analysis and use of the derivative to aid reconstruction are presented. Separable 3D cubic spline filters were used by Drebin, Carpenter, and Hanrahan [Drebin+88] in their implementation. Hanrahan has presented a generalized serial transformation method that uses only three passes for a 3D affine transformation [Hanrahan90]. Westover uses a gaussian reconstruction kernel convolved with each data point to approximate reconstruction and integration [Westover89]. Ray casting is often used with reconstruction by trilinear interpolation and integration by sampling along the view ray [Levoy88] [Sabella88] [Upson+88]. A method of projecting rectilinear cells and a comparative study of reconstruction and integration methods was recently published by Wilhelms and Gelder [Wilhelms+91]. This work also showed the effects of varying precision in the compositing process. Cell projection with linear reconstruction and analytic integration was presented by Max, Hanrahan, and Crawfis [Max+90]. Hashimoto did a comparison of some 3D separable cubic reconstruction filters [Hashimoto90]. Some of his work will likely be repeated here.

4. Research Areas and Methods

4.1. Reconstruction Methods -

Most published algorithms have used one of four rendering approaches; each uses reconstruction methods suited to it. These are the reconstruction methods I will compare.

4.1.1. Ray-casting

- Ray-casting a volume involves resampling along lines passing though the volume. The Ω and **E** functions must be reconstructed at the new sample points along the rays. Typically reconstruction is done by trilinear interpolation of function values evaluated at the lattice vertices. The cost for interpolating a scalar value is: 14 adds and 7 multiplies, or 7 adds and 14 multiplies, depending on the procedure used.

4.1.2. Serial Transformations

- An affine view transformation is decomposed into sequential 3D shear operations, each of which is a 1D transformation of the form $x' = Ax + B$. Three sequential 1D linear or cubic reconstructions are performed to facilitate the resampling. The resampled volume is view-aligned and ready for integration. If linear interpolation is used to reconstruct the function for resampling, the cost is 3 multiplies and 6 additions per sample value for a complete transformation. Cubic interpolation increases this cost to 27 multiplies and 18 additions. Cubic interpolation has been used with good results in one implementation [Hanrahan90].

4.1.3. Splatting

- This approach computes the effect of each voxel on the pixels near the point to which it projects. Voxels are sorted by view order and convolved with a filter kernel. The reconstructed function is resampled and accumulated on a view-aligned grid. Since the filter is position-invarient for affine transformations, software lookup table methods approximate the filter function quickly [Westover89].

4.1.4. Cell Projection

- Volumes are decomposed into polyhedra whose vertices are the volume data lattice [Shirley+90] [Max+90] [Wilhelms+91]. The polyhedra are converted to polygons by projecting them under the view transformation. For affine projections all the polyhedra project to geometrically identical polygons. Reconstruction is done to obtain each polygon's vertex values for the opacity and emission functions. The resulting polygons are rendered by conventional means using a painter's algorithm and alpha compositing. These methods are useful because they can use existing polygon rendering hardware. The reconstruction functions are usually linear or bilinear since the hardware does linear interpolation of the polygon vertex values anyway.

4.2. Integration Methods - The intensity function is generally an integral with no closed form solution. Numerical methods will be examined that break the integral path into small segments each of which have constant Ω and E function values. The segment T and I values are then composited together, as previously described, to produce the integral of the entire path. Over each interval the **T** and **I** functions become

$$\mathbf{T}(p) = e^{-\Omega p} \qquad \text{and} \qquad \mathbf{I}(p) = \frac{E}{\Omega}(1 - e^{-\Omega p})$$

There are three basic parameters that may be varied to affect the approximation accuracy: the interval step size of integration, the function which approximates a constant Ω and E over each interval, and the method of approximating the exponential terms. Wilhelms and Gelder [Wilhelms+91] describe several of theses approximations and show some results of their use.

4.2.1. Interval Step Size - As the step size is decreased the approximation becomes more accurate but at the expense of more computation. Typically one step per data sample is taken.

4.2.2. Interval Parameter Function - The Ω and E values used for a segment may be determined by one or more points. Typically one point, or the average of the front and back points of a segment are used.

4.2.3. Exponential Evaluation - Several options are possible for evaluating the exponential terms quickly. We may approximate $(1 - e^{-\Omega p})$ by $\min(1, \Omega p)$. Alternatively, the result of the opacity function Ω may already be considered the exponential. This has the added benefit of limiting the domain of the opacity function to [0, 1]. Lastly, if Ω is encoded as an integer of relatively few bits precision, look up tables can be used to compute the exponential terms.

4.3. Image Comparison - Images made by the reconstruction and integration methods will be compared to an analytically computed *standard* image. Analytically computed images will be possible since the test volumes will be analytically defined. The test volumes will be functions of sine terms to limit the frequency content of the data. The opacity and emittance functions will also be sines to band-limit the shaded volume. Some more interesting volumes or shading functions may not allow closed form integration; those will be integrated with very small steps and true exponential evaluation (see §4.2 above). Test data will contain thin cones and flat surfaces, intersecting and overlaid objects, and opaque and transparent features. Scintillation of small structures and aliasing are often only evident in animation; therefore a sequence of eight views will be computed for all test methods. Since the human eye is sensitive to changes in intensity, the derivative of intensity is the main image property used to compare the test and standard images. The derivative at each pixel will be computed by convolution with a suitable kernel. The absolute value of each pixel is attributed some importance; so the image error is defined as

$$\mathbf{Q} = \mathrm{Avg}\left|i_t - i_s\right| + \mathrm{Max}\left(\left|\frac{\partial i_t}{\partial x} - \frac{\partial i_s}{\partial x}\right| + \left|\frac{\partial i_t}{\partial y} - \frac{\partial i_s}{\partial y}\right|\right)$$

where i is the image pixel value and the subscripts t and s refer to the test and standard image pixels, respectively. To compare rendering methods I will set a threshold value of Q and attempt to vary the parameters of each method such as integration step size and image resampling density, until the test images all fall within tolerance. At that point the computation cost of the test method may be assessed. Costs will be in terms of operations such as additions, subtractions, lookups, and booleans. Operand types (integer, floats, etc) will be considered equivalent. Reasonable attempts at optimization will be made. The cost of the inner loops of reconstruction and integration are all that is of concern. After all methods have been tested, a comparison will reveal the most efficient one.

4.4. Redistribution Cost Analysis - A simulation of redistribution message traffic will be performed. A ray casting algorithm will be used to determine a set of integration paths through the volume. Once these are known for a given data distribution, the message traffic at each node may be determined for any specified integration algorithm. The simulation will be run for a sequence of eight views. The peak and average message statistics will be computed and used as a basis for comparing different data distributions.

4.4.1. Data Distributions - The raw data must be distributed among the memory spaces in the system. Full data replication is a trivial option whose price is deemed too high for large systems, while partial data replication is often necessary or desirable. Data subsets may be *slabs* or *blocks*, *packed* or *interleaved*, and *static* or *dynamic*. The routing patterns that occur during the redistribution process are determined by the data distribution. I have used interleaved slices and packed cubes in the "Splat" and "Vrn" implementations, respectively. Dynamic distributions are possible with "sort-first" redistribution (see §4.5.1). These allow the data to migrate among nodes as needed without occupying a fixed node as its owner. I will consider static packed distributions of slabs and blocks with and without replication of data. If I have time or preliminary results warrant I may look at dynamic distributions as well.

4.5. Redistribution Phase - There are three points in the computation pipeline where redistribution may occur: *sort-first, sort-middle, and sort-last*. The screen and depth sorts may occur in one phase or they may be spread over multiple phases, producing hybrid approaches. The reconstruction and integration method will impact which of these options makes sense.

4.5.1. Sort-First - This approach entails redistributing the raw data before the Ω and **E** functions are evaluated. For static distributions this occurs from the same start point each frame. For dynamic distributions the data incrementally follows its position on the screen.

4.5.2. Sort-Middle - Any approach that sorts an intermediate result would fit into this class. For example, volume data at each node may be shaded to produce <Color, Opacity> tuples which are then redistributed to screen region processors where their projection is computed.

4.5.3. Sort-Last - Redistribution in this case is a depth-order sort to facilitate global compositing of the local images computed by each processing element.

4.6. Implementation and Evaluation - The most efficient technique will be implemented on a MIMD multicomputer with the purpose of verifying the aggregate performance of the constituent pieces. At this time, due to its availability and my familiarity with it, Pixel-Planes 5 is my planned target. Since I would use only the i860 GPs and the frame buffer, both of which are fairly standard, the performance should be representative of other similar machines. The implementation may be limited in terms of shading capability and user interface but it should produce real time renderings equivalent in quality to those produced during tests. The performance will be characterized in terms of the time spent in various portions of the algorithm; eg: reconstruction, integration, shading, redistribution, compositing, and synchronizing. Various machine configurations and data sizes will be tried. It is hoped that the performance will closely track that predicted by the efficiency tests. I anticipate writing most of the code in C with select inner loops written in assembly language for better performance.

5. Schedule

The following is an estimate of time to be spent on various tasks. They will not necessarily be performed in this sequence or in contiguous periods of time.

Proposal acceptance	Summer '91	
Oral exam preparation and further reading	8 weeks	(Fall '91)
Comparison of reconstruction and integration methods	12 weeks	(Fall '91)
Simulation of redistribution costs	8 weeks	(Winter '92)
Implementing and characterizing real-machine algorithm	12 weeks	(Spring '92)
Dissertation writing and illustrations	16 weeks	(Summer '92)
Total	56 weeks	

6. References

[Blinn82] Jim Blinn Light Reflection Functions for Simulation for Clouds and Dusty Surfaces. Proceedings of SIGGRAPH '82, Computer Graphics 16, 3. July 1982, pp. 21 - 29.

[Drebin+88] Robert A. Drebin, Loren Carpenter, and Pat Hanrahan. Volume Rendering. Proceedings of SIGGRAPH '88, Computer Graphics 22, 4. August 1988, pp. 65 - 74.

[Fuchs+89] Henry Fuchs, John Poulton, John Eyles, Trey Greer, Jack Goldfeather, David Ellsworth, Steve Molnar, Greg Turk, Brice Tebbs, Laura Israel. A Heterogeneous Multiprocessor Graphics System Using Processor Enhanced Memories. Proceedings of SIGGRAPH '89, Computer Graphics 23, 4. August 1989, pp. 79 - 88.

[Goldwasser+85] Samual M. Goldwasser, et al. Physician's Workstation with Real-Time Performance. IEEE Computer Graphics & Applications. December 1985, pp. 45 -57.

[Hanrahan90] Pat Hanrahan. Three-Pass Affine Transforms for Volume Rendering. Proceedings of the San Diego Workshop on Volume Visualization, Computer Graphics 24, 5. November 1990, pp. 71 - 78.

[Hashimoto90] Roy T. Hashimoto. A Comparison of Reconstruction Methods for Volume Rendering of Implicit Surfaces. Unpublished paper. May 1990.

[Kajiya+84] Jim Kajiya and Brian Herzen. Ray Tracing Volume Densities. Proceedings of SIGGRAPH '84, Computer Graphics 18, 3. July 1984, pp. 165 - 174.

[Laur+91] David Laur and Pat Hanrahan. Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering. Proceedings of SIGGRAPH ‘91, Computer Graphics 25, 4. July 1991, pp. 285 - 288.

[Levinthal+84] Adam Levinthal and Thomas Porter. Chap - A SIMD Graphics Processor. Proceedings of SIGGRAPH '84, Computer Graphics 18, 3. July 1984, pp. 77 - 82.

[Levoy88] Marc Levoy. Display of Surfaces From Volume Data . IEEE Computer Graphics and Applications 8, 3. March 1988, pp. 29 - 37.

[Levoy89] Marc Levoy. Design for a Real-Time High Quality Volume Rendering Workstation. Conference Proceedings, Chapel Hill Workshop on Volume Visualization. May 1989, pp. 85 - 92.

[Max+90] Nelson Max, Pat Hanrahan, and Roger Crawfis. Area and Volume Coherence for Efficient Visualization of 3D Scalar Functions. Conference Proceedings, San Diego Workshop on Volume Visualization. Computer Graphics 24, 5. November 1990, pp. 27 - 33.

[Mitchell+88] Don Mitchell and Arun Netravali. Reconstruction Filters in Computer Graphics. Proceedings of SIGGRAPH '88, Computer Graphics 22, 4. August 1988, pp. 221 - 228.

[Neumann91] Ulrich Neumann. Accelerating Volume Rendering with a Pixel-Processor Array. To be submitted for publication to The 1992 Symposium on Interactive Real Time Graphics.

[Neumann91b] Ulrich Neumann. Taxonomy and Algorithms for Volume Rendering on Multicomputers. Tech Report TR91-015, Dept. of Computer Science, UNC at Chapel Hill, January 1991.

[Sabella88] *Paolo Sabella. A Rendering Algorithm for Visualizing 3D Scalar Fields. Proceedings of SIGGRAPH '88, Computer Graphics 22, 4. August 1988, pp. 51 - 58.

[Shirley+90] Peter Shirley and Allen Tuchman. A Polygonal Approximation to Direct Scalar Volume Rendering. Conference Proceedings, San Diego Workshop on Volume Visualization. Computer Graphics 24, 5. November 1990, pp. 63 - 70.

[Shroder+91] Peter Shroder and James B. Salem. Fast Rotation of Volume Data on Data Parallel Architectures. Accepted for publication in Visualization'91.

[Upson+88] Craig Upson and Michael Keeler. V-Buffer: Visible Volume Rendering. Proceedings of SIGGRAPH '88, Computer Graphics 22, 4. August 1988, pp. 59 - 64.

[Westover89] Lee Westover. Interactive Volume Rendering. Conference Proceedings, Chapel Hill Workshop on Volume Visualization. May 1989, pp. 9 - 16.

[Westover91] Lee Westover. Splatting a Feed Forward Approach to Volume Rendering. Ph.D. Dissertation. Dept. of Computer Science, UNC at Chapel Hill. TR91-?? July 1991.

[Wilhelms+91] Jane Wilhelms and Alan Van Gelder. A Coherent Projection Approach for Direct Volume Rendering. Proceedings of SIGGRAPH '91, Computer Graphics 25, 4. July 1991, pp. 275 - 284.

[Yazdy+90] Farzad E. Yazdy, et al. CARVUPP: Computer Assisted Radiological Visualization Using Parallel Processing. NATO ASI Series, Vol. F 60. 3-D Imaging in Medicine. June 1990, pp. 363 - 375.