FreeVoxel Attenuation for Order Independent Volume Rendering

Kartic Subr Pablo Diaz-Gutierrez Renato Pajarola M. Gopi

{kartic,pablo,pajarola,gopi}@ics.uci.edu
Department of Computer Science
University of California, Irvine

Abstract

Traditional volume rendering requires back-to-front (or front-to-back) rendering of voxels since the contribution of each voxel is dependent on the contributions of the voxels in front of and behind it along the viewing direction. In this paper, we introduce the concept of FreeVoxels, in which the data required for operations on voxels are pre-computed and stored so that the operations can be performed independently. Specifically we introduce FreeVoxel attenuation to achieve order-independent rendering. The advantage of using FreeVoxels goes beyond involving space-time trade-offs; it introduces incredible flexibility to the process of volume rendering. In splatting-based volume rendering, we show that the FreeVoxel data structure can be used to solve problems like attenuation leakage, that occur due to incorrect blending operations on adjacent voxels. In parallel volume rendering, the notion of FreeVoxels opens new doors by allowing arbitrary static data distribution with no data migration and synchronization-free rendering without compromising on load-balancing. We also describe a hierarchical extension of FreeVoxels that lends itself to multi-resolution rendering.

Keywords: Order Independence, Volume Rendering, Parallel and Distributed Rendering

1 Introduction and Motivation

Research in volume visualization, particularly volume rendering[Brodlie and Wood 2001], has focused on important aspects in the field such as handling large data[Camahort and Chakravarty 1993; Bhaniramka and Demange 2002; Guthe et al. 2002; Guthe et al. 2002; Wilson et al. 2002], improving quality of rendering[Frieder et al. 1985; Drebin et al. 1988; Levoy 1990; Levoy 1992; Williams and Uselton 1996; Dachille et al. 1998; Hadwiger et al. 2003] and improving rendering efficiency [Nieh and Levoy 1992; Totsuka and Levoy 1993; Lacroute and Levoy 1994; Zwicker et al. 2001; Garcia and Shen 2002a].

Parallel volume rendering has enjoyed much attention [Hsu 1993][Li et al. 1997][Garcia and Shen 2002b][Takeuchi et al. 2003], motivated by applications([Park et al. 2001; Nadeau et al. 2002]) dealing with large amounts of data. In [Wittenbrink 1998], it is reported that only few cluster-parallel algorithms have been developed, for ray casting and Fourier domain volume rendering. However, splatting on cluster has been left out so far.

Dependency on order-dependent rendering leads to serious disadvantages in parallel rendering, like lower potential speed-up, data distribution issues and serious synchronization problems. These issues have motivated research in segmentation and data distribution[Hsu 1993; Camahort and Chakravarty 1993], and synchronization. The lower potential speedup is a consequence of the degree of parallelism being affected by imposed order. Thus data need to be distributed carefully amongst the different rendering nodes to minimize communication overhead while rendering overlapping segments; another restriction on the data segmentation is that the segments should be necessarily convex and preferably compact. Despite strategic distribution of data, there is an implicit need for synchronization between the different nodes involved in rendering a frame.

Gao et al.[Gao et al. 2003] describe a method of improving speedup by applying a Plenoptic Opacity Function, computed as preprocess, to perform visibility culling for volume rendering. Their method uses a preprocess to construct the data structure that is used to decide whether a voxel contributes its color to the image or not; however, front-to-back order is imposed on rendering.

In this paper, we introduce the concept of FreeVoxels that enables operations on individual voxels independent of other voxels at the cost of constant amount of extra storage per voxel. From this general concept, we derive FreeVoxel Attenuation that specifically solves the problem of order-independent rendering. Further, in Section 4.2, we extend this concept to accommodate a single pass lighting and rendering algorithm by proposing FreeVoxel Color Attenuation functions. In order to seamlessly integrate FreeVoxel attenuation in multi-resolution volume hierarchies, that are common in handling large data sets, we also introduce a novel attenuation filter to compute the FreeVoxel attenuations for interior nodes in the hierarchy from those of their children.

As a consequence of order independent rendering due to FreeVoxel attenuation, we show that in parallel volume rendering, the constraints on data distribution are eased and there is no need for synchronization within a frame. Further, restrictions on data segmentation are eliminated. The FreeVoxel attenuation can also be used to avoid attenuation leakage problems that arise in traditional splatting-based volume rendering systems due to incorrect blending operations. Thus FreeVoxel attenuation goes beyond the obvious trade-off between memory and speed, and brings in incredible flexibility in data management and elegantly solves other problems as mentioned above.

Main contributions: The following are the main contributions of this paper.

- *Order-independent rendering:* We use FreeVoxel attenuation to achieve order-independent rendering and hence achieve highest potential speedup.
- *Filtering scheme:* We adapt the scheme proposed to cope with multi-resolution volume rendering by defining a filter on the FreeVoxel attenuation that enables the building of a hierarchy.

- *Avoiding attenuation leakage:* We show how the FreeVoxel attenuation can be used to solve the leakage problem in some splatting techniques.

The main concepts involved are presented in Section 2, explanations of algorithms in Section 3, applications of the data structure in 4, and implementation in Section 5.

2 FreeVoxels

FreeVoxels encapsulate all data required by each voxel, to be operated on independently of which the FreeVoxel attenuation enables order-independent rendering. This section introduces the concept of FreeVoxel attenuation and the extension that allows its hierarchical representation.

2.1 FreeVoxel Attenuation

A five-dimensional Plenoptic function[Adelson and Bergen 1991; Gao et al. 2003] $A(x,y,z,\theta,\phi)$ denotes the attenuation of a ray of light along the direction (θ,ϕ) from a 3D point (x,y,z) to infinity. Thus

$$A(x,y,z,\theta,\phi) : \Psi \times \Theta \times \Phi \rightarrow \alpha \tag{1}$$

where Ψ denotes the range of values for the coordinates of a 3D point in space, $\theta \in [-\pi/2, \pi/2]$ and $\phi \in [0, 2\pi]$ denote the direction of light in spherical coordinates and $\alpha = [0,1]$. A zero value for A indicates maximum attenuation; it implies that the point (x,y,z) is totally occluded along the direction (θ,ϕ).

The *FreeVoxel attenuation* is a Plenoptic function defined for a restricted 3D space (Figure 1). For any point P(x,y,z) in the restricted 3D space, FreeVoxel attenuation in the direction (θ,ϕ), is a 2-dimensional function

$$A_P(\theta,\phi) = A(x,y,z,\theta,\phi). \tag{2}$$

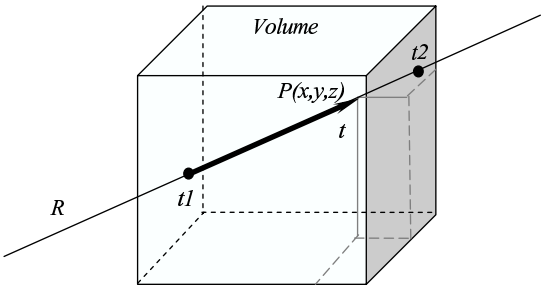

Figure 1: Figure shows a ray R and the interval (thickened) along R where R is attenuated before reaching P

The volume rendering problem involves a transport equation that regulates the interaction of light and matter in a particle model. Traditionally, volume rendering algorithms have approximated this transport equation, along the ray $R(\theta,\phi)$ between ray parameters t_1 and t_2, with an integral equation

$$I_R = \int_{t_1}^{t_2} C(t) e^{-\int_{t_1}^{t} \alpha(s) ds} dt \tag{3}$$

where I_R is the intensity along ray R, *C(t)* is a volume intensity function (that includes emitted, scattered and reflected light) and $\alpha(t)$ is the opacity function. Let t_1 and t_2 describe the entry and exit points of the ray on the boundary of the restricted space. The exponential term is the attenuation of a ray until it reaches the point (with parameter t) on the ray. By definition, the FreeVoxel attenuation is given by

$$A_P(\theta,\phi) = e^{-\int_{t_1}^{t} \alpha(s) ds}. \tag{4}$$

The attenuation along a ray between any two points L and M in the restricted space with parameters t_L and t_M ($t_1 < t_L < t_M < t_2$) can be computed as A_M/A_L.

Employing a zero-order quadrature of the integral along with a first order approximation of the exponential in Equation 4, we get

$$A_P(\theta,\phi) = \prod_{i=1}^{k-1} (1-\alpha_i) \tag{5}$$

where α_i are the opacity values for a discrete set of $(k-1)$ points before P, along the ray.

2.2 Hierarchy and Attenuation Function Filtering

Volume hierarchies are common in multi-resolution rendering. The hierarchy consists of *voxels* at the lowest level and *blocks* at other levels. The attributes of each block are computed from the attributes of its children in the hierarchy. Hierarchical FreeVoxel attenuation can similarly be computed for a block B, given the attenuations of its children in the hierarchy $A_{B1}, A_{B2}, ..., A_{Bn}$.

$$A_B = \Im(A_{B1}, A_{B2}, ..., A_{Bn}) \tag{6}$$

where $\Im$ is a filter on the individual FreeVoxel attenuations.

We consider the level in the hierarchy as the third dimension to equation (4) to yield a hierarchical attenuation function

$$A_P^h(h,\theta,\phi) : H \times \Theta \times \Phi \rightarrow \alpha \tag{7}$$

where H is a set of integers in the range $[0, \lceil log_n N \rceil]$.

To summarize, the method involves computation and then storage of the attenuation functions associated with every voxel/block in each level of the hierarchy.

Clearly the method involves storing large amounts of data that depends on the chosen representation for the attenuation functions. Common representations are approximations using Principal Component Analysis or Spherical Harmonics[MacRobert 1948; Cabral et al. 1987; Sillion et al. 1991]. Section 3.1 details our simple method for storing FreeVoxel attenuation.

3 Algorithms

In this section, we describe the algorithms adopted to construct the FreeVoxel attenuation, hierarchical attenuation filtering and rendering. For the purpose of implementation, without loss of generality, we discretize and compute the FreeVoxel attenuations along a few chosen directions. For other directions, the FreeVoxel attenuations are obtained by bilinear interpolation.

3.1 FreeVoxel Attenuation Construction

We choose 26 ray directions such that they pass through the voxel centers and one of voxel corners, voxel edge midpoints, and face midpoints. We compute the FreeVoxel attenuation at each voxel in the volume along each of the chosen ray directions using raycasting. Along each ray, an incrementally accumulated product

$$A_{v_k} = \prod_{i=1}^{k-1} (1-\alpha_{v_i})$$

is updated and stored at each voxel v_k.

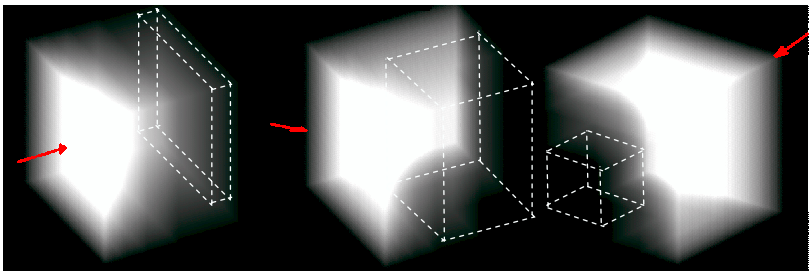

Figure 2: FreeVoxel Attenuation of a uniformly gray and translucent cube along three different directions (red arrows). The dotted boundaries show areas where maximum attenuation occurs for each of the three directions.

Figure 2 shows images of a uniformly gray translucent cube whose voxels all have the same transparency. For illustration, the images show a view different from the viewing directions (shown with red arrows) indicate viewing direction. The attenuation can be seen to increase through the volume along the viewing direction. Regions of the volume that are totally occluded are shown with dotted boundaries.

Time complexity: The algorithm adopted does not visit any voxel twice for computing attenuation along a direction. Since, in our sampling of rays, every ray passes through centers of all the voxels along its path, the number of unique rays traced is O(N^2). Since for each ray, the attenuation of up to N voxels will be updated, the method is O(N^3).

3.2 Hierarchy and Attenuation Function Filtering

Given a fine resolution of FreeVoxels, computing a hierarchy of FreeVoxels involves, first constructing a hierarchy of blocks of voxels and then the FreeVoxel attenuation for each block in the hierarchy. We use an octree hierarchy for the former. The attenuation of each block is obtained by filtering the FreeVoxel attenuations of its eight children in the octree. Recall that the attenuation represents the exponential term in Equation 3. We approximate the integral in the exponent for a block to be the arithmetic mean of the integrals in the exponents of its children. Thus, at each value of the parameter t along a ray

$$A_M = e^{-\int_{t_1}^{t} \alpha_M(s)ds} = e^{\frac{\sum_{i=1}^{8} -\int_{t_1}^{t} \alpha_i(s)ds}{8}} = \sqrt[8]{\prod_{i=1}^{8} A_{B_i}} \qquad (8)$$

The subscripts to the functions α denote the block/sub-block under consideration. Thus geometric mean is chosen as the filter function in Equation 6.

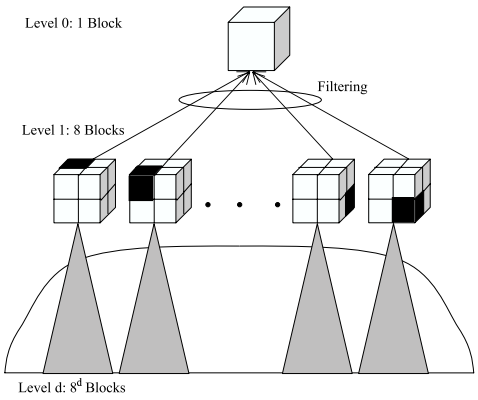

Figure 3: Blocks in the Octree hierarchy of Attenuation functions.

Storage: The total amount of storage S required in our implementation is given by $S = N_B S_B$ where N_B is the number of blocks and S_B is the storage required per block. N_B is O(N^3) since it is simply the number of nodes in the octree (figure 3) . Each block stores 27 bytes: 26 bytes for attenuation along 26 directions (with one byte used to represent each value between 0 and 1) and 1 byte to store the scalar value of the voxel. Thus the overall storage required is O(N^3) for a cubical volume with N voxels along each side.

3.3 Rendering

The rendering phase is quite straightforward and needs the viewpoint and the view frustum information as its only input. The blocks in each level of the hierarchy are uniformly and statically distributed amongst the render nodes. Each node, given view frustum information, identifies the blocks to be rendered, their levels in the hierarchy and directions along which attenuations and hence colors should be computed. The level in the hierarchy is decided for each block depending on parameters like screen projection area. For every block rendered, a look-up is performed on the pre-computed 3D function (equation 7) to obtain A_P^h. At the end of each frame, the images rendered by each node are added to yield the final image.

The color C of a block B, is retrieved from filtered color data, and the contribution I of B to the image *is determined independent of any other block* as $I = C\alpha_B A_P^h$ where α_B the filtered opacity value of the block. Thus, each voxel is rendered independent of any other voxel.

Rendering each frame involves determination of contribution made by each block to the image and clearly takes time that is O(N_B) where N_B is the number of blocks rendered. If the Freevoxel attenuation data are uniformly and statically distributed across R_n render nodes, since there is no dependency between rendering nodes, the potential speedup can theoretically be R_n.

During the construction of the FreeVoxel attenuation, a floating point intermediate result is maintained to prevent accrual of errors. However, at the end of the preprocess, the attenuations are stored along each direction in one byte and so the resolution of the representation is 1.0/256. Thus error is maintained low except when the viewpoint is inside the volume. In this case, a division is required (see Section 2.1) and thus there is a slight amplification of error.

4 Applications of FreeVoxels

4.1 Order Independence

In the case of discrete volume data, sampled along a structured grid, the coordinates of P(Section 2.1), *x*,*y* and *z* take integral values and P represents a voxel in the volume. A numeric solution

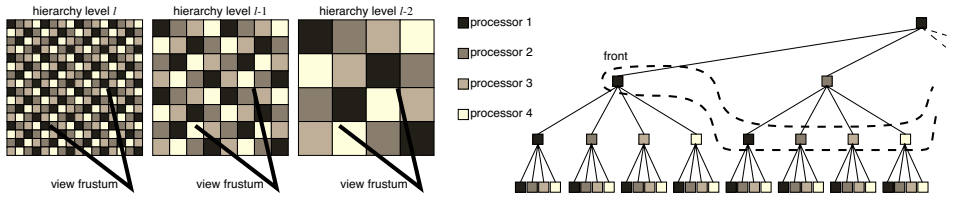

Figure 4: Static, uniform data distribution a) Distribution of blocks among multiple rendering nodes. b) Blocks being rendered by different rendering nodes are shown in the hierarchy. Note the round robin distribution of blocks in each level of the hierarchy among the different rendering nodes.

to the integral equation above is commonly obtained by employing a zero-order quadrature of the inner integral along with a first order approximation of the exponential, while the outer integral is approximated with a finite sum of uniform samples.

$$I_R = \sum_{k=1}^{M} [C_k \alpha_k \prod_{i=1}^{k-1} (1-\alpha_i)] \tag{9}$$

C_k is the color derived from the illumination model and α_k) is the transparency of samples along the ray. Since the product term $\prod_{i=1}^{k-1}(1-\alpha_i)$ is stored for each voxel parameterized on the ray direction R (θ,ϕ), the contribution of any one voxel v to the image can be computed as $C_v \alpha_v A(\theta,\phi)$. Rendering involves a simple accumulation (by addition) of contributions by each voxel along its viewing direction. This summation liberates us from order dependency, since we have stored the product term for each voxel and the summation is order independent.

4.2 Single-pass Lighting and Rendering

The FreeVoxel attenuation of each voxel represents the amount of light reaching it along different directions. When the volume is illuminated, the attenuation of a voxel v along the direction from the light source, L_1, is used to scale the amount of light reaching v from L_1. As a result of the pre-computed attenuation, shadows are visible on voxels with opaque, gray occluders between them and the light source. The method is applicable to colored light sources as well, by considering the three channels separately. By appending the FreeVoxel data structure with FreeVoxel Color attenuation along each direction for each channel, shadow effects due to light sources through colored voxels can be obtained. Figure 9 shows an illuminated volume.

Illumination by multiple light sources can also be achieved, by simply aggregating the effects of individual sources; this computation still requires only one pass through the volume. When each voxel is being rendered, the FreeVoxel data structure needs to be queried for FreeVoxel attenuation once along each direction of illumination and once along the viewing direction.

4.3 Data distribution and Synchronization

The FreeVoxel data structure contains enough information to compute the color contribution of a voxel to the image. Hence no exchange of information is required. Since the rendered frame by each renderer can be composited in any order, no synchronization is required.

The FreeVoxels (FreeVoxel attenuation functions and the filtered color hierarchy) constructed during preprocessing are distributed statically and uniformly across each level in the hierarchy, amongst the rendering nodes. Static distribution implies that a particular block B is always rendered by the same rendering node while uniform distribution implies that statistically, for a random sequence of view-points, the expected number of blocks rendered by each render node will be the same.

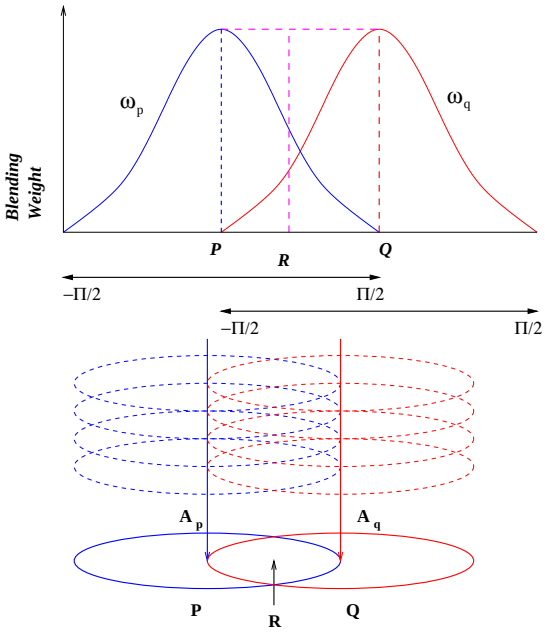

Figure 5: Top: Blending weights for two adjacent splats. Bottom: Accumulation of splats along the viewing direction.

4.4 Attenuation Leakage-free Splatting

Splatting techniques[Mueller and Crawfis 1998; Zwicker et al. 2001] that use weighted normalization on the opacity value of voxels for computing color are faced with the undesirable *attenuation leakage* problem. Consider two voxels P and Q (Figure 5) with transparency values α_P and α_Q, colors C_P and C_Q and attenuations along a particular direction as A_P and A_Q. Let their normalization weights be w_P and w_Q at some point R on screen where their splats overlap, then their individual colors are given by $I_P = \alpha_P\, w_P\, C_P\, A_P$ and $I_Q = \alpha_Q\, w_Q\, C_Q\, A_Q$ respectively. Note that $w_P + w_Q = 1$.

In traditional back-front splatting, one of the splats (say P) would be in front of the other and the screen color would be computed as

$$I_1 = I_P + (1 - \alpha_P\, w_P) I_Q \tag{10}$$

However using FreeVoxel attenuation, we correctly compute the screen color to be

$$I_2 = I_P + I_Q \tag{11}$$

The difference $|I_1 - I_2|$ is called the *attenuation leakage*.

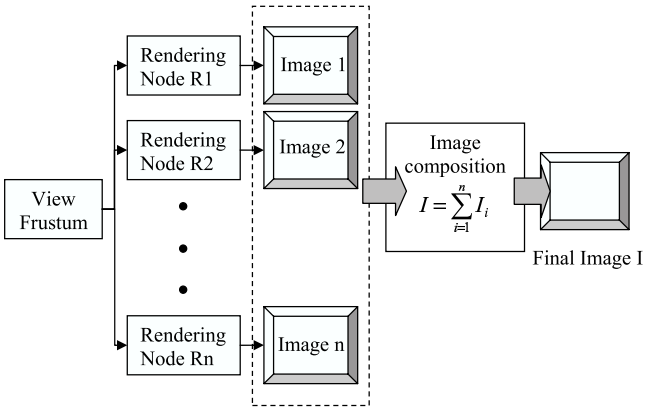

Figure 6: Distributed volume rendering using FreeVoxel attenuation does not involve any data migration or synchronization between rendering nodes. The final image is just an addition of individual images.

5 Implementation

All experiments were performed on a 2.20GHz Intel Pentium 4 workstation with 512MB RAM. The FreeVoxel data structure has a size of 27 bytes per block. Twenty-six of these are used to store FreeVoxel attenuation and one byte to store the scalar value representing color information.

A memory mapped array employing a 3D Hilbert space filling curve for linearization[Hilbert 1891; Lawder and King 2001] was used to store attenuation information. The array is both persistent and capable of handing large amounts of data (tested positively up to $4000 \times 2000 \times 2000$ elements).

A simple splatting technique was implemented for rendering the volume data. The splat-shape chosen was a disc with a Gaussian blending function on the alpha channel to modulate the color I of the voxel being rendered. The intensity of a pixel x_i, y_i belonging to the splat is given by $I\ cos^2 x_i\ cos^2 y_i$, where x_i and y_i are pixel coordinates with respect to the splat center, scaled to be in $[-\pi/2, \pi/2]$. This formulation guarantees uniform blending as long as, in image space, the boundary of each rendered splat reaches the center of its neighbors. In order to ensure this condition and to avoid aliasing effects, we render each splat as facing the viewer and perform a non-uniform scaling of the shape depending on the view angle. Interestingly, these calculations need not to be computed more than once per frame when using orthographic projection, where all splats assume the same shape.

A realistic simulation of multiple rendering nodes was performed by distributing data statically and uniformly amongst multiple virtual render nodes. The simulation is a good indicator of actual speedup because speedup is a ratio and also because no communication is required between render nodes until image composition for each frame rendered. A graph (figure 8) was plotted of the speedup against the number of nodes simulated. The graph supports our theory and shows that the speedup is indeed close to the number of render nodes.

6 Conclusion and Future Work

This paper describes a concept that allows us to render each voxel independent of other voxels. By doing this, we eliminate the need for any particular order during rendering. As a result, we can achieve maximum potential speedup (theoretically) without constraints on the data distribution requirements or synchronization between render nodes in a distributed volume rendering setup. The attenuation leakage problem (Section 4.4) is solved by using the FreeVoxel attenuation. Using a cluster for rendering and adopting the splatting technique, each rendering node independently renders a frame with data from a static distribution. Thus communication is required only to accumulate the image for each frame.

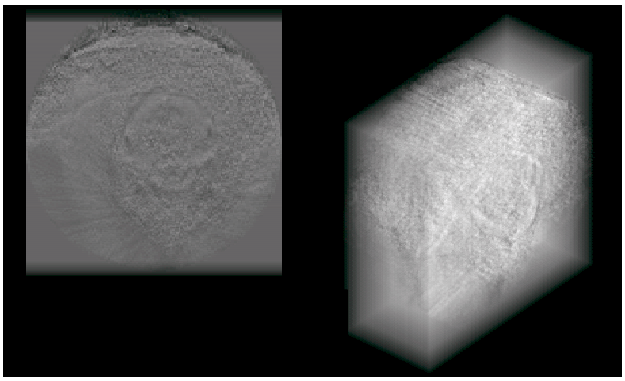

Figure 7: 128x128x128 Human brain dataset obtained from CT scans.

This paper also demonstrates a scheme that allows the representation of the Freevoxel attenuation to be stored hierarchically, which lends itself to multi-resolution rendering.

We believe that there are a number of improvements that can be made to the implementation. High quality ray casting as preprocess, compact representations for the attenuation, cluster implementations of preprocess and rendering dealing with larger datasets, incorporating perspective projection and interesting filters are some of the improvements foreseen.

References

ADELSON, E. H., AND BERGEN, J. R. 1991. *Computational Models of Visual Processing.* Cambridge, MA: MIT Press.

BAJAJ, C., IHM, I., AND PARK, S. 2001. *Visualization-Specific Compression of Large Volume Data.* Proceedings of Pacific Graphics, October.

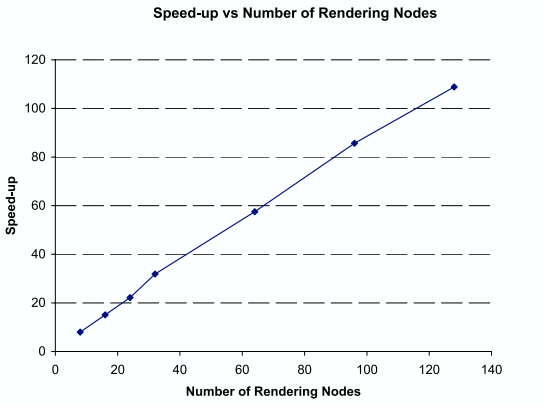

Figure 8: Graph shows that speed-up is almost equal to the number of rendering nodes. Speed-up was calculated with 8, 16, 32, 64, 96 and 128 render nodes.

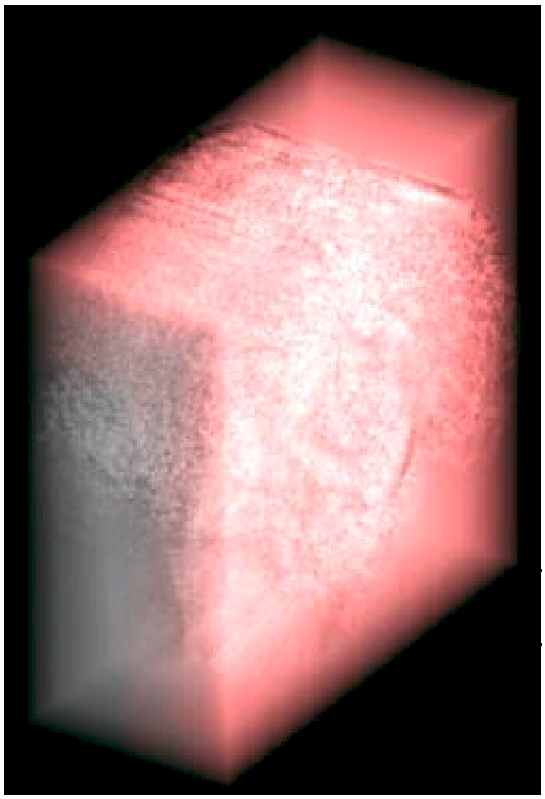

Figure 9: Human head dataset illuminated with red light.

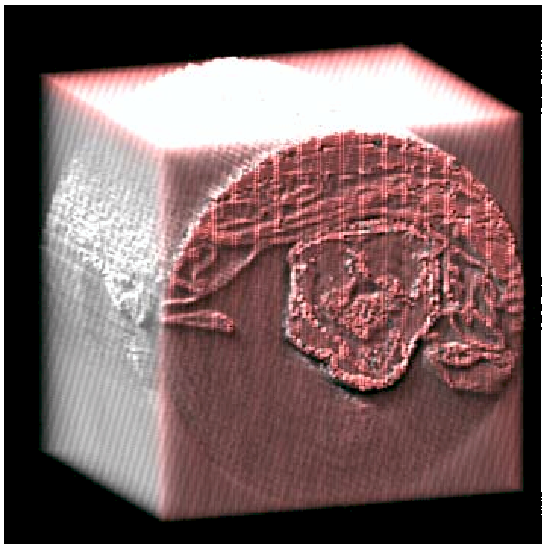

Figure 10: 128x128x128 Human head dataset(modified) from Stanford volume data archive with red and green illumination.

BHANIRAMKA, P., AND DEMANGE, Y. 2002. Opengl volumizer: a toolkit for high quality volume rendering of large data sets. In *Proceedings of the 2002 IEEE symposium on Volume visualization and graphics*, IEEE Press, 45–54.

BRODLIE, K., AND WOOD, J. 2001. *Recent Advances in Volume Visualization*. Computer Graphics Forum.

CABRAL, B., MAX, N., AND SPRINGMEYER, R. 1987. *Bidirectional reflection functions from surface bump maps*. ACM SIGGRAPH.

CAMAHORT, E., AND CHAKRAVARTY, I. 1993. Integrating volume data analysis and rendering on distributed memory architectures. In *Proceedings of the 1993 symposium on Parallel rendering*, ACM Press, 89–96.

DACHILLE, F., KREEGER, K., CHEN, B., BITTER, I., AND KAUFMANY, A. 1998. *High-Quality Volume Rendering Using Texture Mapping Hardware*. Eurographics/Siggraph Workshop on Graphics Hardware.

DREBIN, R. A., CARPENTER, L., AND HANRAHAN, P. 1988. *Volume rendering*.

EXLUNA, INC. 2002. *Entropy 3.1 Technical Reference*, January.

FRENKEL, K. A. 1989. Volume rendering. *Commun. ACM 32*, 4, 426–435.

FRIEDER, G., GORDON, D., AND REYNOLDS, R. 1985. *Back to Front Display of Voxel-Based Objects*. Computer Graphics and Applications, January.

GAO, J., HUANG, J., SHEN, H.-W., AND KOHL, J. A. 2003. *Visibility Culling Using Plenoptic Opacity Functions for Large Volume Visualization*. IEEE Visualization.

GARCIA, A., AND SHEN, H.-W. 2002. *An Interleaved Parallel Volume RendererWith PC-clusters*. Fourth Eurographics Workshop on Parallel Graphics and Visualization.

GARCIA, A., AND SHEN, H.-W. 2002. *An interleaved parallel volume renderer with PC-clusters*.

GRIBBLE, C., CAVIN, X., HARTNER, M., AND HANSEN, C. 2003. *Cluster-Based Interactive Volume Rendering with Simian*. Technical Report.

GUTHE, S., WAND, M., GONSER, J., AND STRABER, W. 2002. Interactive rendering of large volume data sets. In *Proceedings of the conference on Visualization '02*, IEEE Computer Society, 53–60.

HADWIGER, M., BERGER, C., AND HAUSER, H. 2003. *High-Quality Two-Level Volume Rendering of Segmented Data Sets on Consumer Graphics Hardware*. IEEE Visualization.

HILBERT, D. 1891. *Ueber stetige Abbildung einer Linie auf ein Flachenstuck*. Mathematische Annalen.

HSU, W. M. 1993. Segmented ray casting for data parallel volume rendering. In *Proceedings of the 1993 symposium on Parallel rendering*, ACM Press, 7–14.

HUANG, J., MUELLER, K., SHAREEF, N., AND CRAWFIS, R. 2000. *Fastsplats: Optimized splatting on rectilinear grids*. In Proceedings IEEE Visualization.

KAJIYA, J. T., AND HERZEN, B. P. V. 1984. *Ray tracing volume densities*. ACM SIGGRAPH.

LACROUTE, P., AND LEVOY, M. 1994. *Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation*. ACM SIGGRAPH Proceedings.

LAWDER, J. K., AND KING, P. J. H. 2001. Querying multi-dimensional data indexed using the hilbert space-filling curve. *SIGMOD Rec. 30*, 1, 19–24.

LEVOY, M. 1990. *Efficient Ray Tracing of Volume Data*. ACM Transactions on Graphics.

LEVOY, M. 1992. *Volume Rendering using the Fourier Projection-Slice Theorem*. Proceedings Graphics Interface.

LI, P. P., WHITMAN, S., MENDOZA, R., AND TSAIO, J. 1997. Parvox: a parallel splatting volume rendering system for distributed visualization. In *Proceedings of the IEEE symposium on Parallel rendering*, ACM Press.

MACROBERT, T. 1948. *Spherical harmonics; an elementary treatise on harmonic functions, with applications*. Dover Publications.

MARSCHNER, S. R., AND LOBB, R. J. 1994. An evaluation of reconstruction filters for volume rendering. In *Proceedings of the conference on Visualization '94*, IEEE Computer Society Press, 100–107.

MUELLER, K., AND CRAWFIS, R. 1998. Eliminating popping artifacts in sheet buffer-based splatting. In *Proceedings of the conference on Visualization '98*, IEEE Computer Society Press, 239–245.

NADEAU, D. R., KREMENEK, G., EMMART, C., AND WYATT, R. 2002. *A Case Study in Large Data Volume Visualization of an Evolving Emission Nebula*. Supercomputing.

NIEH, J., AND LEVOY, M. 1992. *Volume Rendering on Scalable Shared-Memory MIMD Architectures*. Proceedings Workshop on Volume Visualization.

PARK, S., BAJAJ, C., , AND IHM, I. 2001. *Effective Visualization of Very Large Oceanography Time-varying Volume Dataset*. CS & TICAM Technical Report, University of Texas at Austin.

SILLION, F. X., ARVO, J., WESTIN, S. H., AND GREENBERG, D. 1991. *A global illumination solution for general reflectance distributions*. ACM SIGGRAPH.

TAKEUCHI, A., INO, F., AND HAGIHARA, K. 2003. An improvement on binary-swap compositing for sort-last parallel rendering. In *Proceedings of the 2003 ACM symposium on Applied computing*, ACM Press, 996–1002.

TOTSUKA, T., AND LEVOY, M. 1993. *Frequency Domain Volume Rendering*. ACM SIGGRAPH Proceedings.

WILLIAMS, P. L., AND USELTON, S. P. 1996. *Foundations for Measuring Volume Rendering Quality*. Technical Report, NASA Ames Research Center.

WILSON, B., MA, K.-L., AND MCCORMICK, P. S. 2002. A hardware-assisted hybrid rendering technique for interactive volume visualization. In *Proceedings of the 2002 IEEE symposium on Volume visualization and graphics*, IEEE Press, 123–130.

WITTENBRINK, C. M. 1998. *Survey of parallel volume rendering algorithms*. In Proceedings Parallel and Distributed Processing Techniques and Applications.

ZHANG, X., BAJAJ, C., AND BLANKE, W. 2001. *Scalable Isosurface Visualization of Massive Datasets on COTS-Cluster*. IEEE Symposium on Parallel and Large-Data Visualization and Graphics.

ZWICKER, M., PFISTER, H., VAN BAAR, J., AND GROSS, M. 2001. *EWA Volume Splatting*. Eurographics.