University of Zurich[UZH]

Kartic Subr
Pablo Diaz-Gutierrez
Renato Pajarola
M. Gopi

Order Independent, Attenuation-Leakage Free Splatting using FreeVoxels

TECHNICAL REPORT – No. IFI-2007.01

2007

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Kartic Subr, Pablo Diaz-Gutierrez, Renato Pajarola, M. Gopi
Order Independent, Attenuation-Leakage Free Splatting using FreeVoxels
Technical Report No. IFI-2007.01
Visualization and Multimedia Lab
Department of Informatics (IFI)
University of Zurich
Binzmuehlestrasse 14, CH-8050 Zurich, Switzerland
http://vmml.ifi.uzh.ch/

Order Independent, Attenuation-Leakage Free Splatting using FreeVoxels

Kartic Subr Pablo Diaz-Gutierrez Renato Pajarola M. Gopi
Computer Graphics Lab, University of California Irvine
Visualization and MultiMedia Lab, Unviersity of Zürich
{kartic, pablo, gopi}@ics.uci.edu, pajarola@acm.org

Technical Report ifi-2007.01, Department of Informatics, Unviersity of Zürich

Abstract

In splatting-based volume rendering, there is a well-known problem of attenuation leakage, that occurs due to blending operations on adjacent voxels. Hardware accelerated volume splatting exploits the graphics hardware's alpha-blending capability to achieve attenuation from layers of voxels. However, this alpha-blending functionality results in accumulated errors(attenuation leakage), if performed on multiple overlapping alpha-values. In this paper, we introduce the concept of FreeVoxels which are self-sufficient structures in which the data required for operations on voxels are pre-computed and stored. These data are used to render each voxel independently in any order and also to eliminate the attenuation leakage. The drawback of the FreeVoxel data structure, that this paper does not address, is that it requires a significant amount of extra storage. Despite that, the advantages of a FreeVoxel data structure warrant extensive investigations in this direction. Specifically, FreeVoxel can be used, other than in solving the attenuation leakage problem, to achieve order-independent rendering; in parallel volume rendering, use of FreeVoxels allows arbitrary static data distribution with no data migration; it also enables synchronization-free rendering without compromising load-balancing. A similar data structure with comparable memory requirements has also been used for opacity based occlusion culling in volume rendering by [10]. In this paper, we also describe a hierarchical extension of FreeVoxels that lends itself to multi-resolution rendering.

1. Introduction and Motivation

Research in volume visualization, particularly volume rendering [3], has focused on important aspects in the field such as handling large data [5, 2, 13, 13, 33], improving quality of rendering [9, 8, 20, 21, 32, 7, 14] and improving rendering efficiency [26, 30, 17, 35, 12].

Parallel volume rendering has enjoyed much attention [16, 22, 11, 29], motivated by applications ([27, 25]) dealing with large amounts of data. In [34], it is reported that only few cluster-parallel algorithms have been developed, for ray casting and Fourier domain volume rendering. However, splatting on clusters has been left out so far.

Even non-parallel hardware supported implementation of traditional splatting-based volume rendering suffers from the attenuation leakage problem due to incorrect blending operations. Specifically, when discretized voxels are rendered, adjacent voxel contributions are aggregated in their overlap region. This aggregating function is not correctly normalized resulting in the incorrect attenuation in the overlap region – the problem referred to as attenuation leakage.

Traditionally, volume rendering is done by sweeping the data either front-to-back or back-to-front. In parallel rendering, such order-dependency leads to serious disadvantages, like lower potential speed-up, data distribution issues and serious synchronization problems. These issues have motivated research in segmentation and data distribution [16, 5], and synchronization. The limited 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.

In this paper, we introduce the concept of FreeVoxels that, in general, enables operations on individual voxels independent of other voxels at the cost of constant amount of extra storage per voxel. Specifically, if this extra information that is stored is the accumulated attenuation then this FreeVoxel Attenuation solves the problem of attenua-

tion leakage in splatting based volume rendering. Interestingly, it also enables order-independent rendering solving many problems in parallel implementation of these algorithms. In this context, we show that by using FreeVoxel Attenuation, the constraints on data distribution are eased and there is no need for synchronization within a frame. Further, restrictions on data segmentation are eliminated. 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.

In Section 4.3, we extend this concept of FreeVoxels to accommodate a single pass lighting and rendering algorithm. 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.

The drawback of the FreeVoxel data structure, that this paper does not address, is that it requires significant more storage than conventional techniques. A similar data structure with comparable memory requirements has also been used for opacity based occlusion culling in volume rendering by [10]. Gao et al. [10] describe a method of improving speedup by using a data structure constructed as a preprocess to decide whether a voxel contributes its color to the image or not when seen from a specific direction; however, front-to-back order is imposed on rendering. They have also reported tremendous speed-up in parallel implementation of occlusion culling. Our work presented in this paper uses similar amount of memory to solve various other problems like attenuation leakage and single-pass lighting. In fact, the plenoptic opacity function can also be well integrated into the FreeVoxel data structure.

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

- *FreeVoxel data structure:* We introduce a generic structure that expends memory to store additional pre-computed information to be used not just for the sake of improving efficiency but also the correctness of the result.
- *Avoiding attenuation leakage:* We show how the FreeVoxel attenuation can be used to solve the leakage problem in some splatting techniques.
- *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.

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

2. FreeVoxels

FreeVoxels encapsulate all data required by each voxel, to be operated on independently of each other. 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 [1, 10] $A(x,y,z,\theta,\phi)$ 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

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

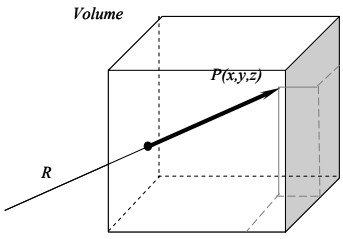

Figure 1. Figure shows a ray R and the interval (thickened) along R where R is attenuated before reaching P

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

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

where I_R is the intensity along ray R, $I(t)$ 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

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

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

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

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

For a hierarchically organized volume, the FreeVoxel attenuation A_P^h for a block P in the hierarchy level h is

$$A_P^h(h, \theta, \phi) : H \times \Theta \times \Phi \rightarrow \alpha \quad (6)$$

where H is a set of integers in the range $[0, \lceil log_n N \rceil]$. This value for every block is calculated using an appropriate filter function on the FreeVoxel attenuation values of its children's sub-blocks.

3. Algorithms

In this section, we describe the algorithm to construct the FreeVoxel attenuation, the filtering technique to compute the FreeVoxel attenuation for a hierarchy of voxels, and finally the volume rendering algorithm that uses the FreeVoxel attenuation data structure.

3.1. FreeVoxel Attenuation Construction

FreeVoxel attenuation for a voxel is the accumulation of opacity values from the viewpoint until that voxel in the viewing direction. We compute this value for a given voxel and viewing direction using ray-casting. In our implementation we choose 26 ray directions such that they pass through the voxel centers and one of voxel corners, voxel edge midpoints, and face midpoints. These are the only rays that pass through the centers of all the voxels along their path. The choice of these rays simplifies the precomputation of FreeVoxel attenuation and brings down the complexity of the number of unique rays traced in the ray casting to O(N^2), where N is the size of the data along one dimension. Since for each ray, up to N voxels will be updated, the complexity of computing the FreeVoxel attenuation for the entire volume is O(N^3). 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, where $1 \leq i \leq k-1$ denotes the sequence of voxels along the ray direction from outside the volume until the voxel v_k.

We illustrate the computed values of A_{v_k} in Figure 2. This figure 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 accumulated attenuation A_{v_k} 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.

In our implementation, 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.

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 (refer to Figure 3). 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, the FreeVoxel attenuation of a block M with children B_i ($1 \leq i \leq 8$) is given by

$$A_M = e^{-\int_{t_1}^{t} \alpha_M(s) ds} = e^{\frac{\sum_{i=1}^{8} -\int_{t_1}^{t} \alpha_{B_i}(s) ds}{8}} = \sqrt[8]{\prod_{i=1}^{8} A_{B_i}} \quad (7)$$

Thus, we obtain the FreeVoxel attenuation of a block by computing the geometric mean of the attenuations of its children. More generally, this corresponds to computing the

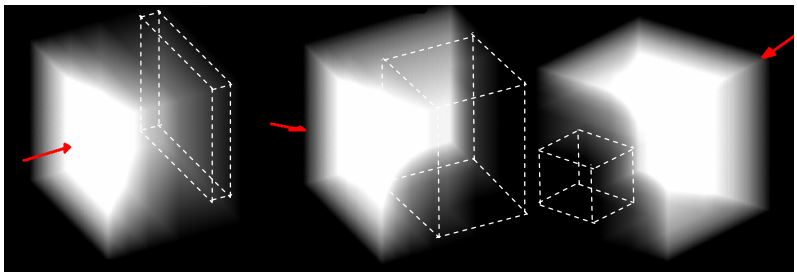

Figure 2. FreeVoxel Attenuation of a uniformly gray and translucent cube along three different directions (red arrows). The dotted boundaries show areas where maximum attenuation occurs for each of the three directions.

attenuation of a block using the mean optical depth of its children. In contrast, the arithmetic mean of the attenuation of its children computes the average of the energy reaching them.

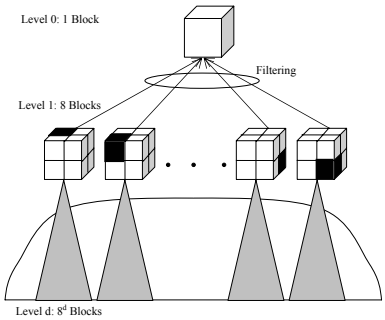

Figure 3. Blocks in the Octree hierarchy of Attenuation functions.

3.3. Rendering

The rendering phase is quite straightforward and needs the viewpoint and the view frustum information as its only input. Using this information, the voxels/blocks to be rendered, their levels in the hierarchy and their directions towards the viewpoint are determined. The level in the hierarchy is decided for each block depending on parameters like screen projection area. For every block B rendered, a look-up is performed on the pre-computed FreeVoxel data structure to obtain the accumulated opacity value is A_B^h in the given direction (Equation 6). Further, the color C_B and the filtered opacity value α_B of the block B are also retrieved. The final contribution of B to the image is $C\alpha_B A_P^h$, which *is computed independent of any other block*. Thus, voxels are rendered independent of each other and the final image is just an accumulation of individual voxel renderings.

It is important to note that the viewpoint can also be placed within the volume, say in voxel V. In this case, the accumulated opacity value of any other voxel/block B is computed as A_B^h/A_V along the direction from V to B.

Parallel rendering: For cluster-parallel rendering, the blocks in each level of the hierarchy are uniformly and statically distributed amongst the render nodes R_n. Since blocks are rendered independent of each other, there is no dependency between rendering nodes, and hence the potential speedup is n.

3.4. Storage requirements

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). The space required per voxel is proportional to the number of rays casted. Despite its high space requirements, the advantages of the FreeVoxels data structure we show in the next section lead us to think it deserves a deeper study.

4. Applications of FreeVoxels

4.1. Attenuation Leakage-free Splatting

Splatting techniques such as [31, 18, 6, 24, 35] use overlapping blending kernels for a weighted interpolation of opacity values for computing properly attenuated color contributions of voxels to the direct volume rendering (DVR) integral. With hardware supported α-blending such splatting is faced with an undesirable *attenuation leakage* problem.

Consider two voxels p and q (Figure 4) with transparency values α_p and α_q, intensity I_p and I_q and blending kernels ω_p and ω_q (with $\omega_p + \omega_q = 1$ for any point R in their intersection). The contribution of voxels p and q to a point R on screen is thus given by $I_{pq} = \omega_p\, \alpha_p\, I_p + \omega_q\, \alpha_q\, I_q$. More

importantly, the attenuation effect of p and q on voxels s and t farther away from R is determined by $\alpha_{pq} = \omega_p\ \alpha_p + \omega_q\ \alpha_q$ and the composition of the two layers of voxels is computed as

$$I_R = I_{pq} + (1 - \alpha_{pq}) \cdot I_{st}. \quad (8)$$

However, back-to-front α-blended splatting will yield the following composition (i.e. if the splatting order after I_{st} is first p and then q)

$$I_R^* = \omega_q \alpha_q\, I_q + (1 - \omega_q \alpha_q)\,(\omega_p \alpha_p\, I_p + (1 - \omega_p \alpha_p)\, I_{st}) \quad (9)$$

with the total attenuation of I_{st} being $(1 - \omega_q \alpha_q)(1 - \omega_p \alpha_p)$ rather than $1 - \omega_p\ \alpha_p - \omega_q\ \alpha_q$ as in Equation 8. Due to this incorrect attenuation we refer to the effective difference $I_R - I_R^*$ as *attenuation leakage*. If many layers of voxels are splatted by back-to-front α-blending (Figure 4 top) the effective attenuation leakage due to Equation 9 is greatly amplified.

With the FreeVoxel attenuation introduced in Section 2 we can correctly compute the contribution of individual splatted voxels avoiding the attenuation leakage problem outlined above. Given the attenuation A_p of a voxel p along a direction (θ, ϕ) through the volume to a point R on screen, we can directly compute the final contribution of layered voxels p, q and s, t to point R as

$$I_R = A_p \omega_p \alpha_p I_p + A_q \omega_q \alpha_q I_q + A_s \omega_s \alpha_s I_s + A_t \omega_t \alpha_t I_t. \quad (10)$$

Therefore, we can avoid the excessive multiplication of $(1 - \omega\alpha)$ terms, as required in back-to-front α-blending, which causes the attenuation leakage effect.

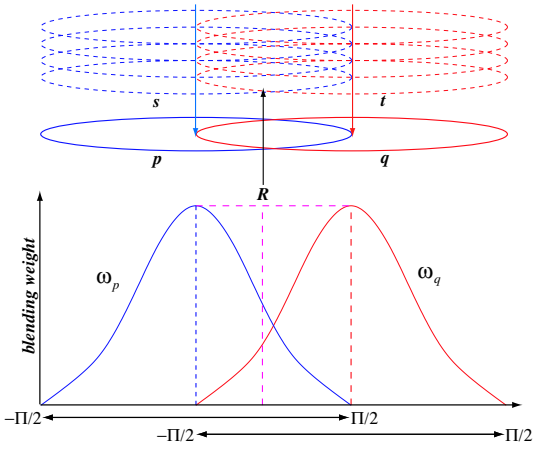

Figure 4. Top: Accumulation of splats along the viewing direction. Bottom: Blending weights for two adjacent splats.

4.2. Order Independence

In the case of discrete volume data, sampled along a structured grid, the coordinates of a point x,y and z take integral values and v represents a voxel in the volume. A numeric solution to the integral equation above (Equation 3) 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.

$$I_R = \sum_{k=1}^{M} [I_k \omega_k \alpha_k \prod_{i=1}^{k-1} (1 - \omega_i \alpha_i)] \quad (11)$$

I_k is the color intensity derived from the illumination model, α_k the transparency of samples along the ray and ω_k are the blending weights. Since the product term $\prod_{i=1}^{k-1}(1 - \omega_i \alpha_i)$ is stored as the FreeVoxel attenuation $A_v(\theta, \phi)$ for each voxel on the ray direction $R(\theta, \phi)$, the contribution of any one voxel v to the image can be computed as $I_v \omega_v \alpha_v A_v(\theta, \phi)$. Thus the value inside the outer summation in Equation 11 can be computed just by using the information stored in every FreeVoxel. Since rendering involves a simple addition (outer summation) of contributions by each voxel along its viewing direction, it is independent of the order in which voxels' contributions are accumulated.

4.3. 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 precomputed attenuation, shadows are visible on voxels with opaque, gray occluders between them and the light source. 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. Figure 7 shows an illuminated volume.

4.4. Data distribution and Synchronization

The FreeVoxel data structure contains all the required information to compute the final color contribution of a voxel to the image. Hence, in cluster-parallel volume rendering, no exchange of information is required between rendering nodes. Since the frame rendered by each render node can be composited in any order, no synchronization is required within a frame.

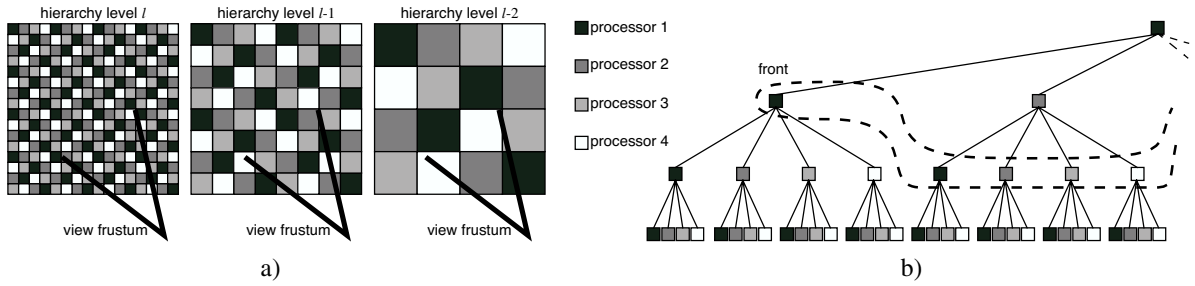

Figure 5. Static, uniform data distribution a) Distribution of blocks among multiple rendering nodes. b) Blocks being rendered by different rendering nodes are shown in the hierarchy. Note the round robin distribution of blocks in each level of the hierarchy among the different rendering nodes.

The FreeVoxels (attenuation functions and the filtered color hierarchy) constructed during preprocessing can be distributed statically and uniformly across each level in the hierarchy, amongst the rendering nodes as shown in Figure 5. Static distribution implies that a particular block B is always rendered by the same rendering node and there is no data migration between nodes. 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, leading to fair load balancing. The combined advantages of no data migration, hardly any synchronization constraints and fair load balancing, as a result of using FreeVoxels, allow for potentially ideal speedup.

5. Implementation

To perform initial tests we implemented the FreeVoxel data structure and a basic hardware accelerated splatting technique. As mentioned in Section 3, the FreeVoxel data structure has a size of 27 bytes per block. Twenty-six of these are used to store the FreeVoxel attenuation values and one byte to store the scalar, from which the color and per voxel α values are determined based on the transfer function. A memory mapped array employing a 3D Hilbert space filling curve for linearization [15, 19] is used to store attenuation data.

Our splatting system incorporates pre-integrated footprints of blending kernels ω_v that are applied to a voxel v as α-textured splats. Hence the pre-integrated kernel modulates the voxel's basic intensity contribution $\alpha_v\ I_v$. Each voxel is rendered as a splat with the footprint kernel texture and facing the viewer.

Figure 6 illustrates the effect of the FreeVoxel approach on the attenuation leakage problem outlined in Section 4.1 on a uniformly transparent cube. On the left side, a standard back-to-front ordered α-blended splatting is shown which exhibits significant attenuation leakage. The image on the right side demonstrates the improved attenuation properties using FreeVoxels.

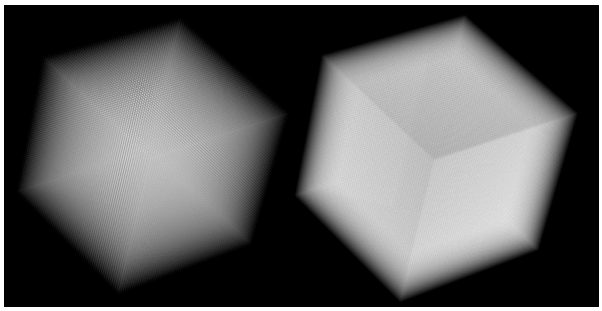

Figure 6. Left: Standard back-to-front α-blended splatting with noticeable attenuation leakage. Right: Order-independent accumulation-splatting using the FreeVoxel attenuation.

Figure 7 demonstrates the effect of single-pass lighting and rendering outlined ind Section 4.3 on a volume data set illuminated with red light. The image clearly shows the shadowing and color bleeding effects that can easily be achieved with the FreeVoxel attenuation.

We performed a simple simulation of multiple rendering nodes by distributing data statically and uniformly amongst multiple virtual render nodes. The simulation is a good indicator of an 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.

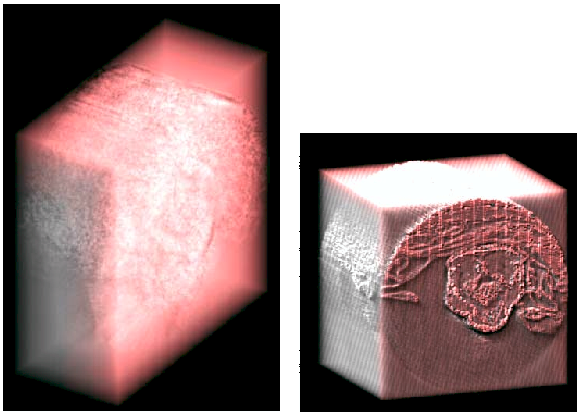

Figure 7. Human head dataset illuminated with red light.

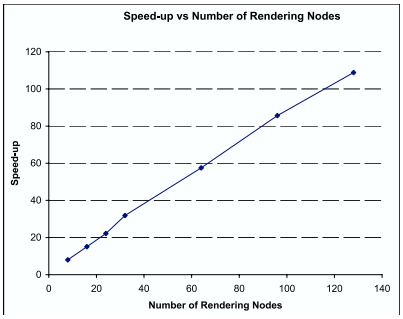

Figure 8. Graph shows that speed-up is almost equal to the number of rendering nodes. Speed-up was calculated with 8, 16, 32, 64, 96 and 128 render nodes.

6. Conclusion and Future Work

This paper describes a concept that introduces an abstraction for each voxel that allows it to be rendered independent of other voxels. Using this concept, we have also shown how the attenuation leakage problem can be eliminated using this concept (Section 4.1). Further, constraints on any particular order during rendering are eliminated. This allows a maximum potential speedup without constraints on the data distribution requirements or synchronization between render nodes in a distributed volume rendering setup. 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.

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.

Although there is indeed a high memory penalty in this particular implementation, methods such as that proposed by [10] can be used for more compact representations of the FreeVoxel attenuation. Some other improvements that we are working on are high quality ray casting as preprocess, cluster implementations of preprocess and rendering dealing with larger datasets and interesting filtering operations for multi-resolution rendering.

References

[1] E. H. Adelson and J. R. Bergen. *Computational Models of Visual Processing*. Cambridge, MA: MIT Press, 1991.

[2] P. Bhaniramka and Y. Demange. 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*, pages 45–54. IEEE Press, 2002.

[3] K. Brodlie and J. Wood. *Recent Advances in Volume Visualization*. Computer Graphics Forum, 2001.

[4] B. Cabral, N. Max, and R. Springmeyer. *Bidirectional reflection functions from surface bump maps*. ACM SIGGRAPH, 1987.

[5] E. Camahort and I. Chakravarty. Integrating volume data analysis and rendering on distributed memory architectures. In *Proceedings of the 1993 symposium on Parallel rendering*, pages 89–96. ACM Press, 1993.

[6] R. Crawfis and N. Max. Texture splats for 3D scalar and vector field visualization. In *Proceedings IEEE Visualization 93*, pages 261–266. Computer Society Press, 1993.

[7] F. Dachille, K. Kreeger, B. Chen, I. Bitter, and A. Kaufmany. *High-Quality Volume Rendering Using Texture Mapping Hardware*. Eurographics/Siggraph Workshop on Graphics Hardware, 1998.

[8] R. A. Drebin, L. Carpenter, and P. Hanrahan. *Volume rendering*, 1988.

[9] G. Frieder, D. Gordon, and R. Reynolds. *Back to Front Display of Voxel-Based Objects*. Computer Graphics and Applications, January 1985.

[10] J. Gao, J. Huang, H.-W. Shen, and J. A. Kohl. *Visibility Culling Using Plenoptic Opacity Functions for Large Volume Visualization*. IEEE Visualization, 2003.

[11] A. Garcia and H.-W. Shen. *An interleaved parallel volume renderer with PC-clusters*, 2002.

[12] A. Garcia and H.-W. Shen. *An Interleaved Parallel Volume RendererWith PC-clusters*. Fourth Eurographics Workshop on Parallel Graphics and Visualization, 2002.

[13] S. Guthe, M. Wand, J. Gonser, and W. Straber. Interactive rendering of large volume data sets. In *Proceedings of the conference on Visualization '02*, pages 53–60. IEEE Computer Society, 2002.

[14] M. Hadwiger, C. Berger, and H. Hauser. *High-Quality Two-Level Volume Rendering of Segmented Data Sets on Consumer Graphics Hardware*. IEEE Visualization, 2003.

[15] D. Hilbert. *Ueber stetige Abbildung einer Linie auf ein Flachenstuck*. Mathematische Annalen, 1891.

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

[17] P. Lacroute and M. Levoy. *Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation*. ACM SIGGRAPH Proceedings, 1994.

[18] D. Laur and P. Hanrahan. Hierarchical splatting: A progressive refinement algorithm for volume rendering. In *Proceedings ACM SIGGRAPH 91*, pages 285–288. ACM Press, 1991.

[19] J. K. Lawder and P. J. H. King. Querying multi-dimensional data indexed using the hilbert space-filling curve. *SIGMOD Rec.*, 30(1):19–24, 2001.

[20] M. Levoy. *Efficient Ray Tracing of Volume Data*. ACM Transactions on Graphics, 1990.

[21] M. Levoy. *Volume Rendering using the Fourier Projection-Slice Theorem*. Proceedings Graphics Interface, 1992.

[22] P. P. Li, S. Whitman, R. Mendoza, and J. Tsaio. Parvox: a parallel splatting volume rendering system for distributed visualization. In *Proceedings of the IEEE symposium on Parallel rendering*. ACM Press, 1997.

[23] T. MacRobert. *Spherical harmonics; an elementary treatise on harmonic functions, with applications*. Dover Publications, 1948.

[24] K. Mueller and R. Crawfis. Eliminating popping artifacts in sheet buffer-based splatting. In *Proceedings IEEE Visualization 98*, pages 239–245. Computer Society Press, 1998.

[25] D. R. Nadeau, G. Kremenek, C. Emmart, and R. Wyatt. *A Case Study in Large Data Volume Visualization of an Evolving Emission Nebula*. Supercomputing, 2002.

[26] J. Nieh and M. Levoy. *Volume Rendering on Scalable Shared-Memory MIMD Architectures*. Proceedings Workshop on Volume Visualization, 1992.

[27] S. Park, C. Bajaj, , and I. Ihm. *Effective Visualization of Very Large Oceanography Time-varying Volume Dataset*. CS & TICAM Technical Report, University of Texas at Austin, 2001.

[28] F. X. Sillion, J. Arvo, S. H. Westin, and D. Greenberg. *A global illumination solution for general reflectance distributions*. ACM SIGGRAPH, 1991.

[29] A. Takeuchi, F. Ino, and K. Hagihara. An improvement on binary-swap compositing for sort-last parallel rendering. In *Proceedings of the 2003 ACM symposium on Applied computing*, pages 996–1002. ACM Press, 2003.

[30] T. Totsuka and M. Levoy. *Frequency Domain Volume Rendering*. ACM SIGGRAPH Proceedings, 1993.

[31] L. Westover. Footprint evaluation for volume rendering. In *Proceedings SIGGRAPH 90*, pages 367–376. ACM SIGGRAPH, 1990.

[32] P. L. Williams and S. P. Uselton. *Foundations for Measuring Volume Rendering Quality*. Technical Report, NASA Ames Research Center, 1996.

[33] B. Wilson, K.-L. Ma, and P. S. McCormick. A hardware-assisted hybrid rendering technique for interactive volume visualization. In *Proceedings of the 2002 IEEE symposium on Volume visualization and graphics*, pages 123–130. IEEE Press, 2002.

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

[35] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. *EWA Volume Splatting*. Eurographics, 2001.