# Fast Volume Visualization using Pre-computed Volume Radiance Transfer

X. Hao[1], D. Xu[1,2], B. S. Peterson[1,2]

[1]Psychiatry, Columbia University, New York, New York, United States, [2]New York State Psychiatric Institute, New York, New York, United States

**Introduction** There are two kinds of volume visualization algorithms - indirect and direct. Indirect volume rendering such as Marching Cube first converts the volume data into a polygonal isosurface and then displays it. This method can leverage the advances in graphics hardware for display of large triangle datasets for interactive rendering. However it is unable to reveal the variations of density within the visualized volume. Direct volume rendering is better suited for visualizing such detail, but suffers from low interactivity. Our work has been motivated by a desire for an algorithm that can have the advantages of both indirect and direct volume rendering, i.e., the speed performance of indirect (surface) volume rendering with high quality subsurface details of direct volume rendering.

**Methods** The visual appearance of direct volume rendered objects depends on the radiance incident on volume grids, as well as the scattering of the radiance along the ray to the viewer. This is inherently a non-uniform scattering process. Most volume rendered datasets are characterized by an underlying structure, usually with meaningful boundaries. For instance, medical imaging datasets embody connected tissues and bones. Although the effects of direct volume rendering are from volume scattering processes, the final appearance of the object is determined by the total radiance that exits its surface. Under distant illumination, the appearance of a volumetric object with a well-defined boundary is determined by a 6D function – two dimensions to represent incoming light direction, two dimensions to represent the location of the exiting radiance point, and two dimensions for the viewing direction. We shall refer to this 6D function as the *Volume Radiance Transfer*.

Our algorithm consists of three steps:

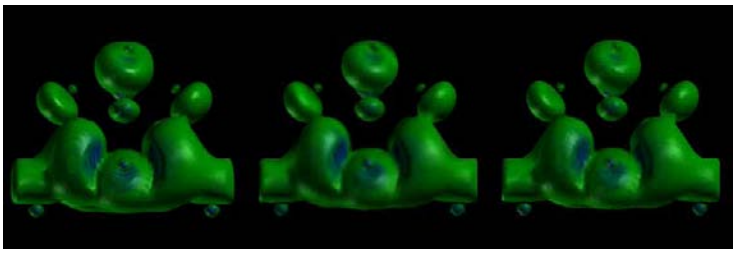
1. We first generate a surface representation of the volume data and we will associate the volume radiance transfer function for every extracted surface point on this representation. For objects with a well-defined boundary, a ray will not change after it exits the surface. We record this ray at the surface point where it exits. If we have this information for all the surface points and for all the view directions, we can very efficiently synthesize the appearance of the volumetric object at run time for any given view direction.
2. We then sample the volume-scattered radiance along sufficient number of directions using direct volume rendering algorithms and compress this volume radiance transfer function with spherical harmonics basis functions.
3. At run-time, we render the scene by re-sampling the surface-mapped volume radiance transfer for a given viewing direction.

**Results** We show the results obtained by our algorithm on semi-transparent volumetric datasets with varying viewing but fixed lighting conditions. We have used a 2GHz Pentium 4 PC running Windows 2000 with a nVIDIA GeForceFX 5900 graphics card. We have summarized our test results on Neghip dataset (a volumetric electronic density function) in Table 1 and Figure l. The images have a size of 512*512 and we sample 4000 view directions for the pre-computed radiance transfer. We have used a publicly available direct volume rendering software – OpenSplat to sample the radiance transfer function. The volume size of the dataset is $64^3$, and its isosurface consists of 70569 triangles and 35435 vertices. SHs is the abbreviation for the number of spherical harmonic basis functions used in the compression stage.

| | OpenSplat | 49 SHs | 100 SHs | 196 SHs | 400SHs | Interpolated |
|---|---|---|---|---|---|---|
| Memory (bytes/vertex) | N/A | 294 | 600 | 1.2K | 2.4K | 12K |
| Frame rate (frame/second) | 0.042 | 15.8 | 12.5 | 7.9 | 4.5 | 31.5 |

**Table 1**: Comparison of running time and memory requirements

One can see that the rendering speed declines as the number of spherical harmonic basis functions used to represent the volume radiance transfer increases. The *Interpolated* column in Table 1 refers to directly using the pre-computed volume radiance transfer function, without any spherical harmonic compression. For this case, we linearly interpolate among the four pre-computed radiance values that are closest to the current viewing position. We can achieve about 32 frames per second in rendering for interpolated radiance transfer with this method. This is orders of magnitude faster than OpenSplat. With spherical harmonics compression, we can achieve about 8 frames per second with run-time evaluation of 196 basis functions on the CPU. Our run-time algorithm has lower time complexity than direct volume rendering algorithms. The run-time rendering costs of our approach scale linearly with the size of the isosurfaces that lie along meaningful volume data boundaries. In general, for a volume data with n voxels, the isosurface complexity is $O(n^{2/3})$, while the cost for direct volume rendering is $O(n)$. This suggests that the running times for our approach will scale favorably as the dataset sizes increase.



(a) OpenSplat (0.042 fps) (b) Use of 196 SHs (7.9 fps) (c) Interpolated (31.5 fps)

**Figure 1.** Comparisons of images generated by OpenSplat and our algorithm

**Discussion** The unified approach we present here can take advantage of the better interactivity of surface-based indirect volume rendering and the high-quality appearance with subsurface details of direct volume rendering. Our approach can efficiently render semi-transparent volume data with meaningful boundaries and thus are quite useful for medical datasets, such as MRI data. We believe that our method can profitably combine the best features of direct- and indirect-volume rendering to deliver high-quality volume renderings at interactive rates.

**References**

[1] Max, N., *Transactions on Visualization and Computer Graphics*, 1995. 1: p99-108.

[2] Mueller, K., et al. *IEEE Transactions on Visualization and Computer Graphics*, 1999. 5: p116-134.

[3] Ramamoorthi, R. and Hanrahan, P., *ACM Transactions on Graphics*, 2001, 20: p117-128