Interactive Transfer Function Modification for Volume Rendering Using Pre-Shaded Sample Runs

Vivek Srivastava Uday Chebrolu Klaus Mueller

Center for Visual Computing, Computer Science, Stony Brook University, USA

Abstract

This paper describes a software-based method for interactive transfer function modification. Our approach exploits the fact that, in general, a user will rarely want to modify the viewpoint and the transfer functions at the same time. In that spirit, we optimize the latter by first fixing the viewpoint and then storing a list of pre-shaded, but uncolored, samples along each ray. Then, each time the RGBA transfer function is modified, the algorithm traverses the sample lists, colors the samples, and composites them along each ray until full opacity is reached. Since neither the expensive sample interpolation nor the shading are no longer necessary, we can obtain near-interactive framerates for a variety of datasets.

1 Introduction

Volume rendering [5] is the process of exploring volumetric data using visuals. The exploration process aims to discover and emphasize interesting structures and phenomena embedded in the data, while de-emphasizing or completely culling away occluding structures currently not of interest. In volume rendering, two main instruments exist that control the exploration process: view navigation and transfer functions. Both are essential. The former determines the spatial position and orientation from which the user observes the scene. The latter controls the look-and-feel of the scene itself, which is done by ways of the transfer functions that map the raw object density data to color and transparencies. This also constitutes a navigation task, performed in a 4D transfer function space, assuming 3 axes for RGB color and one for transparency (or opacity).

Transfer function space navigation can be a time consuming task for two main reasons: (i) the space to be explored is large, and (ii) the volume rendering is, at least at the present time, only interactive when supported by hardware [8][9]. Hardware solutions, however, are only practical for moderate dataset sizes, unless expensive machines are used [7]. On the other hand, interactive software solutions also exist [4].

Common to all transfer function exploration efforts (see [3] for a sophisticated one) is the need for rendering the volume using the new settings. A key observation here is that the user rarely changes the viewing parameters *and* the transfer function simultaneously. Usually, the object stays fixed in space, and only the transfer function is modified, or vice versa, with the reason for this being that the user is simply too occupied with one of the two tasks to deal with the other. While view navigation can use temporal coherencies to speed up the rendering [6][11], for this research we seek to identify the coherencies that exist in the task of transfer function navigation. We start by realizing that the main work in volume rendering is formed by the following pipeline: interpolation of the ray samples, illumination (shading), coloring, and compositing. Since we leave both the view and the light source position unchanged, there is no need to perform the sample interpolation and shading for every new transfer function setting. Instead, we may cache away the densities and shades of the samples for repeated use in the transfer function exploration process.

{vivek, uday, mueller}@cs.sunysb.edu, http://www.cs.sunysb.edu

2 Theory

The low-albedo volume rendering integral can be written as follows:

$$r(u,v) = \int_0^L c(\mathrm{d}(s))\tau(\mathrm{d}(s))\exp\left(-\int_0^s \tau(\mathrm{d}(t))dt\right)ds \qquad (1)$$

where $r(u,v)$ is the value of the ray spawned at image coordinate (u,v), $d(s)$ is the (interpolated) volume density at location s along the ray, and c and τ are the color and extinction mapped to d, respectively, via the transfer functions. Note, that c is a 3-vector of (r, g, b). Using a Riemann sum and a Taylor series approximation of the exponential, we get the familiar compositing equation [5] that is used in practice:

$$r(u,v) = \sum_{i=0}^{L/\Delta s} c(\mathrm{d}(i\Delta s))\alpha(\mathrm{d}(i\Delta s))\left(\prod_{j=0}^{i-1}(1-\alpha(\mathrm{d}(j\Delta t)))\right) \qquad (2)$$

where α is the sample opacity, normalized for sample distance when the sampling distance $\Delta s \neq 1$.

The fact that we will be re-using the $d(s)$, i.e., the interpolated samples along the ray for a fixed viewpoint, leads to a simpler task than having to solve (1) directly from the raw volume, involving interpolation. Note also that this approach is different from that of Kaneda et al. [2] who decomposed (1) into a cosine series which allowed them to modify the color portion of the transfer function with interactive rendering response, but not the opacities.

3 Implementation

Our algorithm first acquires a list of samples along

each ray from the fixed viewpoint at unit sampling distance. Suppose, for this discussion, that the sequence of n samples is enumerated as integer pairs (s_k,d_k), where the s_k = [0,1,2,...n] denote the sample locations along the run (we exclude samples with densities below a certain value, e.g., that for air), and the d_k are the interpolated densities at each s_k, rounded to bytes for more efficient storage. Upon rendering, we traverse the list and use the densities in sequence. We use the densities to look up the colors and opacities in the current transfer functions, and we composite the RGBA samples front-to-back with early ray termination. Shaded rendering can be facilitated by preshading the samples using the local gradients and light source positions to resolve the dot products, but deferring the coloring of the preshaded samples to the final compositing step, which is re-run whenever the transfer function has been modified. The preshaded samples and the densities are stored in separate lists. Alternative to the preshaded samples, we may also store an index into a reflection cube [10]. To cut down on the preprocessing time, we perform lazy shading, i.e., we only shade the samples on the first encounter during a rendering step. We have also experimented with compressing the pixel runs for better storage efficiency. These results will be presented in a future paper.

4 Results

Fig. 1 shows a variety of renderings for the neghip dataset. The exploration could be pursued at 10 frames/s on at 1GHz CPU. Our interface also allows the user to specify two separate transfer functions at a time, which gives rise to two separate images. These can either be merged after completion of the rendering or during the rendering by a volume-intermixing operation [1]. Fig. 2 shows a post-rendering merge, which leaves both skin and skull focused and sharp, while still providing a semitransparent view onto the skull across the skin.

All renderings could be achieved at rates of about 10 frames/s. On the other hand, the unoptimized raycaster without using the ray sample list extension requires tens of seconds to complete an image, which would make it quite tedious to come up with good transfer functions quickly. Thus the extension implemented for this paper proofs extremely helpful for this task.

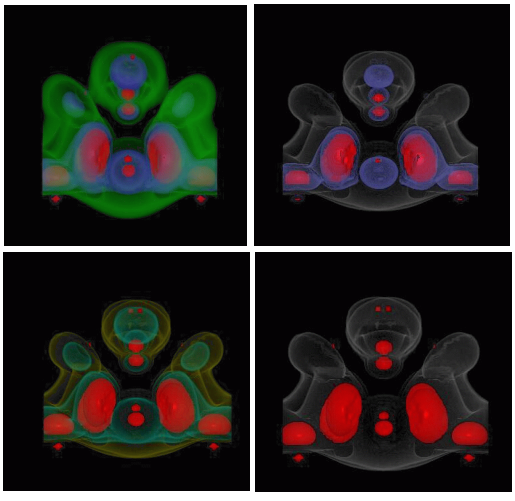

Figure 1: The neghip dataset with different transfer function settings

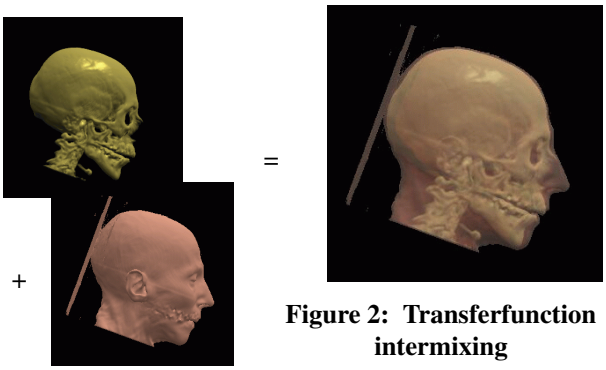

Figure 2: Transferfunction intermixing

5 Conclusions

We have presented a method that, given a fixed viewpoint and moderate pre-processing time, allows the user to explore arbitrary settings of the transfer function at near-interactive rates on a standard PC, without the need for special graphics hardware. We think that our method will have potential applications whenever texture mapping hardware is less advantageous to use or simply not available. An attractive feature of our method is that it is easy to implement and that it may incorporated and added into any volume rendering application, to provide a facility for fast transfer function pre-viewing and exploration.

References

[1] H. Hauser, L. Mroz, G. Bischi, and M. Gröller, “Two-level volume rendering,” *IEEE Trans. on Visualization and Computer Graphics,* vol. 7, no. 3, pp. 242-252, 2000.

[2] K. Kaneda, Y. Dobashi, K. Yamamoto, and H. Yamashita, “Fast volume rendering with adjustable color maps,” *1996 Symposium on Volume Visualization*, pp. 7-14, 1996.

[3] J. Kniss, G. Kindlmann, and C. Hansen, “Interactive volume rendering using multidimensional transfer functions and direct manipulation widgets, “ *Visualization’01*, pp. 255-262, 2001.

[4] P. Lacroute and M. Levoy, “Fast volume rendering using a shear-warp factorization of the viewing transformation,” *Proc. SIGGRAPH ‘94*, pp. 451- 458, 1994.

[5] M. Levoy, “Display of surfaces from volume data”, *IEEE Comp. Graph. & Appl.*, vol. 8, no. 5, pp. 29-37, 1988.

[6] K. Mueller, N. Shareef, J. Huang, and R. Crawfis, "IBR-assisted volume rendering," *Late Breaking Hot Topics of Visualization’99,* October 1999.

[7] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P.-P. Sloan, “Interactive Ray Tracing for Isosurface Rendering, “ *Proc. Visualization ’98*. pp. 233-238, 1998.

[8] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler, “The VolumePro real-time raycasting system,” *Proc. SIGGRAPH 99,* p. 251-260, Los Angeles, CA, August 1999.

[9] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl, "Interactive volume rendering on standard PC graphics hardware using multi-textures and multi-stage-rasterization" *SIGGRAPH/Eurographics Workshop on Graphics Hardware’00*, pp. 109-118, 2000.

[10] J. van Scheltinga, J. Smit, and M. Bosma, “Design of an on-chip reflectance map,” *Proc. Eurographics Workshop on Graphics Hardware’95*, pp. 51-55, 1995.

[11] R. Yagel and Z. Shi, “Accelerating volume animation by space-leaping,” *Proc.Visualization’93*, pp. 62-69, 1993.