Interactive Transfer Function Modification For Volume Rendering Using Compressed Sample Runs

Vivek Srivastava Uday Chebrolu Klaus Mueller

Center for Visual Computing, Computer Science, Stony Brook University

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 compressed list of samples along each ray. Then, each time the transfer function is modified, the algorithm traverses the compressed sample lists, decompressing the runs and compositing the newly colored samples along each ray until full opacity is reached. Since the expensive sample interpolation and shading is no longer necessary, we can obtain near-interactive framerates for a variety of datasets, while our RLE-based compression of linearly varying sample runs helps keep the storage complexity down, with little decompression overhead. Decompression cost is reduced by storing decompression results in an on-the-fly constructed 2D table.

1 Introduction

A volumetric data object is described as a space-filling three-dimensional grid of discrete sample points, which, in turn, support the interpolation of any arbitrary point within the grid's 3D bounding box. A great variety of disciplines generate, use, and modify volumetric data. Examples are the medical field in diagnosis and surgical simulation, engineering in CAD/CAM prototyping, the oil and gas industry in natural resource exploration, designers and artists in virtual sculpting and industrial design, educators in teaching biology, chemistry, and anatomy, the computer game industry in the generation of realistic natural phenomena, computational scientists in scientific data exploration, and the business world in visual data mining.

Volume rendering is the process of exploring the 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). Note that while spatial navigation also exists in the surface rendering of polygonal objects, transfer function navigation is unique to volume rendering.

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 [2][5][17][22][24]. Hardware solutions, however, are only practical for moderate dataset sizes, unless expensive machines are used [21]. On the other hand, interactive software solutions also exist, and a prominent representative for these is the shear-warp algorithm [12], which, however, has a considerable quality-speed trade-off, especially at odd viewing angles [26].

Addressing the complexities associated with the magnitude of the transfer function space, a number of techniques have been devised that use gradient histograms and other derived data to point out interesting locations in transfer function space [1][10][11][27]. The user may then use these hints to perform a more detailed search around these critical points. Another suggested strategy is to render a large number of images with arbitrary transfer function settings and present these to the user, who then selects a subset of these for further refinement by ways of genetic algorithms. This approach is taken by the Design Galleries project [16], which is based, in part, on the method published in [8]. A good sample of the existing approaches were squared off in a recent panel, termed the 'Transfer Function Bake-off', at the Visualization 2000 conference [23].

Common to all transfer function exploration efforts 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 [3][19][28], for this research we sought 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. This was recognized quite some time ago in [15]. A caveat with this approach, however, is that all samples along a ray, within the limits of the vol-

Email: {vivek, uday, mueller}@cs.sunysb.edu
Mail: Computer Science Department, State University of New York at Stony Brook, Stony Brook, NY, 11794, USA

ume's bounding box, are now potentially visible, depending on the user's choice of transfer function setting. Hence, they must all be cached, which gives rise to a storage complexity approaching that of the volume. A suitable compression scheme is clearly needed, with the requirement of fast decompression speed since our main goal is to achieve an interactive system for transfer function modification. Note that our approach uses a pure software solution in an attempt to explore a technique that would work even in conjunction with larger volume, for which the use of texture mapping hardware may be less attractive. On the other hand, there is also a sizable class of volumes with sparse, yet space-filling features, such as arteries or nerve cells, where software rendering speed can actually match that of brute-force texture mapping hardware implementations[18].

Our paper is structured as follows. First, in Section 2, we provide some preliminary background and the theoretical aspects of our work, while Section 3 describes the actual implementation. Section 4 reports on the results, and Section 5 concludes with final remarks and pointers for future work.

2 Theory

2.1 Preliminaries

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

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

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). Since the continuous integral is, in the general case, not solvable analytically, it is commonly written and evaluated in discrete form:

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

where Δs is the ray sampling interval. If we replace the exponential function by its Taylor series, we get the familiar compositing equation [13]:

$$r(u,v) = \sum_{i=0}^{L/\Delta s} c(\mathrm{d}(\Delta s))\alpha(\mathrm{d}(\Delta s))\left(\prod_{j=0}^{i-1}(1-\alpha(\mathrm{d}(\Delta t)))\right) \quad (3)$$

where α is the sample opacity, normalized for sample distance when $\Delta s \neq 1$ [14].

Let us first concentrate on the continuous form of the low-albedo volume rendering integral given in (1), and the fact that we will be re-using the d(s), i.e., the interpolated samples along the ray. This seems to be a simpler task than having to solve (1) directly from the raw volume, involving interpolation. Can we now obtain an approximate analytical solution for r(u,v), or at least a fast discrete one? We shall investigate this next.

An effective way to go about the problem is to try and decompose c, τ (and perhaps also d) into sets of basis functions, and then just modify the weights of these basis functions according to a new transfer function setting before recombination. One such decomposition is the Fourier transform, or the related cosine transform. Kaneda et.al. [9] attempted this a few years ago, but only succeeded for the color transfer function. For the Fourier approach to work, the inner integral of (1) needs to remain fixed as a multiplicative constant, and hence the opacity transfer function remains fixed as well, which prevents the Fourier method from being able to change the transparencies of exterior and interior structures on the fly.

But the Fourier series is not the only mechanism to obtain a decomposition into basis functions. Another set of basis functions can be obtained by a series of polynomials, which is described next.

2.2 Polynomial fit

We may express the transfer functions as well as the list of sample densities along a ray as polynomial functions, which could be obtained by a least squares fit. We will get:

$$\begin{aligned} c(d) &= a_n d^n + a_{n-1}d^{n-1} + \ldots + a_0 = p_1^n \\ \tau(d) &= b_n \tau^n + b_{n-1}\tau^{n-1} + \ldots + b_0 = p_2^n \\ d(s) &= e_n s^n + e_{n-1}s^{n-1} + \ldots + e_0 = p_3^n \end{aligned} \quad (4)$$

Recall that c is a 3-vector, and so will be the a-coefficients. Note that we will only need to find the coefficients for the c(d) and τ(d) on the fly, the coefficients for d(s), on the other hand, will be computed in a pre-processing step. We shall now express (1) with these polynomials:

$$r(u,v) = \int_0^L p_1^n(p_3^n(s)) \cdot p_2^n(p_3^n(s))\exp\left(-\int_0^s p_2^n(p_3^n(t))dt\right)ds \quad (5)$$

$$r(u,v) = \int_0^L p_4^{2n}(s) \cdot p_5^{2n}(s)\exp\left(-\int_0^s p_6^{2n}(t)dt\right)ds \quad (6)$$

$$r(u,v) = \int_0^L p_7^{4n}(s)\exp(-p_8^{2n+1}(s))ds \quad (7)$$

Using the first two terms of the Taylor series expansion of exp(x):

$$r(u,v) = \int_0^L p_7^{4n}(s)(1-p_8^{2n+1}(s))ds \quad (8)$$

$$r(u,v) = \int_0^L p_9^{6n+1}(s)ds = p_{10}^{6n+2}(L) \quad (9)$$

We obtain a polynomial of degree (6n+2) that we could evaluate for L to get the solution for (1). To build the polynomial we would encounter costs of O(n), the degree of the initial polynomial. Required storage per ray is also just

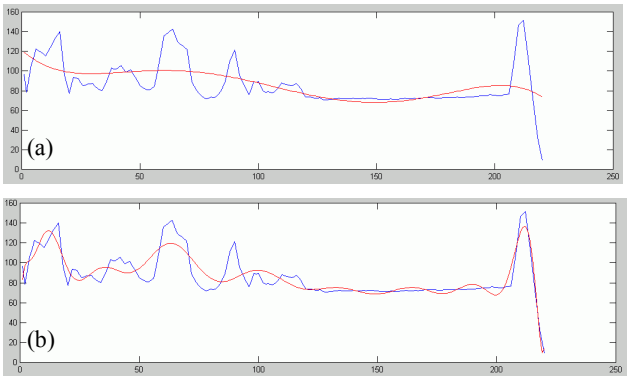

Figure 1: Polynomial fits (red) for a run of samples (blue) across the neck area of the UNC CT Head dataset. In (a) the degree of the polynomial is 3, in (b) degree is 20. Given the variability of the data, even polynomials of higher degree don't provide sufficient fits in the general case.

O(n), the coefficients of the polynomials. Thus a large compression could be the gain. However, the problem with the polynomial approach is that there is a high chance that only a high-order polynomial can provide a good fit to a given ray sample sequence, and the same is true, however, in a less severe way, for the transfer functions. See, for example, Fig. 1, where we show a typical ray sample sequence with two fitted polynomials, one of degree 3 and one of degree 20. The latter provides a somewhat adequate fit, but a higher degree would probably be better. Unfortunately though, a polynomial of large degree will not reduce the rendering complexity much and also diminish the compression gains greatly. In addition, high-order polynomials may exhibit "Runge's Phenomena" - fast oscillations at discontinuities. Hence, it does not seem that the continuous polynomial approach offers useful benefits for our application, at least not in the general case.

To get a better fit, it seems more advisable to use piecewise polynomials instead, for either transfer functions or the rays, or both. This potentially breaks the integral of (1) into many separate pieces, which will result, in the limit, in the discretized form of (3). However, it is likely that the pieces are sufficiently large, and a good amount of compression can be achieved, while maintaining a good fit. Using the piece-wise linear polynomial fit can provide a much better compression than a straightforward RLE encoding, as was done in [4]. We shall now outline our chosen approach which uses piecewise linear polynomials for sample run compression.

3 Implementation

We first describe the basic algorithm that uses piecewise polynomials for run compression, and then we shall describe a few enhancements that provides extra speed for the decompression of these runs.

3.1 Basic algorithm

Our algorithm first acquires a list of samples along each ray at unit sampling distance. Suppose, for this discussion,

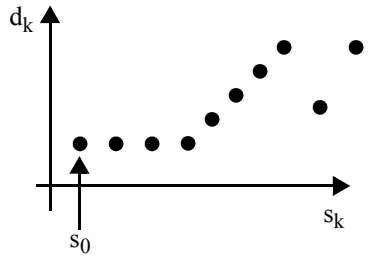

Figure 2: Sample run encoding example.

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 location along the run, and the d_k are the interpolated densities at each s_k, rounded to integers (see Fig. 2 for an illustration of an example density profile). We now seek to fit linear polynomials that span more than 2 samples and interpolate the endpoints. Suppose that a set of current endpoints is (s_i,d_i) and (s_j,d_j), respectively, where s_j-s_i>1. We require that for each sample k for which i<k<j:

$$d_k - \frac{(d_j - d_i)}{(s_j - s_i)} \cdot (s_k - s_i) < errorThreshold \qquad (10)$$

If errorThreshold<1, then we are within the discretization error of the transfer function indexing. Greater error thresholds will yield larger transfer function index errors and will result in the retrieval of a potentially inaccurate color and opacity from the transfer function. However, since the index is still close to the correct one, the error may not be large, unless the transfer function has high frequencies.

For each sample run, our algorithm starts with the sample at s_0 and then tries to expand a linear polynomial until the error for any sample between the current start and end point of the polynomial is greater than the error threshold. Once this happens the polynomial is stored away and a new one is started. If the length of the resulting polynomial is only 1 (that is, no expansion was possible), then the search is started from the second sample and the first is stored in a separate format, which will be explained in the course of the next paragraph.

Our compression format is as follows. We distinguish between sample runs of length > 1, fitting a linear polynomial within the pre-specified error threshold, and sample runs that cannot fit such an approximation. The fitted runs require 2 bytes of storage: length and end density. The non-fitted runs are stored as a list of samples, with a leading byte that indicates the length of the run. We use the leading bit of each length-byte to encode the type of the run, LIST or POLY. Note that 4 consecutive samples that do not fit a single POLY will require 5 bytes of storage as a LIST run, but 3·2=6 bytes of storage if encoded as 3 POLYs. On the other hand, a POLY of length 3 will require 2 bytes of storage, while a LIST will use 4 bytes. Thus the POLY is more efficient for fitted runs of length greater than 2. If a run exceeds a length of 128, which would not be able to fit the space allocated to the 'length' byte, we simply start a new run. Returning to Fig. 2, the encoding for this sequence would be as follows:

d_0 {POLY3, d_3} {POLY4, d_7} {LIST2, d_8, d_9}

which 8 bytes of storage, while a linear encoding would have taken 10 bytes. Note, that the value of the first sample is always stored explicitly, since the POLY stores only the density of the end point.

Upon rendering, we traverse the encoded list and either use the LIST densities in sequence or compute the values of the samples covered by the current linear polynomial. For the latter, the slope is given by the ratio of the start and end densities and the length of the curve. We use the interpolated and rounded densities to look up the colors and opacities in the current transfer functions. Finally, we composite the samples front-to-back with early ray termination.

Note that the previous scheme only works for unshaded samples, which is similar to the method devised in [9]. However, we would also like to provide shaded displays. To facilitate this, we provide a second rendering mode, where we preshade the samples using the local gradients and light source positions to resolve the dot products, but defer 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 run-encoded lists, which are traversed simultaneously during rendering. Typically, the compression of the preshaded sample list is less than that of the density runs, due its sensitivity to gradients. Alternative to the preshaded samples, we may also store an index into a reflection cube [25].

3.2 Enhancements

One enhancement of our algorithm is to perform lazy shading, that is, only shade the samples on the first encounter during a rendering step. This reduces start-up time, which will matter most for larger volumes and iso-surface rendering where rays turn quickly opaque.

Further speed-ups can be obtained by caching already composited POLY-run colors and opacities of partial rays. This way, the decompression and compositing has to be done only once per density interval, for a given transfer function setting. Since the compositing result depends on the length of the run, we would need a 3D table, indexed by the start density d_s, the end density d_e, and the run length L. Since this would give rise to a fairly large table, we decided to use a 2D table instead, coding only for the start and end densities, with a fixed run length. Similar to [6] we can approximate the compositing result for deviating sample run lengths by simple scaling:

$$c = c_{table} \cdot \frac{L_{current}}{L_{table}} \qquad \alpha = \alpha_{table} \cdot \frac{L_{current}}{L_{table}} \tag{11}$$

where $L_{current}$ is the length of the currently encountered sample run, while L_{table} is the length of the composited run cached in the table. We have found that this approximation fails to produce useful images once the L-ratio becomes too large. We cope with this in two ways. First, as a preprocessing step, we find the average sample run L_{avg} for each d_s-d_e pair. We then store the compositing result for this L_{avg} in the table. Since the L_{avg} may vary among the d_s-d_e pairs we need a separate table to store the L_{avg} to be used as L_{table} in (11) during decompression. Second, whenever the ratio $L_{current}/L_{table}=L_{current}/L_{avg}$ exceeds a certain value provided by the user (3 seems to work well), we composite the run in an n-step process, using n cached results in the table. As an illustration, let us assume that the current run d_s-d_e encompasses 7 ray samples, but $L_{avg}[d_s\text{-}d_e]$ is only 3. Then we try to break up the run into 2 pieces and perform the following compositing operation of two cached partial rays of length $L_{s\text{-}e1}$ and $L_{e1\text{-}e}$:

$$c = c_{table}[d_s, d_{e1}]\frac{L_{s-e1}}{L_{avg}[d_s, d_{e1}]} + c_{table}[d_{e1}, d_e]\frac{L_{e1-e}}{L_{avg}[d_{e1}, d_e]} \tag{12}$$

The α-evaluation is similar. If we are left with a partial ray of $L_{s\text{-}e} \ll L_{table}$ we just evaluate the samples directly, without using the table.

Currently our method fills the entire table beforehand, however, only the density pairs affected by the transfer function will be updated after the first run. We could also use a lazy scheme where one calculates a table entry only upon the first time it is encountered after a local transfer function change. In that case, all table entries affected by a transfer function modification would have to be marked "invalid".

Finally, we have observed that we can considerably improve the lengths of the sample runs (and therefore the compression) by a slight smoothing or median smoothing of the density (and pre-shaded) runs. Longer runs are favorable since the initial setup cost for a run decompression amortizes better for longer runs.

4 Results

We use a suite of standard datasets (foot, CT head, lobster, neghip) to demonstrate our algorithm. The datasets are all about 256^3 in size, with exception of the neghip dataset which is 64^3. Our (unoptimized) raycaster without using the compressed ray sample list extension requires tens of seconds to complete an unmagnified image, which would make it quite tedious to come up with good transfer functions quickly. On the other hand, our raycast renderings based on sample runs can be achieved at rates of about 10 frames a second and more, which proofs the algorithm quite helpful for the task intended. The pre-processing needed to construct the ray sample list requires about 30 seconds, but this step is only needed once for a fixed view. We shall now report detailed results we have obtained.

Fig 3 shows the reduction of the data points due to the piece-wise linear polynomial compression, for three preset error thresholds. We re-use the ray sample sequence introduced in Fig. 1. It can be seen that the ray compresses significantly, even for small error thresholds, but yet a good fit is maintained.

The bottom row of Fig. 5 (colorplate) shows the foot dataset rendered with sample runs that were allowed to have various degrees of compression errors. We observe that even with an error threshold of 5 (see right-most image in that row), the image quality is not significantly decreased, although the storage is reduced by a factor of 22 (see Table 1). Compare this number with gzip, a lossless compression

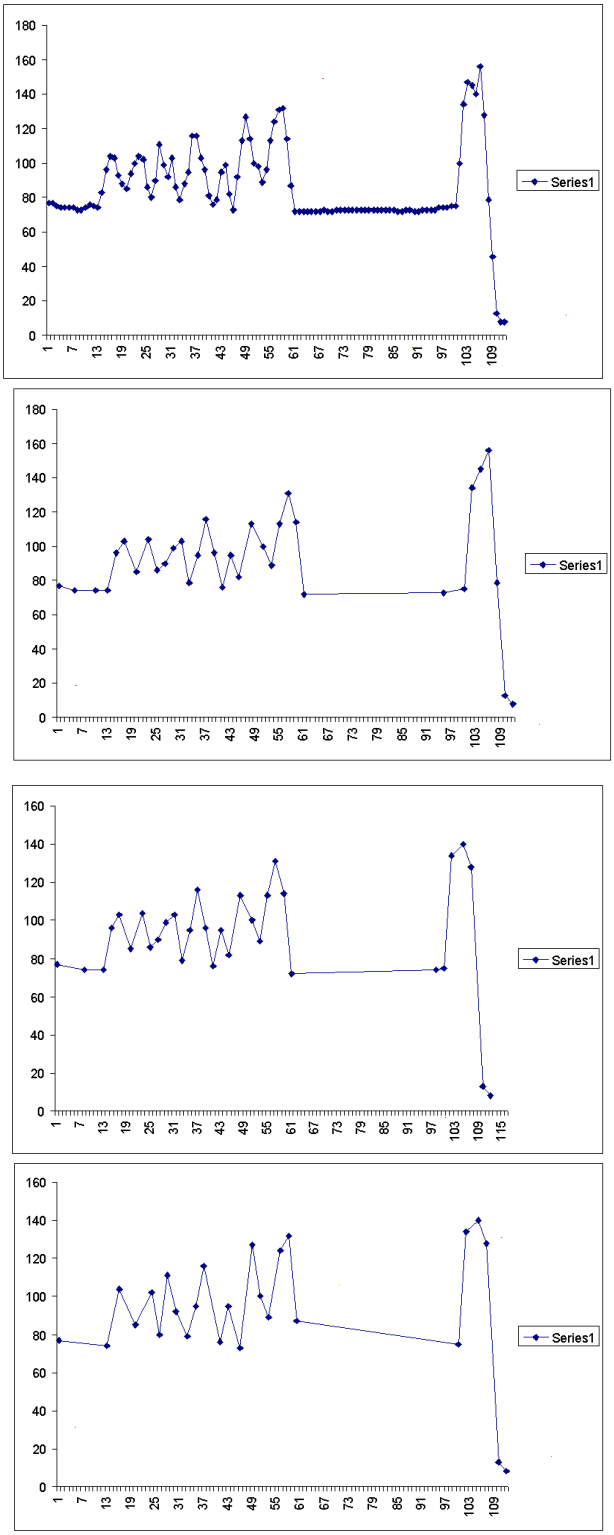

Figure 3: The effect of error threshold on the number of support points for the piecewise linear polynomial compression. We re-use the run of samples of Fig. 1 for illustration. From top to bottom: original run of samples, error threshold =1, 3, 5. The dots mark the start and end points for the polynomials (POLY), or the list points (LIST).

scheme, that only yields a factor of 3-4. It appears that the loss in information does not really affect the result, perhaps due to our specific application. In most cases, the transfer function identified with the compressed sample set can be used unchanged with the (only RLE-encoded) sample set associated with error threshold 0. In other cases, very little tweaking is required to match the visual result. Hence, the method appears to be a very suitable transfer function previewer. Notice also that the RLE-encoding alone (error thershold=0), as done in [4], does not yield very high compression rates.

The speed decreases for the rendering from compressed runs, due to the decompression overhead. We can regain the speed by using the cache table. Using tables increases memory usage, of course, but the tables also yield speedups of almost 2 when compared to the uncompressed rendering.

For the shaded renderings we don't use tables since we have two compressed runs here: (i) density runs for color and opacity transfer function look-up and (ii) pre-shaded samples. These two unsynchronized runs would make caching ambiguous. We observe that shaded rendering reduces the compression ratio and speed. The former is understandable since the dot product in the pre-shaded runs depends on the gradient which is bound to fluctuate more then the density. We tried to decrease this effect by placing both eye and light at infinity. By median-smoothing the shaded runs we were able to almost double the compression ratio and also slightly improve the speed. The speed itself is similar for the both rendering from compressed and from uncompressed runs, but significantly lower than for the unshaded renderings. Fortunately, we have observed that shaded rendering is not an absolute necessity for transfer function design (see Fig. 4 and Fig. 5b, f), one can get a good idea for the colors and occlusions just by using faster unshaded rendering.

The remaining image sets shown in Fig. 5 (colorplate) were obtained with the error threshold set to 5. The associated timings and space complexities are listed in Table 2. Fig. 5a shows the Bonsai tree rendered with different transfer functions, one for each season. The transfer function modifications could be performed at a rate of 5-7 modifications a second. Fig. 5b shows a variety of renderings for the neghip dataset. The exploration could be pursued at similar speeds (even for larger magnifications). Fig. 5c shows a mixed rendering of the skull and the skin of the CT Head dataset. Our interface 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. Fig. 5c shows a post-rendering merge, which

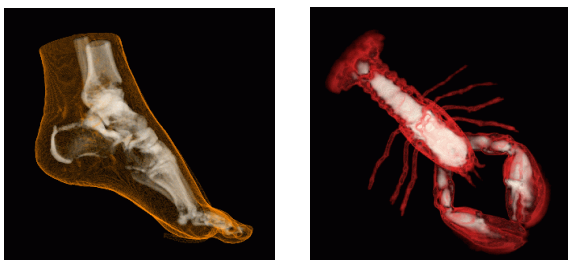

Figure 4: Unshaded foot and lobster dataset.

Volume	Unshaded rendering					Shaded rendering			
	Error threshold	Speed (ms) (no compr.)	Comp. ratio	Speed (ms) (with compr.)	Speed (w/tables)	Error threshold	Speed (ms) (no compr.)	Compr. ratio	Speed (ms) (with compr.)
Foot	8	47	29.7	47	20	0.3	120	3.3	150
	5	47	22.5	60	20	0.2	120	2.3	170
	1	47	5.7	125	50	0.1	120	1.6	250
	0	47	2.4	70	78	0.05	120	1.4	187
Foot (median smoothed runs)	8	47	37.7	47	20	0.3	120	6.2	130
	5	47	30.5	47	20	0.2	120	4.0	140
	1	47	11.4	94	30	0.1	120	2.5	187
	0	47	3.2	80	60	0.05	120	2.0	170
CT-head	8	80	11.0	140	50	0.3	250	5.4	300
	5	80	8.0	150	50	0.2	250	3.6	300
	1	90	3.4	170	70	0.1	250	2.3	350
	0	90	1.5	110	110	0.05	250	1.8	280
CT-head (median smoothed runs)	8	80	13.8	125	50	0.3	250	7.3	300
	5	80	10.0	140	50	0.2	250	4.6	320
	1	80	4.2	170	60	0.1	250	2.7	350
	0	80	1.9	120	100	0.05	250	2.1	300
Neghip	8	15	9.5	15	10	0.3	30	10.7	30
	5	15	7.5	20	10	0.2	30	6.8	30
	1	15	4.0	20	10	0.1	30	4.1	45
	0	15	2.0	15	10	0.05	30	2.6	30

Table 1: Comparison of compressions and timings for a variety of datasets.

leaves both skin and skull focused and sharp, while still providing a semitransparent view onto the skull across the skin. This is similar to the spectral volume rendering approach put forward by Noordman's et. al. [20]. Finally, Fig. 5d, e, and f show two more renderings of the CT Head and one each from the lobster and the neghip dataset. For the CT Head image on the right, we smoothed the sample runs with a 3-point median filter. This achieved an interesting softening effect in the volume rendering, somewhat reminiscent to non-photo realism, as well as an improvement in compression and performance.

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. Examples for the former are large or sparse datasets, or high-precision rendering in the presence of objects with low color and low opacities, for which the product of the two quantities requires more than the 8 bits provided by the graphics hardware framebuffer. Our method also allows different modes of volume rendering applied during compositing. For example, one could easily implement the two-level volume rendering approach of [7] in which MIP, X-ray, and direct volume rendering are mixed along a ray. 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.

Acknowledgments

This work was partially supported by NSF Career grant ACI- 0093157.

References

[1] C. Bajaj, V. Pascucci, and D. Schickore, “The contour spectrum,” *Proc. Visualization’97*, pp. 167-173, 1997.

[2] B. Cabral, N. Cam, and J. Foran, “Accelerated volume rendering and tomographic reconstruction using texture mapping hardware”, *Symp. on Volume Visualization’94*, pp. 91-98, 1994.

[3] B. Chen, A. Kaufman, Q. Tang, “Image-based rendering of surfaces from volume data,” *Volume Graphics Workshop 2001*, pp. 279-295, 2001.

[4] M. Cohen, J. Painter, M. Metha, K.-L. Ma, “Volume seedlings,” *1992 Symposium on Interactive 3D Graphics (I3D)*, pp. 139-145, 1992.

[5] K. Engel, M. Kraus, and T. Ertl, "High-quality pre-integrated volume rendering using hardware-accelerated pixel shading," *Proc. SIGGRAPH Graphics Hardware Workshop’01*, pp. 9-16, 2001.

[6] S. Guthe. S. Roettger, A. Schieber, W. Strasser, and T. Ertl, “High-Quality Unstructured Volume Rendering on the PC Platfo‘rm,” *ACM Siggraph/Eurographics Hardware Workshop*, 2002.

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

[8] T. He, L. Hong, A. Kaufman, and H. Pfister, “Generation of Transfer Functions with Stochastic Search Techniques,” *Proc. Visualization’96*, pp. 227-234, 1996.

[9] 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.

[10] G. Kindlmann and J. Durkin, “Semi-automatic generation of transfer functions for direct volume rendering, “ *Symp. Volume Visualization’98*, pp. 79-86, 1998.

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

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

[13] M. Levoy, “Display of surfaces from volume data”, *IEEE Computer Graphics. & Applications*, vol. 8, no. 5, pp. 29-37, 1988.

[14] B. Lichtenbelt, R. Crane, and S. Naqvi, *Volume Rendering,* Prentice-Hall, 1998.

[15] K.-L. Ma, M. Cohen, and J. Painter, “Volume seeds: a volume exploration technique,” *The Journal of Computer Graphics and Visualization*, vol. 2, pp. 135-140, 1991.

[16] J. Marks, et.al., Design Galleries: A General Approach to Setting Parameters for Computer Graphics and Animation,“ *Proc. SIGGRAPH 97*, pp. 389-400, 1997.

[17] M. Meißner, U. Kanus, and W. Straßer, "VIZARD II: A PCI-card for Real-Time Volume Rendering", *Proc. Siggraph/ Eurographics Workshop on Graphics Hardware’98,* pp. 61--67, 1998.

[18] M. Meißner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis, "A practical comparison of popular volume rendering algorithms," *Proc. 2000 Symp. on Volume Rendering*, pp. 81-90, October 2000.

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

[20] H.Noordmans, H. van der Hort, and A. Smeulders, “Spectral volume rendering,” *IEEE. Trans. Visualization and Computer Graphics*, vol. 6, no. 3, pp. 196-207, 2000.

[21] 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.

[22] 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.

[23] H. Pfister, B. Lorensen, C. Bajaj, G. Kindlmann, W. Schroeder, L. Avila, K. Martin, R. Machiraju, and J. Lee, “The transfer function bake-off,” *IEEE Computer Graphics & Applications,* vol. 21, no. 3) pp. 16-22, 2001.

[24] 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" *Proc. SIGGRAPH/Eurographics Workshop on Graphics Hardware’00*, pp. 109-118, 2000.

[25] 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.

[26] J. Sweeney and K. Mueller, "Shear-Warp Deluxe: The Shear-Warp algorithm revisited," *Joint Eurographics - IEEE TCVG Symposium on Visualization 2002*, May 2002.

[27] S. Tenginakai, J. Lee, and R. Machiraju. “Salient iso-surface detection with model-independent statistical signatures,” *Proc. Visualization’01*. pp. 231-238, 2001.

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

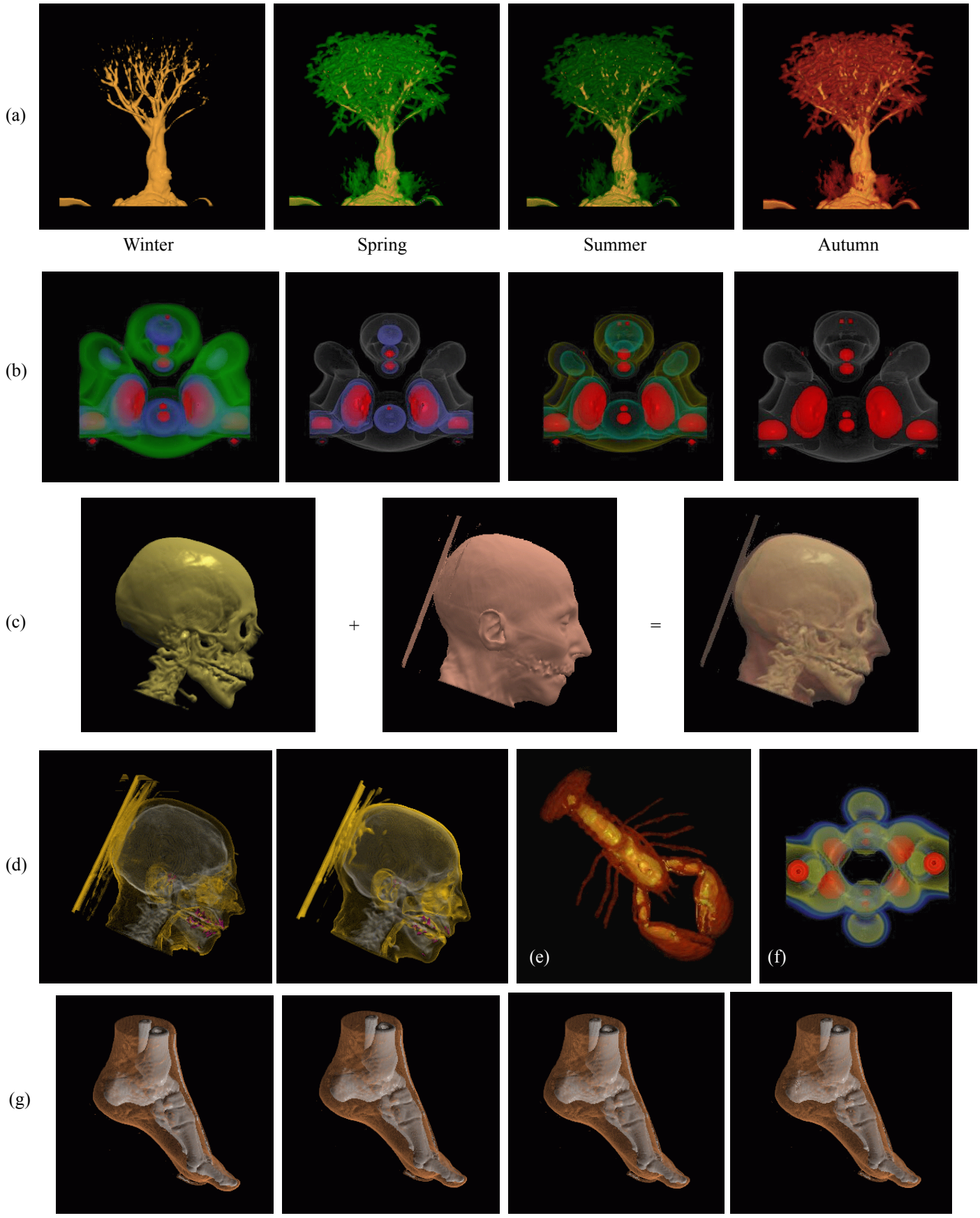

Figure 5: (a) Bonsai tree dataset rendered in different "seasons", (b) Neghip dataset rendered with different transfer functions, (c) UNC CT Head rendered with two different transfer functions and resulting images added together. This leaves both skull and skin sharp. (d) CT Head rendered from original sample runs (left) and median smoothed runs (right), (e) lobster dataset, (f) neghip from a different viewing angle, (g) foot rendered with different error thresholds for run compression (error =0, 1, 3, 5).