IEEE/ EG Symposium on Volume and Point-Based Graphics (2008)
H.- C. Hege, D. Laidlaw, R. Pajarola, O. Staadt (Editors)

Pre-Integrated Volume Rendering for Multi-Dimensional Transfer Functions

M. Kraus

Computer Graphics & Visualization Group, Technische Universität München

Abstract

This work presents an efficient extension of pre-integrated volume rendering for multi-dimensional transfer functions. Since even two-dimensional transfer functions can result in strong rendering artifacts that cannot be resolved by standard ray casting, we propose a space-covering volume sampling scheme that does not require additional samples of the volume data. Based on this sampling scheme, our approach computes box integrals of the transfer functions, which are evaluated with the help of a small number of texture lookups in tabulated integral functions. We demonstrate the achieved performance and improvements in image quality with a prototypical GPU (graphics processing unit) implementation.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

While pre-integrated volume rendering [EKE01] has been successfully combined with many direct volume rendering algorithms [KE04], its application to multi-dimensional transfer functions has remained difficult. Previously published approaches [KPI*03] are restricted to particular transfer functions and rendering algorithms; thus, they lack the general applicability of pre-integrated volume rendering.

On the other hand, multi-dimensional transfer functions have been employed in some of the earliest works on volume visualization [Lev88] and continue to attract research interest [KD98, KKH02]. Thus, the application for pre-integration to multi-dimensional transfer functions was considered one of the main open questions in pre-integrated volume rendering [KE04].

The main technical challenge of a naive application of pre-integrated volume rendering to multi-dimensional transfer functions is the dimensionality of the required pre-integrated lookup tables since a one-dimensional transfer function already requries at least a two-dimensional lookup table (for constant sampling distance) [EKE01], and a two-dimensional transfer function would require at least a four-dimensional lookup table, which is usually of prohibitive size.

We address this problem by the use of integral functions of the same dimension as the transfer functions. In the case of two-dimensional transfer functions, this allows for an efficient implementation on modern GPUs based on floating-point textures [HSC*05]. A similar approach was applied to accelerate the computation of pre-integrated lookup tables [EKE01]; however, the previously employed approximations for opacities and colors were only applicable under certain restrictions and resulted in severe approximation errors for general transfer functions and large sampling distances.

To overcome this problem, we propose a new formula for the approximation of colors in Section 3. Furthermore, we present a variant that is more suitable for the rendering of isosurfaces [Lev88, Kra05] and generalize this approach to multi-dimensional transfer functions.

The second major problem posed by general multi-dimensional transfer functions are strong aliasing artifacts. For example, opacity and color can be assigned to the intersection of two isosurfaces by specifying a sharp point-like peak in a two-dimensional transfer function. This intersection can correspond to an arbitrarily fine line in the volume data, which cannot be rendered by standard volume ray casting algorithms. On the other hand, previously published vol-

© The Eurographics Association 2008.

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

ume antialiasing techniques [YMC06] are likely to result in excessive blurring in these particular cases.

In order to allow for the possibility of rendering unblurred lines (and points for three-dimensional transfer functions), we propose a space-covering sampling scheme in Section 4. Thus, the possibility of rendering arbitrarily thin isosurfaces with the help of one-dimensional pre-integrated volume rendering [EKE01] is successfully generalized to lower-dimensional point sets for higher-dimensional transfer functions. Furthermore, we present a GPU-based technique for buffering volume samples to avoid the need for additional samples of the volume data, which are particularly costly for compressed data sets [FAM*05].

Results obtained with a prototypical GPU implementation of the proposed techniques are presented in Section 5. Before we present our research, however, related work is discussed in the next section.

2. Related Work

Volume rendering with particular multi-dimensional transfer functions was proposed by Levoy [Lev88] to improve the rendering of surfaces in volume data sets. More general multi-dimensional transfer functions were suggested later, for example by Kindlmann and Durkin [KD98] and Kniss et al. [KKH02]. Kniss et al. [KPI*03] also proposed an extension of pre-integrated volume rendering for multi-dimensional transfer functions, which however was limited to transfer functions specified by Gaussian primitives.

Pre-integrated volume rendering itself is related to a technique that was first proposed by Max et al. [MHC90] and later extended and applied to hardware-accelerated tetrahedra projection by Roettger et al. [RKE00]. The application to volume rendering of three-dimensional textures was first suggested by Engel et al. [EKE01]. The common feature of these techniques is a tabulation of the volume rendering integral such that a potentially expensive integration is replaced by a table lookup. A more detailed survey is provided by Kraus and Ertl [KE04]. Pre-integrated volume rendering is still an active research topic; for example, Lum et al. [LWM04] have shown how to pre-integrated Phong-lighting and Kye et al. [KHS05] have presented an alternative approximation of opacities based on integral functions.

There are also recent research results on the volume rendering integral and its evaluation. For example, Kraus [Kra05] derived Levoy's gradient factor for surface rendering [Lev88] from a scale-invariant volume integral with additional silhouette enhancement, and an efficient method for antialiased rendering of multi-resolution volume hierarchies was suggested by Younesy et al. [YMC06].

The increasing flexibility and programmability of graphics hardware in the form of GPUs has led to numerous new methods in real-time rendering and hardware-accelerated volume rendering. For example, on-the-fly decompression of volume data has been proposed by Fout et al. [FAM*05]. Thus, data transfer between main memory and graphics memory can be avoided by storing compressed data in the graphics memory. However, the required volume decompression tends to result in more costly volume sampling operations.

Texture mapping with summed-area tables, which was introduced by Crow [Cro84], is another example of a technique that was recently enabled by GPUs as demonstrated by Hensley et al. [HSC*05].

3. Volume Pre-Integration Based on Integral Functions

This section discusses the mathematical foundation of our approach. In particular, we derive new approximations for opacities and colors—first in one dimension and then for multi-dimensional transfer functions. In Section 4, we will describe our algorithm for an efficient evaluation of the proposed integrals.

3.1. Standard Volume Rendering Integral

For the derivation of our new formulas for pre-integrated opacities and colors, we will adhere quite closely to the notation of Engel et al. [EKE01] and Kraus and Ertl [KE04]. In particular, we will consider a transfer function $\tau(s) \in [0,\infty[$ for the extinction coefficient and a color transfer function $c(s)$ that has to be multiplied with $\tau(s)$ to yield an actual color intensity $\tau(s)\,c(s)$ for a given scalar value s.

Pre-integrated volume rendering is based on precomputing a lookup table of opacities and colors for a set of parameter tuples (s_f, s_b, d) where s_f and s_b denote the scalar value at the front and back of a short segment of a view ray and d specifies the spatial length of this segment. The main assumption of pre-integrated volume rendering is that the scalar data along the segment is well approximated by a linear interpolation between s_f and s_b. In this case, the volume rendering integral for a segment characterized by (s_f, s_b, d) can be approximated by a lookup in the precomputed table. The opacities $\alpha(s_f, s_b, d)$ and colors $\tilde{C}(s_f, s_b, d)$ in this table are computed as:

$$\alpha(s_f, s_b, d) \stackrel{\text{def}}{=} 1 - \exp\left(-\int_0^1 \tau((1-\omega)\,s_f + \omega s_b)\,d\,d\omega\right), \tag{1}$$

$$\tilde{C}(s_f, s_b, d) \stackrel{\text{def}}{=} \int_0^1 \tau((1-\omega)\,s_f + \omega s_b)\,c((1-\omega)\,s_f + \omega s_b) \times \exp\left(-\int_0^\omega \tau\left((1-\omega')\,s_f + \omega' s_b\right) d\,d\omega'\right) d\,d\omega. \tag{2}$$

To accelerate the computation of these integrals, Engel et al. [EKE01] suggested the following equations, which neglect self-attenuation of colors:

$$\alpha(s_f, s_b, d) \;=\; 1 - \exp\left(\frac{-d}{s_b - s_f}\int_{s_f}^{s_b} \tau(s)\,\mathrm{d}s\right) \tag{3}$$

© The Eurographics Association 2008.

$$= 1 - \exp\left(\frac{-d}{s_\mathrm{b}-s_\mathrm{f}}\left(T(s_\mathrm{b}) - T(s_\mathrm{f})\right)\right), \quad (4)$$

$$\tilde{C}(s_\mathrm{f}, s_\mathrm{b}, d) \approx \frac{d}{s_\mathrm{b}-s_\mathrm{f}} \int_{s_\mathrm{f}}^{s_\mathrm{b}} \tau(s)\, c(s)\, \mathrm{d}s \quad (5)$$

$$= \frac{d}{s_\mathrm{b}-s_\mathrm{f}}\left(K(s_\mathrm{b}) - K(s_\mathrm{f})\right). \quad (6)$$

with the integral functions $T(s) \stackrel{\mathrm{def}}{=} \int_0^s \tau(s')\,\mathrm{d}s'$ and $K(s) \stackrel{\mathrm{def}}{=} \int_0^s \tau(s')\, c(s')\,\mathrm{d}s'$.

While these equations were originally proposed to accelerate the precomputation of lookup tables in software, they can also be implemented on modern GPUs as the exponential function can be evaluated efficiently in fragment programs and the integral functions may be implemented with the help of floating-point textures, which provide sufficient precision. The advantage of using lookup tables for the one-dimensional integral functions $T(s)$ and $K(s)$ compared to the three-dimensional lookup tables for $\alpha(s_\mathrm{f}, s_\mathrm{b}, d)$ and $\tilde{C}(s_\mathrm{f}, s_\mathrm{b}, d)$ (or two-dimensional tables for constant d) is the strongly reduced memory requirement. This is even more important in the case of multi-dimensional transfer functions, as we will see in Section 3.3.

On the other hand, the approximation for $\tilde{C}(s_\mathrm{f}, s_\mathrm{b}, d)$ is difficult to evaluate accurately for $s_\mathrm{f} \approx s_\mathrm{b}$ and diverges for $d \to \infty$ while it should converge to an (opaque) color. Thus, the approximation is only valid for $d\tau(s) \ll 1$. Therefore, we propose a new approximation for $\tilde{C}(s_\mathrm{f}, s_\mathrm{b}, d)$, which converges to a constant color for large d (as α converges to 1) and can be evaluated as efficiently as the previously proposed approximation. Moreover, the new formula allows for a numerically robust implementation.

We derive our new approximation by splitting the integral for $\tilde{C}(s_\mathrm{f}, s_\mathrm{b}, d)$ in Equation 2 into the product of a weighted average of the colors between s_f and s_b and the volume rendering integral for a constant color:

$$\tilde{C}(s_\mathrm{f}, s_\mathrm{b}, d) \approx \frac{\int_0^1 \tau((1-\omega)s_\mathrm{f}+\omega s_\mathrm{b})c((1-\omega)s_\mathrm{f}+\omega s_\mathrm{b})d\mathrm{d}\omega}{\int_0^1 \tau((1-\omega)s_\mathrm{f}+\omega s_\mathrm{b})d\mathrm{d}\omega} \times \int_0^1 \tau\left((1-\omega)s_\mathrm{f}+\omega s_\mathrm{b}\right) \times \exp\left(-\int_0^\omega \tau\left(\left(1-\omega'\right)s_\mathrm{f}+\omega' s_\mathrm{b}\right)d\mathrm{d}\omega'\right) d\mathrm{d}\omega \quad (7)$$

$$= \frac{|K(s_\mathrm{b})-K(s_\mathrm{f})|}{|T(s_\mathrm{b})-T(s_\mathrm{f})|}\,\alpha(s_\mathrm{f}, s_\mathrm{b}, d). \quad (8)$$

For a robust evaluation, we increase the denominator by a small constant $0 \lesssim \varepsilon \ll 1$; e.g., 10^{-10}. This hardly affects the result for $|T(s_\mathrm{b}) - T(s_\mathrm{f})| \gg \varepsilon$ while it decreases the ratio for $|T(s_\mathrm{b}) - T(s_\mathrm{f})| \approx \varepsilon$. However, this decrease results only in a small absolute error of $\tilde{C}(s_\mathrm{f}, s_\mathrm{b}, d)$ since α is approximately zero in this case.

Thus, we achieve a numerically robust approximation of $\tilde{C}(s_\mathrm{f}, s_\mathrm{b}, d)$, which converges to an averaged color for large d, i.e., for $\alpha \approx 1$. Moreover, we can replace the three- or two-dimensional lookup tables by one-dimensional tabulated integral functions, which require significantly less memory. Before generalizing this technique for multi-dimensional transfer functions, an important variant for isosurface rendering is discussed in the next section.

3.2. Gradient-Dependency for Surface Rendering

Using direct volume rendering for (iso)surface rendering was investigated, for example, by Levoy [Lev88], Kindlmann and Durkin [KD98], and Kraus [Kra05]. Various dependencies of the local opacity on the gradient of the scalar data have been suggested to render surfaces since they can become transparent at positions of high gradient magnitude if this dependency is missing.

Here we follow the work by Kraus [Kra05], which suggests that a scaling of the extinction coefficient by the gradient magnitude of the scalar data is a good approximation to render isosurfaces with gradient-independent opacity and silhouette enhancement. To include this modification in our approach, we only have to replace the computation of α by:

$$\alpha(s_\mathrm{f}, s_\mathrm{b}, d, |\nabla s|) = 1 - \exp\left(\frac{-d}{s_\mathrm{b}-s_\mathrm{f}}\left(T(s_\mathrm{b}) - T(s_\mathrm{f})\right)|\nabla s|\right). \quad (9)$$

This value of α is also used to multiply colors in Equation 8; thus, no additional modifications of the computation of colors is necessary.

In order to combine standard volume rendering and surface rendering in a single visualization, we propose an additional, user-specified transfer function $\chi(s) \in [0,1]$ for the "surfaceness" at the scalar value s. A surfaceness of 0 specifies the absence of any surface, i.e., standard volume rendering, while a surfaceness of 1 specifies a surface at s, i.e., full gradient-dependent scaling. Thus, α is computed by:

$$\alpha(s_\mathrm{f}, s_\mathrm{b}, d, |\nabla s|) = 1 - \exp\left(\frac{-d}{s_\mathrm{b}-s_\mathrm{f}}\left(T(s_\mathrm{b}) - T(s_\mathrm{f})\right) \times \left(1 - \chi\left(\frac{s_\mathrm{f}+s_\mathrm{b}}{2}\right)\left(1 - |\nabla s|\right)\right)\right). \quad (10)$$

Scalings of the opacity by other functions of the gradient $f(\nabla s)$ (instead of $|\nabla s|$) should be integrated analogously since this guarantees that the transparency α stays within the interval $[0,1]$ and that colors are changed consistently.

3.3. Multi-Dimensional Transfer Functions

In this section, the proposed approximations are generalized for two-dimensional transfer functions. The case of higher-dimensional transfer functions is straightforward but the equations become rather cumbersome.

Given two scalar fields s_1 and s_2 as well as two-dimensional transfer functions $\tau(s_1, s_2)$ and $c(s_1, s_2)$, we want to compute an opacity and color for any tuple $(s_{1,\mathrm{f}}, s_{2,\mathrm{f}}, s_{1,\mathrm{b}}, s_{2,\mathrm{b}}, d)$ were $s_{1,\mathrm{f}}$ and $s_{2,\mathrm{f}}$ are the samples at the front of the segment of the view ray, $s_{1,\mathrm{b}}$ and $s_{2,\mathrm{b}}$ are the samples at its back, and d is its length.

© The Eurographics Association 2008.

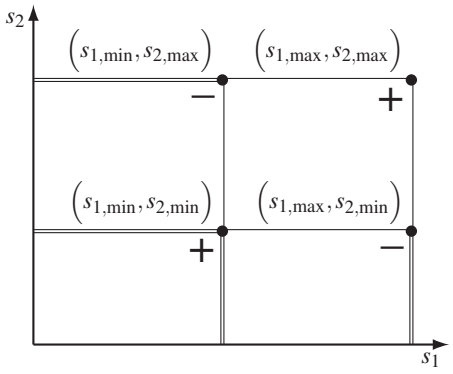


Figure 1: *Integration of a rectangular region specified by $s_{1,\min}$, $s_{1,\max}$, $s_{2,\min}$, and $s_{2,\max}$. The four values of the integral function at the specified points are added or subtracted according to each point's sign.*

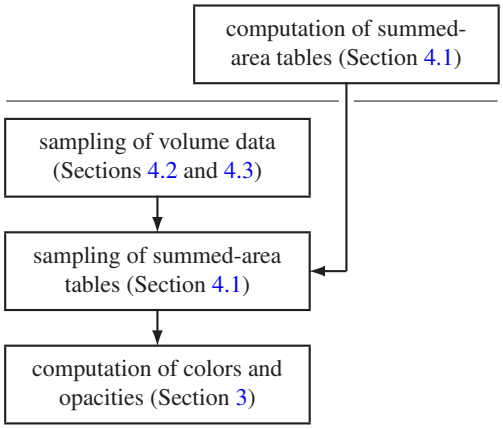


Figure 2: *Basic data flow of our volume rendering technique and the "back-to-front" organization of Sections 3 and 4.*

Integrating along a line from $(s_{1,\mathrm{f}}, s_{2,\mathrm{f}})$ to $(s_{1,\mathrm{b}}, s_{2,\mathrm{b}})$ can result in severe aliasing as explained in more detail in Section 4.2. Therefore, we will integrate over a rectangle defined by the minima $(s_{1,\min}, s_{2,\min})$ and maxima $(s_{1,\max}, s_{2,\max})$ with $s_{1,\min} \stackrel{\mathrm{def}}{=} \min(s_{1,\mathrm{f}}, s_{1,\mathrm{b}})$, $s_{2,\min} \stackrel{\mathrm{def}}{=} \min(s_{2,\mathrm{f}}, s_{2,\mathrm{b}})$, $s_{1,\max} \stackrel{\mathrm{def}}{=} \max(s_{1,\mathrm{f}}, s_{1,\mathrm{b}})$, and $s_{2,\max} \stackrel{\mathrm{def}}{=} \max(s_{2,\mathrm{f}}, s_{2,\mathrm{b}})$. The integral can be computed by adding and subtracting values of the integral function at the corners of the rectangle as illustrated in Figure 1. Thus, the opacities and colors are given by:

$$\begin{aligned}\alpha(s_{1,\min}, s_{2,\min}, s_{1,\max}, s_{2,\max}, d) = 1 - \exp\Bigg(&\tfrac{-d}{(s_{1,\max} - s_{1,\min})} \\ &\times \tfrac{1}{(s_{2,\max} - s_{2,\min})} \Big(T(s_{1,\max}, s_{2,\max}) - T(s_{1,\max}, s_{2,\min}) \\ &- T(s_{1,\min}, s_{2,\max}) + T(s_{1,\min}, s_{2,\min})\Big)\Bigg), \end{aligned} \tag{11}$$

$$\begin{aligned}\tilde{C}(s_{1,\min}, s_{2,\min}, s_{1,\max}, s_{2,\max}, d) = \\ &\alpha(s_{1,\min}, s_{2,\min}, s_{1,\max}, s_{2,\max}, d) \\ &\times \Big(K(s_{1,\max}, s_{2,\max}) - K(s_{1,\max}, s_{2,\min}) \\ &- K(s_{1,\min}, s_{2,\max}) + K(s_{1,\min}, s_{2,\min})\Big) \\ &/\Big(T(s_{1,\max}, s_{2,\max}) - T(s_{1,\max}, s_{2,\min}) \\ &- T(s_{1,\min}, s_{2,\max}) + T(s_{1,\min}, s_{2,\min})\Big). \end{aligned} \tag{12}$$

with the two-dimensional integral functions

$$T(s_1, s_2) \stackrel{\mathrm{def}}{=} \int_0^{s_1} \int_0^{s_2} \tau(s_1', s_2')\, \mathrm{d}s_2' \mathrm{d}s_1', \tag{13}$$

$$K(s_1, s_2) \stackrel{\mathrm{def}}{=} \int_0^{s_1} \int_0^{s_2} \tau(s_1', s_2')\, c(s_1', s_2')\, \mathrm{d}s_2' \mathrm{d}s_1'. \tag{14}$$

It is straightforward to apply the surface rendering discussed in the previous section to two-dimensional transfer functions using two surfaceness transfer functions for the two scalar values.

4. Sampling of Multi-Dimensional Volume Data and Multi-Dimensional Transfer Functions

While the previous Section 3 covered the mathematics of our approach, this section discusses the implementation of the proposed equations, in particular issues of sampling and texture-based representations of integral functions.

This section is organized "back-to-front," i.e., from the computation of colors and opacities to the sampling of volume data while the data flow of the algorithm is directed the other way around as illustrated in Figure 2. This organization was chosen since the reasons for the differences to standard volume rendering propagated from the computations with integral functions via the sampling of summed-area tables to the sampling of volume data. While this order might appear counter-intuitive, it should help to comprehend the motivation of each step.

4.1. Computation and Sampling of Summed-Area Tables

Tabulated two-dimensional integral functions are often referred to as summed-area tables in computer graphics. Crow [Cro84] proposed to employ texture-based representations of summed-area tables for the efficient filtering of texture images. Historically, however, mip-map representations were preferred for this purpose. Recently, Crow's suggestion was revived by Hensley et al. [HSC*05] since modern GPUs are flexible enough to efficiently support it.

© The Eurographics Association 2008.

In particular, modern GPUs provide support for 32-bit floating-point texture formats, which provide sufficient accuracy for summed-area tables of most images of dimensions $2^8 \times 2^8 \times 8$ bit. Since large areas of many two-dimensional transfer functions contain transparent black pixels, the precision is in fact sufficient to support considerably larger images, e.g., $2^{10} \times 2^{10} \times 8$ bit.

In our current implementation, the two-dimensional transfer functions are first sampled in an RGBA floating-point image of dimensions 1024×1024 with $\tau(s_1, s_2)$ in the alpha channel and $\tau(s_1, s_2)\ c(s_1, s_2)$ in the red, green, and blue channels. The summed-area tables (corresponding to $T(s_1, s_2)$ and $K(s_1, s_2)$) are computed in software by iteratively summing left and lower pixel neighbors to each pixel. This is extremely efficient as only one pass over the image is required on a CPU. A slight performance improvement could be achieved by performing the computation of the summed-area table on the GPU [HSC*05]; however, this would not avoid the bottleneck of transferring the texture data to the graphics memory. More accurate integration schemes also offer only limited improvements since the sampling error cannot be undone.

On many current GPUs, 32-bit floating-point textures do not support linear interpolation but only nearest-neighbor access. Thus, two samples of the texture at positions that are closer than the size of one texel can access the same texel and therefore the difference of their values is zero—independently of the texels' values. Our method is numerically robust in this case; thus, the resulting color will be transparent black. In pre-integrated volume rendering, however, it is often preferable to overestimate opacities in order not to skip fine details. Thus, we ensure that the texture coordinates corresponding to $s_{1,\max}$ and $s_{2,\max}$ are at least one texel greater than $s_{1,\min}$ and $s_{2,\min}$ by adding the missing distance if necessary.

4.2. Space-Covering Sampling Scheme

While pre-integrated volume rendering of one-dimensional transfer functions reliably renders arbitrarily thin isosurfaces, the situation is more complicated for two-dimensional transfer functions. A single peak $\delta(s - s_{\text{iso}})$ in a one-dimensional opacity transfer function corresponds to an isosurface; however, a peak in a two-dimensional opacity transfer function corresponds to the separable product $\delta(s_1 - s_{1,\text{iso}})\delta(s_2 - s_{2,\text{iso}})$ of two one-dimensional peaks and therefore to the intersection of two isosurfaces. In general, this intersection is a set of curves; thus, two-dimensional transfer functions can specify almost arbitrary curves in space. In fact, pre-integration even allows for arbitrarily thin curves.

Unfortunately, ray casting of thin curves results in severe aliasing artifacts as infinitesimal thin rays are unlikely to intersect these curves. In order to generalize pre-integration for multi-dimensional transfer functions, it is therefore necessary to substantially extend the ray casting approach.

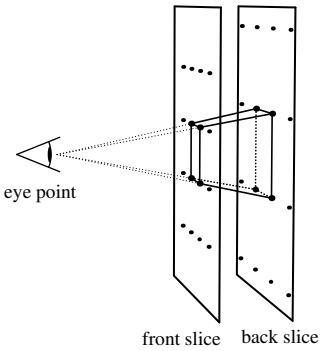


Figure 3: *Illustration of the eight sampling points of one pixel frustum between two slices.*

In this work we propose a space-covering sampling scheme to reliably render infinitesimal thin curves by means of volume rendering. Specifically, we suggest "pyramid casting" in analogy to "beam tracing" [HH84]. Figure 3 illustrates one "pixel frustum" of a traced pyramid, which is defined by eight sampling points. In our slicing approach, the frontmost four sampling points correspond to four adjacent pixels on the front slice while the backmost four sampling points correspond to pixels on a second slice. Thus, one pixel frustum can be defined for each pixel on the front slice and the set of all pixel frusta covers the space between the two slices. Therefore, even infinitesimal thin curves can be rendered provided that an intersection between a curve and a pixel frustum is reliably detected.

It should be noted that one pixel frustum corresponds to the ray segment between two sampling points in standard pre-integrated volume rendering. Moreover, the front and back slices are well-known from pre-integrated volume rendering [EKE01]. Also note that the total number of different sampling points is not increased in comparison to standard (pre-integrated) volume rendering since each sampling point is reused for eight pixel frusta (apart from pixel frusta at the boundary of the view frustum). This feature is exploited in Section 4.3 to avoid additional accesses to the volume data.

In the case of two-dimensional transfer functions, each of the eight sampling points $\mathbf{p}^{(i)}$ $(i = 1, \ldots, 8)$ of one pixel frustum specifies a pair $\left(s_1^{(i)}, s_2^{(i)}\right)$ of sample values. From these 16 values, the minima and maxima for s_1 and s_2 are computed, i.e., the four values $s_{1,\min} \stackrel{\text{def}}{=} \min_{i=1,\ldots,8} \left\{s_1^{(i)}\right\}$, $s_{2,\min} \stackrel{\text{def}}{=} \min_{i=1,\ldots,8} \left\{s_2^{(i)}\right\}$, $s_{1,\max} \stackrel{\text{def}}{=} \max_{i=1,\ldots,8} \left\{s_1^{(i)}\right\}$, and $s_{2,\max} \stackrel{\text{def}}{=} \max_{i=1,\ldots,8} \left\{s_2^{(i)}\right\}$. In combination with the distance between the back and front slice, these four values are sufficient to approximate the opacity and color of the pixel frustum by four table lookups in the precomputed summed-area tables as discussed in Section 3.3.

© The Eurographics Association 2008.

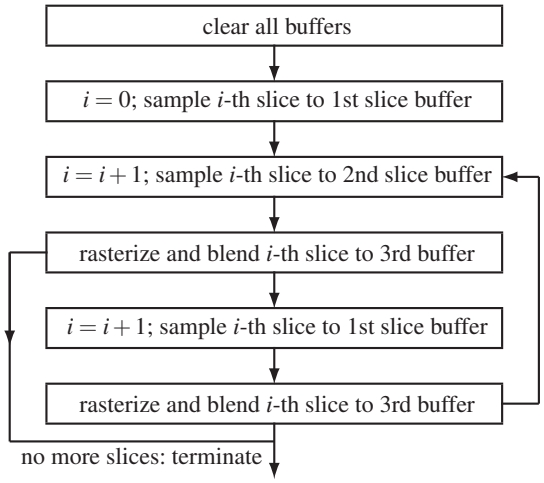


Figure 4: *Buffering of volume samples using three buffers.*

The eye space gradients $\nabla_e s_1$ and $\nabla_e s_2$ can also be approximated from the 16 values $s_1^{(i)}$ and $s_2^{(i)}$. To this end, we assume that the effect of perspective projection is negligible and compute the six components of $\nabla_e s_1$ and $\nabla_e s_2$ by averaging the appropriately scaled differences between sample values along parallel edges of the pixel frustum. The averaging results in some smoothing, which is welcome for the computation of gradients in most kinds of volume data.

For orthogonal model-view transformations, the gradient magnitudes $|\nabla s_1|$ and $|\nabla s_2|$ can be approximated by $|\nabla_e s_1|$ and $|\nabla_e s_2|$, otherwise the gradient vectors have to be transformed to object space first. Moreover, the eye space gradients can be employed directly for volume shading provided that light sources are also transformed into eye space.

4.3. Buffering of Volume Samples

The color and opacity for each pixel frustum is computed in a fragment program based on eight samples as discussed in the previous section. However, it is rather inefficient to sample all eight points directly from the volume data since each sample is used for up to eight pixel frusta. Sampling each point eight times is particular costly for compressed volume data that has to be decoded in the fragment program, as proposed, for example, by Fout et al. [FAM*05].

Therefore, we propose a simple buffering scheme involving three buffers: two slice buffers are used alternately to sample the volume data by rasterizing slices of the volume and performing any decoding of the volume data if necessary. No further processing of the samples is performed by this rasterization. All but the first slice are then also rasterized to a third buffer. In this step, four (already decoded) samples are read from each of the two slice buffers. Thus, the fragment program of this step accesses all eight samples of a pixel frustum to compute its opacity and color. The result is then accumulated in the third buffer by blending with its content.

This process is illustrated in Figure 4 and can be employed for back-to-front and front-to-back blending. Note, however, that one of the slice buffers will not provide valid data for all pixels since different slices cover different regions in screen space. For these pixels the result of the fragment program should be transparent black unless subpixel-accurate methods for boundary clipping are employed. Also note that the buffering of volume samples as well as the frequent changes of the render target results in a considerable overhead. Thus, the technique is most beneficial for costly accesses to the volume data, e.g., for complex decoding operations.

5. Results

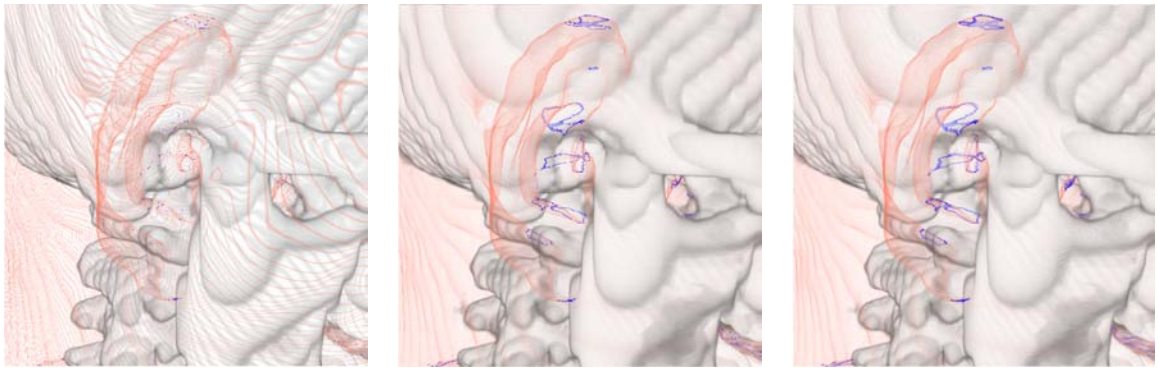
Figure 5 employs a two-dimensional transfer function to render isolines of constant gradient magnitude (in blue) on a semitransparent isosurface (in light red), which approximates the skin of an ear in a medical CT scan of $64 \times 64 \times 93$ voxels. The three figures compare single-sample volume rendering with a two-dimensional lookup table for the two-dimensional transfer function (Figure 5a), pre-integrated volume rendering for two-dimensional transfer functions as discussed in Section 3.3 (Figure 5b), and pre-integrated volume rendering with pixel-frustum sampling (Figure 5c) as discussed in Section 4.2. In all cases, the same image resolution and slice distance was employed.

While standard volume rendering fails to render most parts of the thin isosurface and misses the isolines apart from a few scattered pixels in Figure 5a, pre-integrated volume rendering can render the isosurface and most of the isolines in Figure 5b; however, closed isolines are only achieved with the pixel-frustum sampling employed in Figure 5c.

Figure 6 uses a synthetic second scalar field to apply a one-dimensional "texture" to an isosurface of a medical CT scan by embedding the one-dimensional texture into a two-dimensional transfer function. The resulting volume rendering is shown for the same three techniques as employed in Figure 5 and for two different, rather low image resolutions. Obviously, the pixel-frustum sampling technique strongly reduces aliasing artifacts in Figures 6c and 6f.

Our prototypical implementation is based on OpenGL and the VTK framework by Kitware, Inc. Table 1 compares the performance of the three mentioned volume rendering techniques on a Microsoft Vista-PC with an NVIDIA Quadro FX 5600 graphics board. The single-sample volume rendering is based on a standard slicing approach with a dependent texture lookup to apply the two-dimensional transfer function. On the other hand, the pre-integrated volume rendering approaches were implemented as discussed in Section 4.3, where the "front and back" technique accesses only one sample of each slice, while the "pixel frustum" technique reads

© The Eurographics Association 2008.



(a) (b) (c)

Figure 5: *Comparison of three volume rendering techniques supporting two-dimensional transfer functions: (a) standard post-classified volume rendering (1 sample per fragment), (b) pre-integrated volume rendering (2 samples per fragment), (c) pre-integrated volume rendering with pixel-frustum sampling (8 samples per fragment). The image resolution is* 512×512.

Table 1: *Rendering performance for 300 slices on a* 512×512 *view port in seconds per frame and frames per second.*

	pre-integrated	
single sample	front and back	pixel frustum
0.019 sec	0.066 sec	0.092 sec
53 fps	15 fps	11 fps

four samples of each slice, i.e., eight samples for each pixel frustum as explained in Section 4.2.

The buffering of volume samples is rather costly; however, it should be noted that the overhead is independent of the costs of sampling the volume data, which could be considerably higher for compressed volume data. Moreover, some techniques (e.g., [FAM*05]) also require a slice buffer for volume samples; thus, the additional costs for the proposed buffering would be significantly lower.

6. Conclusions & Future Work

This work demonstrates that efficient pre-integrated volume rendering with two-dimensional transfer functions is possible on modern GPUs. However, our current GPU-implementation suffers from the restriction to nearest-neighbor interpolation for 32 bit floating-point textures and the limited communication between fragment programs, which resulted in the considerable overhead of the proposed technique for buffering volume samples. While future GPU generations are likely to overcome these limitations, it might be possible to alleviate the problem of nearest-neighbor interpolation by employing 16 bit floating-point textures in combination with techniques to improve the accuracy of summed-area tables as suggested by Crow [Cro84] and Hensley et al. [HSC*05].

The most important implication of this work is the possibility to exploit the full potential of two-dimensional transfer functions in direct volume rendering. As we have demonstrated, this includes the anti-aliased rendering of three-dimensional curves and "textured" isosurfaces as well as many more applications, e.g., interactive segmentation and modelling of complex implicit shapes.

References

[Cro84] CROW F. C.: Summed-area tables for texture mapping. *ACM Computer Graphics (Proceedings of SIGGRAPH '84) 18*, 3 (1984), 207–212.

[EKE01] ENGEL K., KRAUS M., ERTL T.: High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings Graphics Hardware 2001* (2001), Mark W., Schilling A., (Eds.), ACM Press, pp. 9–16.

[FAM*05] FOUT N., AKIBA H., MA K.-L., LEFOHN A. E., KNISS J.: High-quality rendering of compressed volume data formats. In *Eurographics/VGTC Symposium on Visualization* (2005), pp. 77–84.

[HH84] HECKBERT P. S., HANRAHAN P.: Beam tracing polygonal objects. *ACM Computer Graphics (Proceedings of SIGGRAPH '84) 18*, 3 (1984), 119–127.

[HSC*05] HENSLEY J., SCHEUERMANN T., COOMBE G., SINGH M., LASTRA A.: Fast summed-area table generation and its applications. *Computer Graphics Forum (Proceedings Eurographics 2005) 24*, 3 (2005), 547–555.

[KD98] KINDLMANN G., DURKIN J. W.: Semi-automatic generation of transfer functions for direct vol-

© The Eurographics Association 2008.

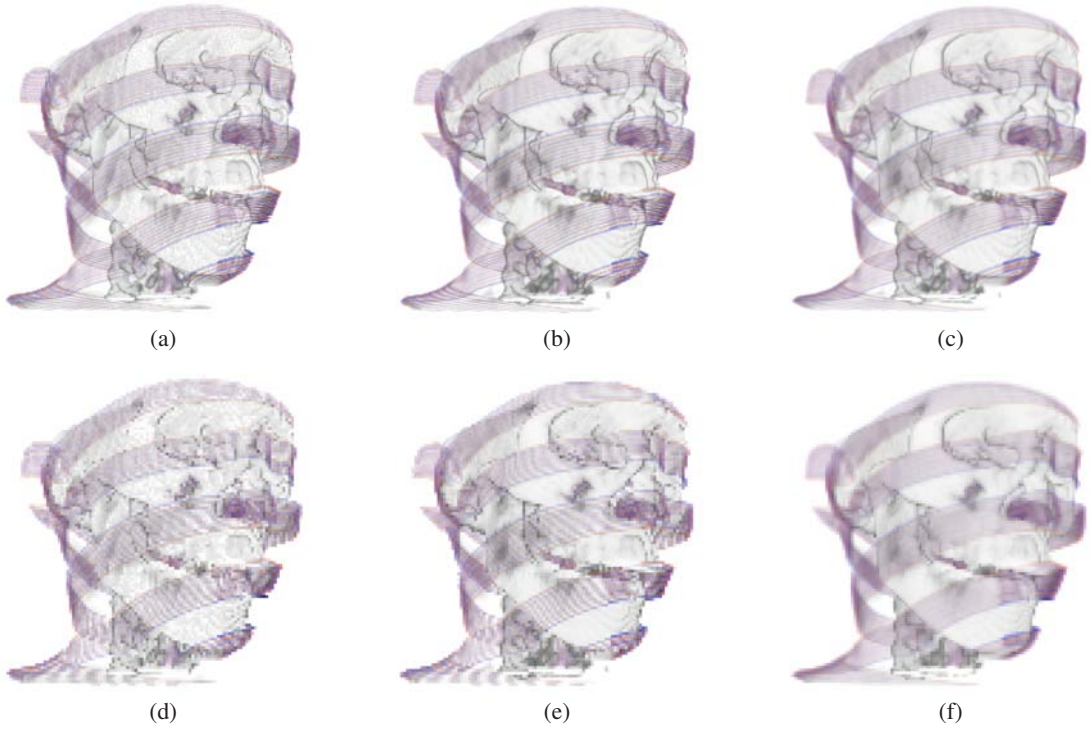


Figure 6: *Comparison of three volume rendering techniques supporting two-dimensional transfer functions at low image resolutions. (a) and (d): standard post-classified volume rendering (1 sample per fragment), (b) and (e): pre-integrated volume rendering (2 samples per fragment), (c) and (f): pre-integrated volume rendering with pixel-frustum sampling (8 samples per fragment). The image resolution is* 256×256 *in the first row (a)-(c), and* 128×128 *in the second row (d)-(f).*

ume rendering. In *Proceedings 1998 IEEE Symposium on Volume Visualization* (1998), pp. 79–86.

[KE04] KRAUS M., ERTL T.: Pre-integrated volume rendering. In *The Visualization Handbook* (2004), Hansen C. D., Johnson C. R., (Eds.), Academic Press, pp. 211–228.

[KHS05] KYE H., HONG H., SHIN Y. G.: Efficient interactive pre-integrated volume rendering. In *Computational Science – ICCS 2005* (2005), Springer Berlin / Heidelberg, pp. 834–837.

[KKH02] KNISS J., KINDLMANN G., HANSEN C.: Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics 8*, 3 (2002), 270–285.

[KPI*03] KNISS J., PREMOŽE S., IKITIS M., LEFOHN A., HANSEN C., PRAUN E.: Gaussian transfer functions for multi-field volume visualizations. *Proceedings Visualization 2003* (2003), 497–504.

[Kra05] KRAUS M.: Scale-invariant volume rendering. In *Proceedings IEEE Visualization 2005* (2005), pp. 295–302.

[Lev88] LEVOY M.: Display of surfaces from volume data. *IEEE Computer Graphics and Applications 8*, 3 (1988), 29–37.

[LWM04] LUM E. B., WILSON B., MA K.-L.: High-quality lighting and efficient pre-integration for volume rendering. In *Proceedings Joint Eurographics-IEEE TVCG Symposium on Visualization 2004 (VisSym '04)* (2004), pp. 25–34.

[MHC90] MAX N., HANRAHAN P., CRAWFIS R.: Area and volume coherence for efficient visualization of 3d scalar functions. *ACM Computer Graphics (Proceedings San Diego Workshop on Volume Visualization 1990) 24*, 5 (1990), 27–33.

[RKE00] RÖTTGER S., KRAUS M., ERTL T.: Hardware-accelerated volume and isosurface rendering based on cell-projection. In *Proceedings IEEE Visualization 2000* (2000), pp. 109–116.

[YMC06] YOUNESY H., MÖLLER T., CARR H.: Improving the quality of multi-resolution volume rendering. In *Eurographics/IEEE-VGTC Symposium on Visualization* (2006), pp. 251–258.

© The Eurographics Association 2008.