Volume Graphics (2003)
I. Fujishiro, K. Mueller, A. Kaufman (Editors)

Integrating Pre-Integration Into The Shear-Warp Algorithm

J.P. Schulze,[1] M. Kraus,[2] U. Lang[1] and T. Ertl[2]

[1] High Performance Computing Center Stuttgart (HLRS)
{schulze,lang}@hlrs.de

[2] Visualization and Interactive Systems Group, University of Stuttgart, Germany
{kraus,ertl}@informatik.uni-stuttgart.de

Abstract

The shear-warp volume rendering algorithm is one of the fastest algorithms for volume rendering, but it achieves this rendering speed only by sacrificing interpolation between the slices of the volume data. Unfortunately, this restriction to bilinear interpolation within the slices severely compromises the resulting image quality. This paper presents the implementation of pre-integrated volume rendering in the shear-warp algorithm for parallel projection to overcome this drawback. A pre-integrated lookup table is used during compositing to perform a substantially improved interpolation between the voxels in two adjacent slices.

We discuss the design and implementation of our extension of the shear-warp algorithm in detail. We also clarify the concept of opacity and color correction, and derive the required sampling rate of volume rendering with post-classification. Furthermore, the modified algorithm is compared to the traditional shear-warp rendering approach in terms of rendering speed and image quality.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation – Display Algorithms; I.4.10 [Image Processing and Computer Vision]: Image Representation – Volumetric.

1. Introduction

Although the shear-warp volume rendering algorithm achieves a high rendering performance, it is still not widely used for interactive volume rendering. The most important competitor is probably texture-based volume rendering; see for example [3]. This approach is very efficient as long as the graphics hardware provides the required functionality. But even then, this approach has several disadvantages: the rendering speed is limited by the pixel fill-rate, shading imposes a serious performance hit, and for interactive rendering the entire volume dataset has to fit into texture memory. The shear-warp algorithm, on the other hand, is a software-based volume rendering algorithm, which traverses the volume data in object order. Therefore, it is extremely flexible, allows run-length encoding of the volume data, and supports efficient cache usage.

Pre-integrated volume rendering provides an efficient way to interpolate in-between slices of the volume data with some loss in rendering performance. Pre-integration is based on the pre-computation of a lookup table, which supplies RGBA values for every possible pair of scalar values. With the help of this table, pre-integrated volume rendering is able to interpolate linearly between the slices, instead of assuming a constant scalar value between the slices (as in the original shear-warp algorithm). Thus, pre-integration achieves significantly improved results, in particular for nonlinear transfer functions. Pre-integrated volume rendering is, therefore, a perfect complement to the shear-warp algorithm.

Before we present our new algorithm, we reference prior work and discuss the underlying theoretical background in Section 2. In particular, we address the employed optical model, opacity and color correction, the required volume sampling rate for standard volume rendering, and the original shear-warp algorithm. In Section 3, we discuss the details of our algorithm and its implementation. Performance results and comparisons of image quality of several variants of our algorithm are presented in Section 4.

2. Theoretical Background

In this section, we address the mathematical foundations of our new pre-integrated volume rendering algorithm.

2.1. Volume Rendering Integral

The basic task of any volume renderer is an approximate evaluation of the *volume rendering integral* for each pixel, i.e., the integration of attenuated colors along each viewing ray. Although the numerical evaluation of this integral is well-known, it is briefly recapitulated here in order to introduce our nomenclature and to remind the reader of the employed approximations.

We specify colors and extinction coefficients for each scalar value s of the volume data by transfer functions $c(s)$ and $\tau(s)$. The color emitted from one point of the volume is determined by $\tau(s)c(s)$; thus, the volume rendering integral for the intensity I along a viewing ray parametrized by x from 0 to D is given by

$$I = \int_0^D \tau(s(x))c(s(x)) \exp\left(-\int_0^x \tau(s(x'))\mathrm{d}x'\right)\mathrm{d}x.$$

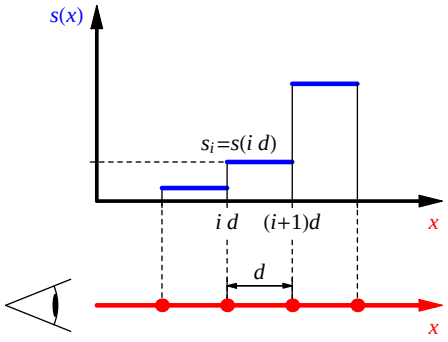


Figure 1: *Sampling of $s(x)$ along a viewing ray.*

The volume rendering integral can be approximated by a Riemann sum of n equal ray segments of length $d := D/n$. This evaluation assumes that $s(x)$ is approximately constant for each ray segment (see also Figure 1):

$$\begin{aligned} I &\approx \sum_{i=0}^{n-1} \tau(s(id))c(s(id))d \exp\left(-\sum_{j=0}^{i-1}\tau(s(jd))d\right) \\ &\approx \sum_{i=0}^{n-1} \tau(s(id))c(s(id))d \prod_{j=0}^{i-1}\exp\left(-\tau(s(jd))d\right) \\ &\approx \sum_{i=0}^{n-1} c_i \prod_{j=0}^{i-1}(1-\alpha_j) \end{aligned}$$

with the opacity α_i of the i-th ray segment, which is defined by

$$\begin{aligned} \alpha_i &:= 1-\exp\left(-\int_{id}^{(i+1)d}\tau(s(x))\,\mathrm{d}x\right) \\ &\approx 1-\exp\left(-\tau(s(id))\,d\right) \\ &\approx \tau(s(id))\,d. \end{aligned}$$

The (premultiplied) color c_i emitted in the i-th ray segment is defined by

$$c_i := \int_{id}^{(i+1)d} \tau(s(x))c(s(x)) \exp\left(-\int_{id}^{x}\tau(s(x'))\mathrm{d}x'\right)\mathrm{d}x.$$

Neglecting the self-attenuation within the ray segment, c_i may be approximated by

$$\begin{aligned} c_i &\approx \int_{id}^{(i+1)d} \tau(s(x))\,c(s(x))\,\mathrm{d}x \\ &\approx \tau(s(id))\,c(s(id))\,d \\ &\approx \alpha_i c(s(id)). \end{aligned}$$

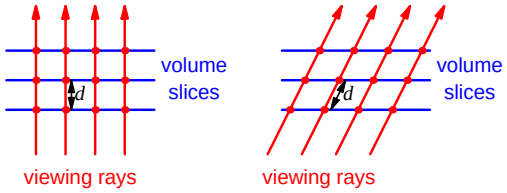
Therefore, a front-to-back compositing algorithm (which is usually employed in the shear-warp algorithm) implements the equations

$$\begin{aligned} \hat{\alpha}_i &= 1-(1-\hat{\alpha}_{i-1})(1-\alpha_i) \\ &= \hat{\alpha}_{i-1}+(1-\hat{\alpha}_{i-1})\alpha_i, \\ \hat{c}_i &= \hat{c}_{i-1}+(1-\hat{\alpha}_{i-1})c_i \end{aligned}$$

for the accumulated opacity $\hat{\alpha}_i$ and color $\hat{c}_i$ of the i-th ray segment.

2.2. Opacity and Color Correction

Some volume rendering algorithms, for example the non-perspective shear-warp algorithm or 2D texture-based volume rendering (see [3]), evaluate the volume rendering integral with equally spaced samples, i.e., a constant distance d between samples. Thus, the opacities α_i and colors c_i may be precomputed for a constant d.



(a) (b)

Figure 2: *Different distances between samples depending on the viewing direction.*

However, the distance d still depends on the viewing direction as illustrated in Figure 2. Thus, it is necessary to correct the precomputed opacities and colors. While the opacity correction is well-known (see for example [8]), the correction of colors appears to be less common. Therefore, both corrections are briefly derived here.

Assuming that an opacity α_i has been computed for a constant scalar s and the sample distance d, the corrected opacity

α_i' for a different sample distance d' may be computed by

$$\begin{aligned}\alpha_i' &= 1-\exp\left(-\tau(s)d'\right)\\ &= 1-\exp\left(-\tau(s)d\frac{d'}{d}\right)\\ &= 1-\exp\left(-\tau(s)d\right)^{d'/d}\\ &= 1-\left(1-\alpha_i\right)^{d'/d}.\end{aligned}$$

As suggested by Lacroute in [8], this correction can be efficiently implemented by a lookup table for α_i' as a function of α.

The premultiplied color c_i has to be corrected correspondingly since it is proportional to α_i:

$$c_i' = c_i\frac{\alpha_i'}{\alpha_i}.$$

A more rigorous derivation of this result can be given by evaluating c_i for a constant scalar s:

$$\begin{aligned}c_i &= \int_{id}^{(i+1)d}\tau(s)c(s)\exp\left(-\int_{id}^{x}\tau(s)\mathrm{d}x'\right)\mathrm{d}x\\ &= \int_{id}^{(i+1)d}\tau(s)c(s)\exp\left(-\tau(s)(x-id)\right)\mathrm{d}x\\ &= \left[-c(s)\exp\left(-\tau(s)(x-id)\right)\right]_{id}^{(i+1)d}\\ &= c(s)\left(1-\exp\left(-\tau(s)d\right)\right).\end{aligned}$$

With α_i and α_i' from above, the corrected color c_i' for d' is

$$c_i' = c(s)\left(1-\exp\left(-\tau(s)d'\right)\right) = c(s)\alpha_i' = c(s)\alpha_i\frac{\alpha_i'}{\alpha_i} = c_i\frac{\alpha_i'}{\alpha_i}.$$

For a physical interpretation of this color correction the cases of very low and very high opacity are of particular interest: For a very low opacity the self-attenuation may be neglected; thus, the color emission is proportional to the length of the ray segment. On the other hand, for a very high opacity the color cannot depend on the length of the ray segment since the light from its far end is blocked and, therefore, cannot influence the integrated color. Both cases are correctly described by the color correction given above.

Note that this color correction is perfectly consistent with the special case of $d'/d = 1/2$ discussed by Sweeney and Mueller in [20] since the correction factor (called λ in [20]) is given by

$$\frac{\alpha_i'}{\alpha_i} = \frac{1-(1-\alpha_i)^{1/2}}{\alpha_i} = \frac{1-\sqrt{1-\alpha_i}}{1-(1-\alpha_i)} = \frac{1}{1+\sqrt{1-\alpha_i}}.$$

2.3. Volume Sampling Rate

The discrete approximation of the volume rendering integral will converge to the correct result only for high sampling rates $1/d$. Unfortunately, nonlinear transfer functions may considerably increase the sampling rate required for a correct evaluation of the volume rendering integral as this sampling rate depends on the product of the Nyquist frequencies of the scalar field and the transfer functions as mentioned (but not proved) by Engel et al. in [3].

The actual sampling rate required for an accurate evaluation may be estimated by the sampling rate required for an accurate reconstruction of the functions $\tau(s(x))$ and $c(s(x))$. This sampling rate may be obtained by comparison with a frequency-modulated signal $s_{\mathrm{fm}}(t)$ (see Section 6.4 in [19]):

$$s_{\mathrm{fm}}(t) := A\cos\left(2\pi f_c t + (\Delta f/f_{\mathrm{m}})\sin\left(2\pi f_{\mathrm{m}} t\right)\right)$$

with the amplitude A, the carrier frequency f_c, the maximum deviation Δf from f_{c}, and the modulation frequency f_{m} (which is the maximum frequency of the modulation signal if it is not a single frequency tone). With the help of the identity

$$\cos(a+x\sin(b)) = \sum_{k=-\infty}^{\infty}\cos(a+kb)\,J_k(x)$$

(with the Bessel function J_k of the first kind of order k), the modulated signal may be written as

$$s_{\mathrm{fm}}(t) = A\sum_{k=-\infty}^{\infty}J_k\left(\Delta f/f_{\mathrm{m}}\right)\cos\left(2\pi f_c t + 2\pi f_{\mathrm{m}} k t\right).$$

The spectrum of $s_{\mathrm{fm}}(t)$ may be obtained directly from this representation: Apart from the carrier frequency f_{c}, there is an infinite number of sidebands at frequencies $f_{\mathrm{c}} \pm f_{\mathrm{m}}k$ with $k \in \mathbb{N}$. Thus, the modulated signal is not bandwidth-limited and there is no maximum frequency. However, according to an approximation by Carson (known as "Carson's rule"), the actually required bandwidth (for more than 98 % of the signal power) is $2(\Delta f + f_{\mathrm{m}})$, i.e., contributions of sidebands outside the interval $[f_{\mathrm{c}} - (\Delta f + f_{\mathrm{m}}), f_{\mathrm{c}} + (\Delta f + f_{\mathrm{m}})]$ are negligible.

In order to apply this result to the problem of determining an appropriate sampling frequency along a viewing ray, some additional symbols have to be introduced. Let $U(S)$ denote a continuous transfer function for scalar values $S \in [0,1]$ with Nyquist frequency f_{U}. (A discontinuous transfer function could be approximated with extremely high frequencies.) In order to define values $U(S)$ for $S \notin [0,1]$, let $U(S)$ be a symmetric function with period 2, i.e., $U(S) = U(-S)$ and $U(S) = U(S+2k)$ for $k \in \mathbb{N}$. Furthermore, let $S(t)$ denote a scalar field with Nyquist frequency f_{S}. Thus, the problem is to determine an appropriate sampling frequency for $U(S(t))$. This problem can be simplified with the help of a Fourier cosine series:

$$U(S(t)) = \sum_{k=0}^{\infty}a_k\cos(\pi\, k\, S(t)).$$

As the appropriate sampling rate for this sum corresponds to the maximum of the sampling rates for the individual summands, it is possible to restrict the following considerations to the summand with the maximum k with $a_k \neq 0$. This $k_{\max}$ corresponds to a maximum frequency $k_{\max}/2$, which is given

by half the Nyquist frequency f_U, i.e.

$$k_{max}/2 = f_U/2.$$

Thus, $U(S(t))$ can be specialized to the form

$$A\cos(\pi\, k_{max}\, S(t)) = A\cos\left((2\pi f_U/2)\, S(t)\right),$$

with $A = a_{k_{max}}$. This function is already close to a frequency-modulated signal, where $S(t)$ corresponds to the modulation signal. As mentioned above, it is common to replace an arbitrary modulation signal by a single-frequency tone of the maximum frequency for the purpose of estimating an appropriate sampling rate. Thus, $S(t)$ is replaced by $\sin((2\pi f_S/2)\, t)$. The new form of $U(S(t))$ is:

$$\tilde{U}(\tilde{S}(t)) = A\cos\left((2\pi f_U/2)\sin((2\pi f_S/2)\, t)\right).$$

In order to apply Carson's rule, $\tilde{U}(\tilde{S}(t))$ has to be matched to $s_{fm}(t)$, which is defined by

$$s_{fm}(t) := A\cos\left(2\pi f_c t + (\Delta f/f_m)\sin(2\pi f_m t)\right).$$

For this purpose, f_m should be identified with half the Nyquist frequency f_S of the scalar field, and f_c has to be 0 as there is no "carrier frequency" for the transfer function. Thus, $\Delta f/f_m$ should be identified with $2\pi f_U/2$.

According to Carson's rule, the required frequencies for this signal ($f_c = 0$) are in the interval $[0, \Delta f + f_m]$ corresponding to $[0, 2\pi f_U f_S/4 + f_S/2]$. For $f_U f_S \gg f_S$ this interval is given by $[0, \pi f_U f_S/2]$, i.e., the required sampling frequency is $\pi f_U f_S$. While Carson's rule is a well-known approximation in signal theory, it has—to our knowledge—not been applied to volume rendering before.

Because of this result, it is by no means sufficient to sample the volume rendering integral with the Nyquist frequency f_S of the scalar field if non-linear transfer functions are employed. Artifacts resulting from this kind of undersampling are frequently observed unless they are avoided by very smooth transfer functions, i.e., transfer functions with a small Nyquist frequency f_U.

2.4. Pre-Integrated Volume Rendering

Pre-integrated volume rendering overcomes the necessity for extremely high sampling rates by splitting the numerical evaluation of the volume rendering integral into two integrations: one for the continuous scalar field $s(x)$ and one for the transfer functions $\tau(s)$ and $c(s)$; thus, the problematic product of Nyquist frequencies is avoided.

Pre-integration is similar to a method published by Max et al. in [10], which was reinvented and generalized for hardware-accelerated tetrahedra projection by Röttger et al. in [16]. However, the name "pre-integrated volume rendering" was first used by Engel et al. in [3] within the context of texture-based volume rendering. The basic concept of pre-integration may be applied to many other volume rendering algorithms; for example, Knittel demonstrated pre-integrated ray casting in [7]. More applications and improvements of pre-integrated volume rendering may be found in [2, 4, 5, 11, 13, 15, 21].

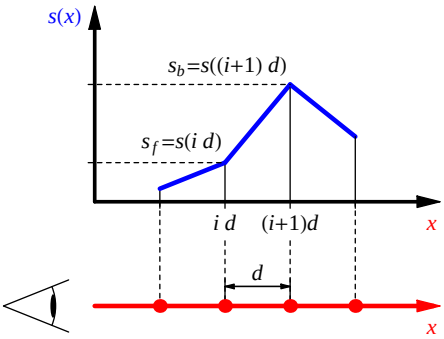


Figure 3: *Piecewise linear interpolation of samples of $s(x)$ for pre-integrated volume rendering.*

For the purpose of pre-integrated volume rendering, the scalar function $s(x)$ is approximated by a piecewise linear scalar function as illustrated in Figure 3. The volume rendering integral for this piecewise linear scalar function is efficiently computed by one table lookup for each ray segment. The three arguments of this table lookup for the i-th ray segment from id to $(i+1)d$ are the scalar value at the start (front) of the segment $s_f := s(i\,d)$, the scalar value the end (back) of the segment $s_b := s((i+1)d))$, and the length of the segment d (see Figure 3). If d is constant for all segments and all viewing rays, the table lookup does, of course, not depend on it and a two-dimensional table is sufficient.

More precisely spoken, the opacity α_i of the i-th segment is approximated by

$$\begin{aligned}\alpha_i &= 1 - \exp\left(-\int_{id}^{(i+1)d} \tau(s(x))\mathrm{d}x\right)\\ &\approx 1 - \exp\left(-\int_0^1 \tau\Big((1-\omega)s_f + \omega s_b\Big)d\mathrm{d}\omega\right).\end{aligned}$$

Thus, α_i is a function of s_f, s_b, and d, if the latter is not constant.

The (premultiplied) colors c_i are approximated correspondingly:

$$\begin{aligned}c_i \approx &\int_0^1 \tau\Big((1-\omega)s_f + \omega s_b\Big)\, c\Big((1-\omega)s_f + \omega s_b\Big)\\ &\times \exp\left(-\int_0^\omega \tau\Big((1-\omega')s_f + \omega' s_b\Big)d\mathrm{d}\omega'\right)d\mathrm{d}\omega.\end{aligned}$$

Analogously to α_i, c_i is a function of s_f, s_b, and d.

Apart from these approximations for α_i and c_i there are no further modifications of the evaluation of the volume rendering integral; i.e., the compositing algorithm from above may be employed for pre-integrated volume rendering, too. In particular, the opacity correction of Section 2.2 also applies to pre-integrated rendering and the color correction of Section 2.2 is an appropriate approximation in this case.

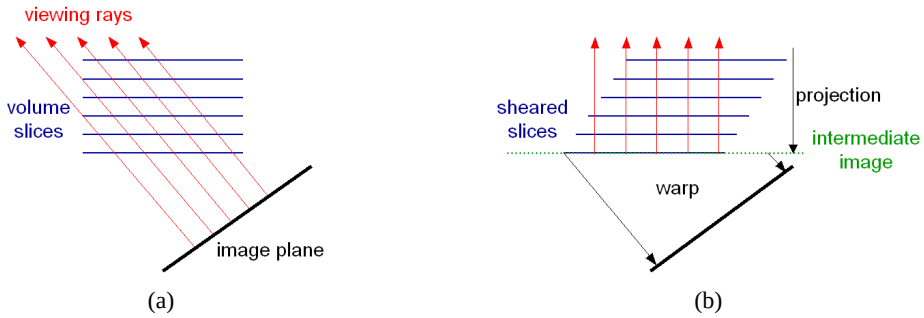


Figure 4: *(a) Standard and (b) factorized viewing transformation.*

Note, however, that the pre-integration always supplies premultiplied colors c_i; thus, a subsequent multiplication with α_i has to be avoided.

The computation of the pre-integrated lookup tables for α_i and c_i is rather expensive as two integrals have to be evaluated numerically for a sufficient number of combinations of values for s_f, s_b, and d. Moreover, these tables have to be recalculated whenever the transfer functions $\tau(s)$ or $c(s)$ are modified; thus, an evaluation at interactive rates is highly desirable. Fortunately, there are several ways of accelerating the computation of these lookup tables, which are discussed in detail in [3]. They allow us to perform the calculation of the two-dimensional lookup tables for constant d at interactive rates.

One remarkable feature of pre-integrated volume rendering is the possibility to render closed isosurfaces even with very low sampling rates by specifying sharp peaks in the transfer function $\tau(s)$; see [16] and [3] for details.

In summary, pre-integrated volume rendering allows us to evaluate the volume rendering integral without the need to increase the sampling rate for any nonlinear transfer function. Therefore, it has the potential to improve the accuracy (by less undersampling) and the performance (by fewer sampling operations) of a volume renderer at the same time.

For an accurate evaluation of the volume rendering integral, the actual sampling rate should be well above the Nyquist frequency of the scalar volume data since pre-integration uses a linear interpolation between samples instead of an ideal reconstruction filter. In practice, however, sampling rates close to this Nyquist frequency appear to result in a sufficient image quality for most data sets.

It should be noted that pre-integrated volume rendering will usually generate slightly different colors and opacities compared to many other volume rendering algorithms even for very smooth transfer functions; for example, because the approximation $1-\exp(\tau(s)d) \approx \tau(s)d$ is never needed for pre-integrated volume rendering. These approximations should not matter for high sampling rates $1/d$; however, many volume rendering implementations perform the compositing of colors with fixed-point arithmetic, resulting in considerable color alterations for high sampling rates.

2.5. Shear-Warp

Since the original presentation of the shear-warp algorithm by Lacroute [8, 9], there have been a number of publications on this algorithm. Work was done in fields like stereo rendering [6], fast slab rendering [22], and fast rotation [1]. The algorithm was implemented in volume rendering hardware [14], and it was compared to other volume rendering algorithms in [12]. In [18], the algorithm was extended to perspective projection, and then parallelized in [17]. In [20], several extensions for improved image quality were described. However, the use of pre-integration within the shear-warp algorithm has not previously been published.

The general viewing transformation V consists of a view matrix M and a projection P, such that $V = P \times M$. The shear-warp algorithm is based on the idea of a factorization of the view matrix M into a shear component S and a warp component W with $M = W \times S$. Lacroute's idea [9] is to do the warp after the projection, such that it becomes a two-dimensional operation, which is fast to compute. Thus, the final shear-warp viewing transformation becomes:

$$V = W \times P \times S.$$

This factorization allows us to perform the compositing in object space. Therefore, the memory cache can work efficiently because of frequent cache hits. The final warp is an affine two-dimensional transformation, which can be done on the processor efficiently, or even faster by using 2D texturing hardware. Figure 4 depicts the factorization into shear and warp. In Figure 4a, the projection of the volume to the image plane is done traditionally, while in Figure 4b the volume slices are first sheared, then projected onto an intermediate image, and finally warped to the actual image plane.

In Lacroute's implementation, the compositing of the volume slices to the intermediate image is done slice by slice and from front to back, with bilinear interpolation within

each slice and pre-interpolated classification. The intermediate image is aligned with the volume slices, i.e., the front voxels occupy one pixel per voxel. Opacity correction and early ray termination are performed for the compositing. Run-length encoding of the volume data is employed to save memory and to speed up the compositing process. In the warp, bilinear interpolation is used for the projection to the final image.

In summary, the most important features of the shear-warp algorithm are: efficient compositing; flexibility to incorporate shading, shadows, and arbitrary compositing models; and independence of the compositing from the output image size. A disadvantage is that the warp introduces additional blurring because of the necessary resampling.

3. Shear-Warp with Pre-Integration

This section discusses our extension of the shear-warp algorithm with pre-integration and the implementation issues that we encountered. The following topics are addressed: slab rendering, buffer slices to avoid redundant computations, the pre-integration table lookup, and rasterization differences between our new and the standard shear-warp algorithm.

3.1. Slab Rendering

As described in Section 2.4, pre-integrated volume rendering computes the color of ray segments instead of point samples on viewing rays. Thus, our variant of the shear-warp algorithm has to render slabs between adjacent slices instead of individual slices; see Figure 5. More specifically spoken, we still traverse the slices of the volume data in front-to-back order but render the slab in front of a slice instead of the slice itself.

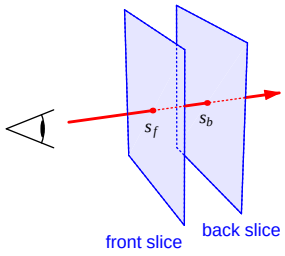


Figure 5: *A viewing ray through a slab between two slices. The scalar values of the volume data on the front slice and the back slice are denoted by s_f and s_b, respectively.*

As each slab between two slices is rendered with the help of the scalar values s_f and s_b on these slices, the bilinearly interpolated scalar values are used twice, once for each adjacent slab. Instead of computing the same bilinear interpolation for each slab, we employ a buffer slice, which is discussed next.

3.2. Buffer Slice

The buffer slice stores interpolated scalar values of the back slice as floating point numbers, such that these values can be reused for the front slice of the next slab. For the implementation of the buffer slice, we experimented with two slightly different approaches. The first option is to store two buffer slices in memory, each with the size of the volume slices that are rendered to the intermediate image (slice-sized buffer slices; see Figure 6a). Two buffer slices are required in order not to overwrite buffered values before they are needed for the pre-integration table lookup. Thus, two blocks of memory have to be allocated, and the size of the two buffer slices has to be adapted whenever the size of the displayed slices changes. Depending on the volume size, this may happen whenever the principal viewing axis changes. In the case of cubic volumes, the size of the buffer slices is always the same because the slices that are rendered to the intermediate image are of the same size for each principal axis. In order not to allocate and de-allocate memory whenever the principal axis changes, we decided to allocate memory for the buffer slices only once and use the size of the largest slices.

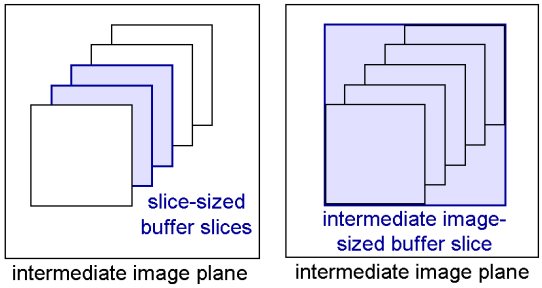


Figure 6: *(a) Slice-sized and (b) intermediate image-sized buffer slices.*

The second approach is to create a single buffer slice, which has the same size as the intermediate image (intermediate image-sized buffer slice; see Figure 6b). In this case only one slice is needed because a scalar value is always buffered right after the scalar value buffered previously at the same position has been read. This approach requires to change the size of the buffer slice whenever the size of the intermediate image changes, i.e., for every change of the viewpoint. This size is easily computed because our implementation of the shear-warp algorithm is based on the same idea for the allocation of the memory for the intermediate image. In order to prevent frequent memory allocation, we can follow the same approach as for the slice-sized buffer slices by allocating memory for the largest intermediate image size.

With the approaches described above, there is no difference in the frequency of memory allocation, but there is a difference in the size of the allocated memory. Let v_x and v_y be the width and height of the slices in vox-

els, respectively. Then, in the case of the parallel projection shear-warp algorithm, the intermediate image consists of $(2 \times v_x) \times (2 \times v_y)$ pixels in the worst case, i.e., when the viewer looks along the diagonal of the object. The intermediate image-sized buffer slice requires almost as many floating point (float) elements as there are intermediate image pixels, i.e., $(2 \times v_x) \times (2 \times v_y) = 4 \times v_x \times v_y$ floats. (Strictly speaking, it requires one row and one column less.)

The two slice-sized buffer slices require $2 \times v_x \times v_y$ floats (again, the correct value is one row and one column less). Thus, the slice-sized buffer slices require almost exactly half the amount of memory compared to the intermediate image-sized slice buffer.

We did not implement a pre-integrated perspective shear-warp algorithm, but similar considerations as for parallel projection apply. The intermediate image-sized buffer slice can be used in an analogous way. However, the slice-sized buffer slices vary with the size of the volume slices that are composited to the intermediate image. As we allocate memory only for the front slice, and change the size of the buffer slice by changing its size variables, there is no memory allocation penalty to the slice-sized buffer slice approach.

3.3. Pre-Integration Table Lookup

The pre-integration table is recomputed whenever the transfer function changes. We compute only a two-dimensional pre-integration table for a constant distance d, because the computation of a three-dimensional table is significantly more expensive and the image quality is hardly improved. Compared to using a two-dimensional table and opacity correction, the images generated with a three-dimensional lookup were slightly brighter in our experiments.

The bilinear interpolation that is performed to determine the scalar values s_f and s_b for the lookup in the pre-integration table generates floating point numbers. Thus, the lookup in the pre-integration table should bilinearly interpolate the tabulated colors and opacities. This is rather expensive, since it adds another bilinear interpolation for the composition of each voxel. Therefore, we experimented with nearest-neighbor interpolation for the pre-integration lookup, and with lookup tables larger than 256 entries, which we usually use. We found that for typical transfer functions, no difference is visible in the resulting images. Thus, it is sufficient to use nearest-neighbor interpolation for the lookup and gain a few percent of rendering speed (see Section 4).

3.4. Rasterization

A fundamental difference in rendering between the traditional approach with bilinear interpolation compared to pre-integration is the number of slices that are actually rendered: traditionally, one slice is rendered for each slice that is present in the volume dataset in the principal viewing direction. Since the pre-integration approach requires two volume slices and renders the slab in-between them, one slice less has to be rasterized with this approach. However, for typical volume sizes starting with about 100 slices this effect can be neglected.

4. Results

After we had integrated all the discussed improvements in our implementation of the shear-warp algorithm for parallel projection, we performed speed tests of the algorithm with different combinations of extensions and compared the resulting image quality.

4.1. Rendering Performance

The rendering performance tests were performed on a PC with a 1.7 GHz Pentium 4 processor, 256 MB RAM, and an ATI Radeon 7500 graphics card. The output image size was 512^2. We used the following datasets for the performance tests: the General Electric CT Engine, the UNC's MR Brain, and Stefan Röttger's Bonsai tree. The opacity transfer function was set to a linear ramp from zero to full opacity, which extended over the entire data range. In the case of the pre-integrated shear-warp, the transfer function does not affect rendering performance in any other way than for the traditional shear-warp, e.g., via early ray termination. We applied an automatic performance measurement procedure, which rotated the volume by 180 degrees in steps of 2 degrees about its vertical axis. The average rendering times per displayed frame are listed in Table 1.

In the table, the first three columns specify the dataset, its size, and the percentage of transparent voxels it contains. The fourth column shows the performance of the standard shear-warp algorithm without pre-integration and without opacity correction. The remaining columns list the times that are achieved with different combinations of extensions. Three types of extensions are distinguished: lookup in the pre-integration table, opacity correction (including color correction), and slice buffers. Opacity correction was implemented as described by Lacroute in [8]. The first four columns of the pre-integrated rendering tests are results from rendering with nearest-neighbor lookup in the pre-integration table, for the last four columns this lookup is improved by bilinear interpolation between the table values. The abbreviations used for the further classification of the table are as follows: OC: opacity correction enabled, NC: no opacity correction, SB: two slice-sized buffer slices, and IB: one intermediate image-sized buffer slice. In all the performance tests, the intermediate image was warped by the 2D texturing hardware, as mentioned in Section 2.5.

The times indicate that the pre-integrated shear-warp algorithm achieves a performance which is between 34% and 88% of the speed of the standard shear-warp, depending on

				Pre-Integration							
				Nearest Neighbor Lookup				Bilinear Lookup			
				NC		OC		NC		OC	
Dataset	Size [voxels]	Transparent	Standard	SB	IB	SB	IB	SB	IB	SB	IB
Engine	$128^2 \times 55$	28.0 %	0.26	0.30	0.30	0.34	0.34	0.36	0.36	0.40	0.40
Brainsmall	$128^2 \times 84$	13.3 %	0.43	0.49	0.49	0.56	0.57	0.58	0.59	0.66	0.66
Bonsai	128^3	79.5 %	0.23	0.56	0.57	0.60	0.60	0.65	0.65	0.69	0.69

Table 1: *Rendering performance in seconds per frame. The abbreviated rendering parameters are: NC: no opacity correction, OC: opacity correction, SB: slice-sized buffer slices, IB: intermediate image-sized buffer slice.*

the dataset. Pre-integration is fastest with nearest-neighbor interpolation in the pre-integration lookup table, no opacity correction, and slice-sized buffer slices. Slice-sized buffer slices are slightly faster than intermediate image-sized buffer slices because the computation of the location within the buffer slices is simpler, but the performance difference is less than 1% and the resulting images are identical. In the performance tests, opacity correction accounts for 6-13% of the rendering time if enabled, bilinear interpolation in the pre-integration lookup table results in 14-20% performance penalty.

4.2. Image Quality

A number of images, which result from different combinations of rendering parameters are presented on the color page. The images were rendered with the same datasets and output image resolution as in the performance tests, but we selected different transfer functions in order to emphasize the differences of the applied algorithms. The inset in the top right corner of every image shows a magnification of the region highlighted by a black square.

In Figure 7, the Engine dataset is depicted using three different settings. Figure 7a was created by the standard shear-warp algorithm without any of the extensions presented in this paper. For Figure 7b, we used the pre-integrated rendering algorithm with nearest-neighbor interpolation in the pre-integration table and no opacity correction. Figure 7c was computed using the same settings, except that opacity correction was enabled. The difference between the standard algorithm and pre-integration is clearly visible: the engine's features are depicted much smoother and show more detail with pre-integration. The impact of opacity correction can clearly be seen by comparing Figures 7b and 7c: the semi-transparent engine block is more opaque in Figure 7c.

For the creation of the images of the Brain dataset in Figure 8, the same pre-integration settings were applied as for the Engine. Here, the subtle details on the cheek, which is enlarged in the inset, can only be seen with pre-integration. Again, opacity correction makes a difference, but due to the nature of the selected transfer function, it can not be seen as clearly as in the previous example.

The images of Figure 9 depict the Bonsai dataset. They were rendered using the same pre-integration settings as before. Pre-integration accounts for significantly less staircasing artifacts on the flower pot than the standard algorithm, as can be seen very well in the inset. Furthermore, the color difference between the standard and the pre-integrated algorithm, as was mentioned in Section 2.4, is clearly visible, especially in the leaves.

In Figure 10, texturing hardware was employed for rendering the Bonsai dataset with the same transfer functions, the same viewpoint, and the same volume resolution as for the shear-warp. In Figure 10a, 128 image plane aligned textured polygons were rendered, which is the same amount of slices as were composited for the shear-warp algorithm. The texturing hardware's capability of performing trilinear interpolation while compositing and sampling at image resolution result in a clearer image than the shear-warp can achieve even with pre-integration. However, staircasing artifacts are obvious in the resulting image. For Figure 10b, 256 textures were rendered. This reduces the staircasing artifacts significantly, but they are still noticeable, even more clearly than in the images rendered by the pre-integrated shear-warp algorithm. Figure 10c demonstrates that 1024 textured polygons result in an image of high quality.

5. Conclusions and Future Work

We have presented the integration of the pre-integrated volume rendering approach into the shear-warp algorithm. Pre-integration imposes a noticeable performance hit on the standard shear-warp algorithm, but it results in substantially improved image quality. Staircasing artifacts are reduced and color transitions are more accurate.

In the future, we are planning to integrate shading, which is essential for the rendering of iso-surfaces. Also, pre-integration can be incorporated into the perspective projection shear-warp algorithm analogously to the case of parallel projection; however, the scaling of the slices slightly

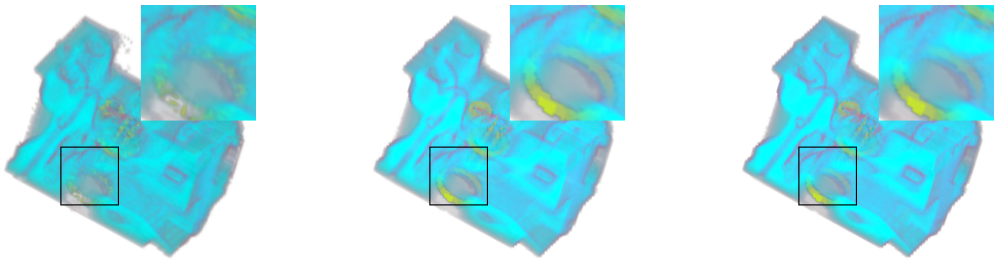
increases the complexity of this approach. The image quality of our algorithm could be further improved by sampling at (or above) the Nyquist frequency of the scalar volume data. This could be accomplished by adapting the approach of Sweeney and Mueller [20].

6. Acknowledgements

This work was partially funded by the German research council (DFG) in the collaborative research center (SFB) 382. We would like to thank Günter Knittel for the fruitful discussion about the appropriate sampling rate for volume rendering with post-classification.

References

1. B. Csebfalvi. *Fast Volume Rotation using Binary Shear-Warp Factorization*. Eurographics Data Visualization '99 Proceedings, pp. 145–154, 1999.
2. Y. Dobashi, T. Yamamoto, and T. Nishita. *Interactive Rendering of Atmospheric Scattering Effects Using Graphics Hardware*. Proceedings of Eurographics/SIGGRAPH Workshop on Graphics Hardware '02, pp. 99–108, 2002.
3. K. Engel, M. Kraus, and T. Ertl. *High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading*. Proceedings of Eurographics/SIGGRAPH Workshop on Graphics Hardware '01, pp. 9–16, 2001.
4. S. Guthe, S. Roettger, A. Schieber, W. Strasser, and T. Ertl. *High-Quality Unstructured Volume Rendering on the PC Platform*. Proceedings of Eurographics/SIGGRAPH Workshop on Graphics Hardware '02, pp. 119–125, 2002.
5. S. Guthe, M. Wand, J. Gonser, and W. Strasser. *Interactive Rendering of Large Volume Data Sets*. IEEE Visualization '02 Proceedings, pp. 53–60, 2002.
6. T. He and A. Kaufman. *Fast Stereo Volume Rendering*. IEEE Visualization '96 Proceedings, pp. 49–56, 1996.
7. G. Knittel. *Using Pre-Integrated Transfer Functions in an Interactive Software System for Volume Rendering*. Short Papers Proceedings Eurographics '02, pp. 119–123, 2002.
8. P. Lacroute. *Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation*. Doctoral Dissertation, Technical Report CSL-TR-95-678, Stanford University, 1995.
9. P. Lacroute and M. Levoy. *Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation*. ACM SIGGRAPH 94 Proceedings, pp. 451–457, 1994.
10. N. Max, P. Hanrahan, and R. Crawfis. *Area And Volume Coherence For Efficient Visualization Of 3D Scalar Functions*. ACM Computer Graphics (San Diego Workshop on Volume Visualization) 24(5), pp. 27–33, 1990.
11. M. Meissner, S. Guthe, and W. Strasser. *Interactive Lighting Models and Pre-Integration for Volume Rendering on PC Graphics Accelerators*. Graphics Interface '02 Proceedings, pp. 209–218, 2002.
12. M. Meissner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis. *A Practical Evaluation of Popular Volume Rendering Algorithms*. IEEE Vol. Vis. 2000 Proceedings, pp. 81–90, 2000.
13. M. Meissner, U. Kanus, G. Wetekam, J. Hirche, A. Ehlert, W. Strasser, M. Doggett, P. Forthmann, and R. Proksa. *VIZARD II: A Reconfigurable Interactive Volume Rendering System*. Proceedings of Eurographics/SIGGRAPH Workshop on Graphics Hardware '02, pp. 137–146, 2002.
14. H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler. *The VolumePro Real-Time Ray-Casting System*. ACM SIGGRAPH 99 Proceedings, pp. 251–260, 1999.
15. S. Roettger and T. Ertl. *A Two-Step Approach for Interactive Pre-Integrated Volume Rendering of Unstructured Grids*. IEEE VolVis '02 Proceedings, 2002.
16. S. Roettger, M. Kraus, and T. Ertl. *Hardware-Accelerated Volume And Isosurface Rendering Based On Cell-Projection*. IEEE Visualization '00 Proceedings, pp. 109–116, 2000.
17. J.P. Schulze and U. Lang. *The Parallelization of the Perspective Shear-Warp Volume Rendering Algorithm*. Proceedings of the 4th Eurographics Workshop on Parallel Graphics and Visualization, pp. 61–69, 2002.
18. J.P. Schulze, R. Niemeier, and U. Lang. *The Perspective Shear-Warp Algorithm in a Virtual Environment*. IEEE Visualization '01 Proceedings, pp. 207–213, 2001.
19. K.S. Shanmugam. *Digital and Analog Communication Systems*. John Wiley and Sons, 1979.
20. J. Sweeney and K. Mueller. *Shear-Warp Deluxe: The Shear-Warp Algorithm Revisited*. Joint Eurographics - IEEE TCVG Symposium on Visualization, May 2002, Barcelona, Spain, pp. 95–104, 2002.
21. M. Weiler, M. Kraus, and T. Ertl. *Hardware-Based View-Independent Cell Projection*. ACM Volume Visualization and Graphics Symposium '02 Proceedings, pp. 13–22, 2002.
22. S.Y. Yen, S. Napel, and G.D. Rubin. *Fast Sliding Thin Slab Volume Visualization*. Symposium on Volume Visualization '96 Proceedings, ACM Press, pp. 79–86, 1996.

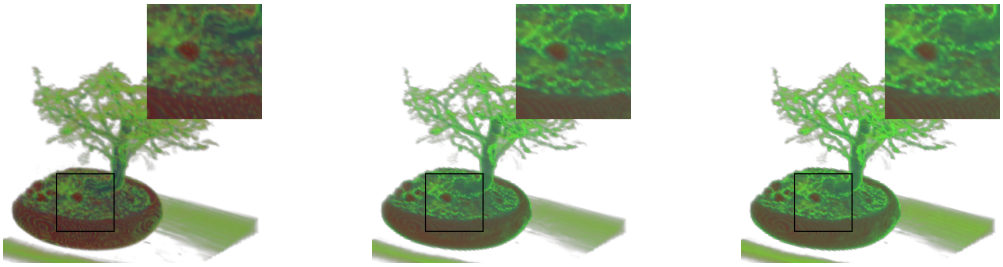


(a) (b) (c)

Figure 7: *The Engine dataset: (a) standard shear-warp, (b) pre-integrated shear-warp without opacity correction and (c) pre-integrated shear-warp with opacity correction.*

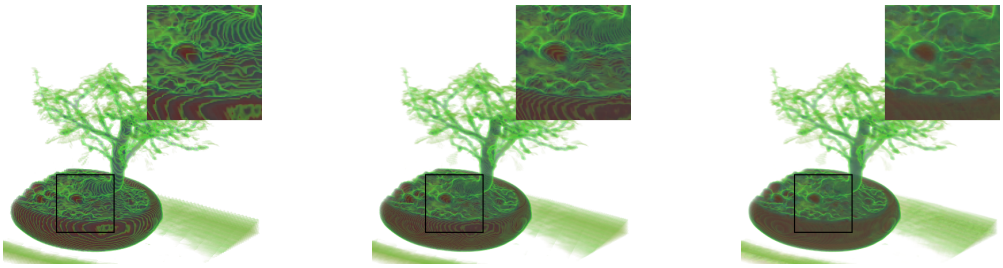
(a) (b) (c)

Figure 8: *The Brain dataset: (a) standard shear-warp, (b) pre-integrated shear-warp without opacity correction and (c) pre-integrated shear-warp with opacity correction.*



(a) (b) (c)

Figure 9: *The Bonsai dataset: (a) standard shear-warp, (b) pre-integrated shear-warp without opacity correction and (c) pre-integrated shear-warp with opacity correction.*



(a) (b) (c)

Figure 10: *The Bonsai dataset rendered with 3D texturing hardware support using different numbers of textured polygons: (a) 128 polygons, (b) 256 polygons, (c) 1024 polygons.*