Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization (2004)
O. Deussen, C. Hansen, D.A. Keim, D. Saupe (Editors)

High-Quality Lighting and Efficient Pre-Integration for Volume Rendering

Eric B. Lum, Brett Wilson, and Kwan-Liu Ma

{lume, wilson, ma}@cs.ucdavis.edu
The Department of Computer Science, The University of California, Davis.

Abstract

Pre-integrated volume rendering is an effective technique for generating high-quality visualizations. The pre-computed lookup tables used by this method are slow to compute and can not include truly pre-integrated lighting due to space constraints. The lighting for pre-integrated rendering is therefore subject to the same sampling artifacts as in standard volume rendering. We propose methods to speed up lookup table generation and minimize lighting artifacts. The incremental subrange integration method we describe allows interactive lookup table generation in $O(n^2)$ time without the need for approximation or hardware assistance. The interpolated pre-integrated lighting algorithm eliminates discontinuities by linearly interpolating illumination along the view direction. Both methods are applicable to any pre-integrated rendering method, including cell projection, ray casting, and hardware-accelerated algorithms.

1. Introduction

Pre-integrated volume rendering [MHC90, EKE01] is a commonly used technique for improving the quality of volume renderings. Because much of the necessary computation is done in advance, this method can generate high-quality images with better performance than heavily super-sampling the volume. It has been used successfully with many types of rendering algorithms including cell projection, ray casting, and hardware-accelerated methods. Unfortunately, the pre-integrated lookup table can take a long time to compute and can not incorporate lighting due to space constraints. The long computation times limit interaction with the transfer function, and current lighting approximations are not general and can introduce artifacts.

We address the problems of table generation speed and high-quality lighting in pre-integrated volume rendering. The proposed interpolated pre-integrated lighting method uses linear interpolation for lighting values between samples along the view direction to achieve smoothly varying results. We also present an $O(n^2)$ alternative to the $O(n^3)$ brute-force algorithm for computing pre-integration tables that we call incremental subrange integration. This method does not make use of approximations and permits the interactive generation of these tables in software for transfer functions with a large number of entries.

2. Pre-Integrated Volume Rendering

Volume rendering consists of integrating color and opacity values across a 3D space. This integration is often performed by sampling the volume at regular intervals: hardware algorithms slice through the volume using a stack of closely-spaced polygons, while software algorithms typically sample viewing rays at regular intervals. According to sampling theory, it is sufficient to sample at the resolution of the scalar field to avoid aliasing with respect to scalar value. In volume visualization, however, the scalar field is sampled before being transformed by a transfer function. This transfer function may add arbitrary frequencies to the data, requiring much higher sampling rates to capture all details.

Consider a very thin surface resulting from a thin spike in the transfer function. If the feature is smaller than the sample spacing, as is often encountered when viewing isosurfaces or somewhat thicker "isoslabs," some rays will sample the detail while others will miss it completely, as illustrated in Figure 1. The result will be a series of aliasing bands rather than a continuous surface. If the feature is somewhat larger

© The Eurographics Association 2004.

than the sample spacing, a similar problem will still occur because some rays will sample the object once while others will sample it twice.

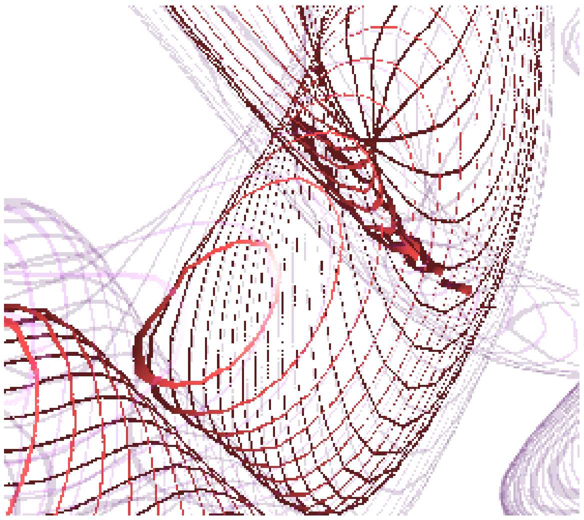


Figure 1: *For transfer functions with spikes, such as this example of two isosurfaces, regular sampling can miss most of a feature, resulting in significant artifacts. Pre-integrated rendering solves the transfer function integration problem.*

Such artifacts can be reduced in traditional volume renderings by sampling at very high rates and using smooth, low-frequency transfer functions such as Gaussian curves to blur the edges of the features. Unfortunately, spread-out transfer functions limit the types of renderings that can be made at high quality, and very high sampling rates limit the performance of both hardware and software implementations while still not guaranteeing sufficient sampling.

The idea behind pre-integrated volume rendering is to calculate the volume rendering integral for pairs of sample values in advance. During this computation, the transfer-function space can be analytically integrated or adaptively sampled at as high a rate as necessary to incorporate all features of the transfer function. At rendering time, a sampling "slab" of data is considered rather than individual sampling points. The volume is sampled at the front and back plane of the sampling slab. These values are then used as indices into a two-dimensional table which stores the pre-computed volume rendering integral for the transfer function between the two samples.

Pre-integrated rendering grew out of work rendering tetrahedral meshes [MHC90, SBM94]. It was applied as an enhancement to the Projected Tetrahedra algorithm using 3D texture hardware for enabling the use of arbitrary transfer functions and for rendering isosurfaces without reconstructing them geometrically [RKE00]. A later enhancements has optimized lookup table creation and final rendering using 2D texture hardware [RE02].

Hardware-accelerated pre-integration was applied to the rendering of regular-grid volume data and isosurfaces by Engel et al. [EKE01]. Higher-quality output was achieved by Roettger, et al. [RGW*03] through accurate clipping and super-sampling to improve lighting. Methods have also been proposed to combine pre-integration with the shear-warp algorithm [SKLE03] and for the pre-integrated rendering of multi-dimensional data [KPI*03].

Although pre-integrated volume rendering is usually used to address sampling problems, it does have sampling problems of its own due to the assumption that the data varies linearly. This assumption occurs in two places. First, data values are interpolated linearly as the pre-integrated table is computed. Accounting for the additional data values needed for higher-order interpolation would require a similarly-higher-dimensional lookup table and would quickly become unwieldly. Second, the data is usually interpolated to get each sample value used for lookup into the pre-integrated table.

2.1. Lighting Limitations of Pre-Integrated Volume Rendering

Lighting presents a problem in pre-integrated volume rendering. The pre-calculated integral in the lookup table is based only on pairs of scalar values, not normals. Integrating three-component normals into the pre-integrated lookup table requires four values each for the front and back samples $(\text{scalar}, N_x, N_y, N_z)$ giving an eight-dimensional lookup table, far too large for a practical implementation.

Engel, et al. address this problem by sampling the lighting at one location (discussed below) and handling isosurfaces as special cases. Because isosurfaces are infinitely thin, it is possible to compute the lighting at exactly one location, the surface itself, for proper results. Isosurface rendering is often the case where precise lighting matters the most, and this method produces very accurate results.

The case of multiple transparent isosurfaces is problematic. Meißner, et al. [MGS02] propose using the closest isosurface for shading, but if two such surfaces intersect one sampling interval, only one will be used for normal calculation as in the top two rays of Figure 2. This will not usually introduce artifacts because the normals of the two surfaces are likely to be very close, but a problem occurs if one surface splits off into its own sampling interval as in the bottom ray of Figure 2. Now the normals for both surfaces will be used. The abrupt change from using only one surface to using both surfaces for normal calculation can cause artifacts.

To avoid these discontinuities in the case of multiple transparent isosurfaces, there must always be one sample per surface. It may seem that such sampling can be accomplished using multiple passes, one for each isovalue. But

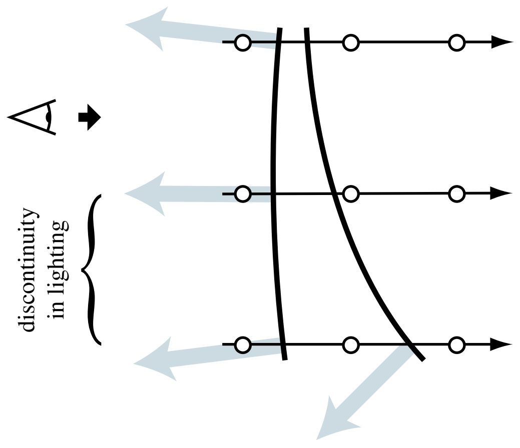


Figure 2: *Special-case isosurface handling can still produce lighting artifacts if multiple transparent isosurfaces intersect the same sampling interval. If one of the surfaces moves to another sampling interval in an adjacent view ray, lighting calculated from the normals (gray arrows) may change abruptly.*

multiple passes do not respect the ordering of the surfaces, which must be drawn from back-to-front for proper compositing. Correct sampling of the lighting of each surface requires looping over each isosurface, an operation that is not efficiently supported in current hardware pixel shaders.

For non-isosurface data, Engel, et al. calculate the lighting from the normal computed at the center of the sampling slab by averaging the front and back normals. This lighting method is equivalent to standard texture-based hardware-accelerated volume rendering with the benefits of pre-integration of color values. Unfortunately, it can introduce distracting visual artifacts because the normal direction can change abruptly between sample slabs. Figure 8(b) and Color Plate 3 show the banding artifacts that can result from this type of sampling. They are especially noticeable on abrupt transitions and can obscure the shape of important features.

Roettger et al. acknowledge this problem and propose supersampling between the front and back samples. This method can reduce such artifacts, and they propose sampling four times over the Nyquist rate as a practical level for high-quality images. However, the required level of sampling is dependent on the projected size of a voxel, no level of sampling can guarantee the result will be free of artifacts, and calculating the additional samples has performance overhead. It would be desirable to achieve high-quality lighting without the need for oversampling.

3. Interpolated Pre-Integrated Lighting

For high-quality lighting, one of the most important requirements is that it varies continuously. The lighting should change consistently as a feature passes from one sampling slab to the next and as the view changes. Sampling the lighting at one or even several points inside the slab may produce sharp changes in lighting, and, as discussed above, true pre-integrated lighting is not feasible due to its high space requirements.

Our solution to the lighting problem is to interpolate lighting values between the front and the back sample planes. The challenge is to properly combine these interpolated lighting values with pre-integrated densities and colors. Instead of using one lookup table as in standard pre-integrated rendering, our method uses two pairs of tables. One pair represents the diffuse and specular contribution, respectively, weighted toward the front sample and the other pair represents the diffuse and specular contribution weighted toward the back sample.

The diffuse lookup table contains the volume rendering integral computed between all possible pairs of samples using the color as specified in the transfer function. The specular lookup table is computed in an identical way but using white for the color over the entire range. Each of these integrals is weighted toward the front or back sample such that their (r,g,b) sum is equal to the unweighted integral.

The rendered result is calculated as a combination of the diffuse and specular contributions of the front and back samples. The rendering process is as follows:

- Retrieve the front sample value i and the back sample value j for the current ray and sampling slab.
- Look up the front-weighted pre-integrated diffuse and specular values and the corresponding back-weighted values. These tables are indexed by (i,j) and contain (r,g,b) values.
- Look up or compute the normals at the two sampling points, N_i and N_j. Multiply each diffuse and specular lighting contribution by the corresponding diffuse and specular table value to get the diffusely $(Diffuse_i, Diffuse_j)$ and specularly $(Specular_i, Specular_j)$ lit material for each.
- The front- and back-weighted colors with lighting (C_i and C_j, respectively) are the sum of the corresponding diffuse and specular colors.
- The output color is then the sum of the front- and back-weighted lit values: $C_{output} = C_i + C_j$. This represents the total pre-integrated color value plus interpolated specular and diffuse lighting.
- The opacity is not affected by lighting. The un-weighted opacity is stored in one of the tables and copied to the output.

The result is analogous to Gouraud shading in which the lighting value is interpolated rather than the normal. This

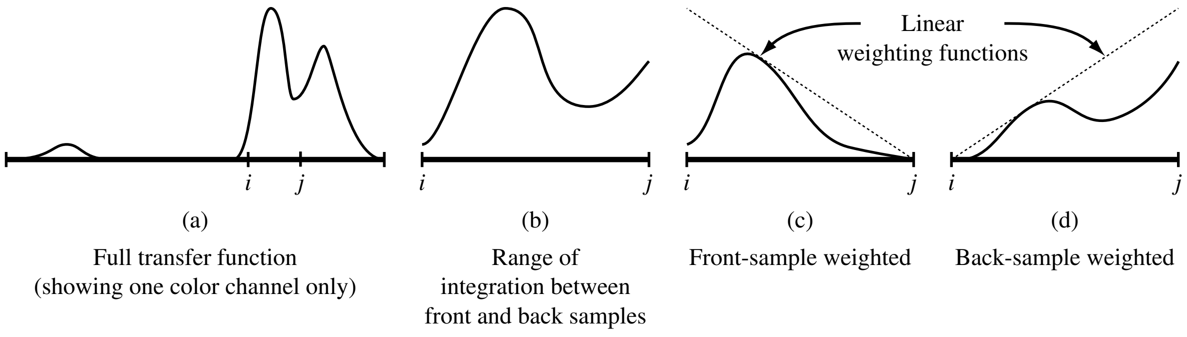


Figure 3: *To calculate the lookup tables for two samples i and j, the transfer function red, green, and blue channels are scaled by two ramp functions, one for each lookup table. The volume rendering integral is then computed over these modified transfer functions for each (i, j) pair.*

algorithm generally produces fewer artifacts than Gouraud shading, however, because only the lighting is interpolated and then only in the view direction. True Gouraud shading interpolates both per-vertex lighting and color in the view plane.

4. Incremental Subrange Integration

4.1. Traditional Lookup Tables

Before considering lit pre-integrated volume rendering lookup tables, it is important to understand how traditional pre-integrated lookup tables are computed. Recall that the goal is to compute the volume rendering integral between a front sample value *i* and a back sample value *j* using a given transfer function. The transfer function may be analytically defined, but typical transfer functions consist of a table of color and opacity values for each possible data value. The interpolation method used between transfer function entries is usually nearest or linear interpolation. The integrals over these small spans are composited together to get the integral over larger spans of the transfer function.

The distance between any pair of samples along the view ray is not necessarily constant. In projected tetrahedra algorithms, for example, the distance is the thickness of the tetrahedra at that point. This means that the table must have a third axis to take into account every possible sample spacing. However, typical ray-casting and 3D texture-based hardware renderers such as ours use a constant world-space sample distance and a two-dimensional table is sufficient.

As the volume rendering integral between i and j is evaluated over the transfer function, there will be $|j-i|$ values composited together, each of which represent a fixed distance in world space. Each opacity must therefore be corrected according to the distance that each composited value represents in world space. (The need for this correction is most readily apparent in the case of a constant transfer function, where the integral between any i and any j should be identical.) Given a world-space sampling slab width Δ, a number of composited transfer function intervals $|j-i|$, and an opacity value from the transfer function which represents opacity per unit in world space, the correction is:

$$\alpha_{\text{for compositing}} = 1 - (1 - \alpha_{\text{from xfer func}})^{\frac{\Delta}{|j-i|}} \qquad (1)$$

This correction must be applied to the α values *before* compositing; it can not be applied to a composited sequence of color values.

Computing the integral between two arbitrary sample values for a transfer function of m entries requires $O(m)$ time. Computing integrals for all possible entries in an $n \times n$ lookup table by brute force therefore requires $O(n \times n \times m)$ time. Given that the table dimension is usually the same as, or a small constant factor of, the size of the transfer function, this algorithm requires $O(n^3)$ time. On a 2.2 GHz Athlon FX-51, this takes 1.5 seconds to run for $n = 256$, clearly too slow for interactive transfer function manipulation.

4.2. Efficient Computation of Traditional Lookup Tables

It is important that the computation of the lookup tables be efficient so that the transfer function may be interactively modified. Engel, et al. discuss a method for quickly approximating the integration step for traditional pre-integrated lookup tables in $O(n^2)$ time by neglecting attenuation within a sampling slab. However, this simplification can cause ordering problems, especially if there are multiple spikes in the transfer function such as thin “isoslabs.” Roettger and Ertl propose a method that uses the graphics hardware to more quickly compute the lookup tables.

We propose an algorithm called incremental subrange integration that computes the exact $n \times n$ lookup table in $O(n^2)$

time. This algorithm can be used as-is when implementing traditional pre-integrated volume rendering, or as a first step in computing the weighted lookup tables used for lighting calculations (discussed below). It can also be used to calculate three-dimensional lookup tables for projected tetrahedra rendering in $O(n^3)$ time rather than the $O(n^4)$ time required by the brute-force method.

The problem with the brute-force method is that the integral for many small ranges over the transfer function must be repeatedly calculated. Ideally, results would be re-used in subsequent steps, but the non-linear opacity correction in Equation 1 depends on the number of ranges of the transfer function that are being considered. Because this dependency can not be factored out, it is not always possible to add new intervals to the beginning or end of a previously computed integral: the opacity scaling for the previously computed color values would be wrong.

To re-use previously computed entries when generating the table, each opacity, and therefore the value $|j-i|$, must not change. This quantity is constant along the table diagonals, so it is possible to composite additional values as long as the final result has $|j-i|$ composite steps. Unfortunately, it is not always possible to "uncomposite" values, so having a value for the integral over $i \rightarrow j$ will not help in integrating over $(i+1) \rightarrow (j+1)$. We therefore have developed a method that computes these integrals in parts.

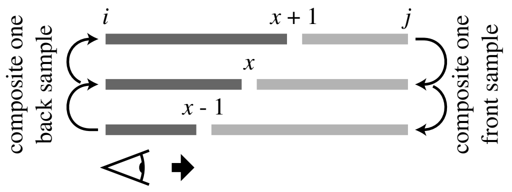


Figure 4: *The update rule for subranges. A left-hand subrange can be expanded one step in the transfer function to the right, and a right-hand subrange can be expanded one step to the left. This process will eventually give all pairs of subranges between i and j.*

The strategy is to compute the integral over a certain range $i \rightarrow j$ in two subranges. Each subrange is computed in constant time and can be reused for the calculation of additional subranges. First, consider some x where $i \leq x \leq j$, there is a left-hand subrange $i \rightarrow x$. Given the integrated value of this subrange, it can be expanded by one to the right by compositing a new value behind it and getting the integral over $i \rightarrow (x+1)$. Likewise, the right-hand subrange $x \rightarrow j$ can be expanded by compositing one value in the front to get an integral over $(x-1) \rightarrow j$. This process is shown in Figure 4.

For the case where $i > j$, corresponding to the values in the pre-integrated lookup table below the diagonal, the view direction is reversed. In these cases, the integrals between transfer function samples and the composite operations must be computed in the opposite direction.

The key to the $O(n^2)$ execution time of this algorithm is that the subrange integrals are organized in a manner that allows for their incremental constant-time computation. For each diagonal of the pre-integrated lookup table, a subrange-integral table is constructed consisting of a series of subranges. Consider the case where $i < j$ and are separated by length n_r, which corresponds to all entries on the diagonal starting at $(i=0, j=n_r)$ and extending to the upper-right. The transfer function with n_t entries is separated into bins of length n_r. Each bin has its subrange integrals incrementally computed from the left to right and right to left, and the intermediate results are stored as shown in Figure 5. Two subrange-integral pairs from adjacent bins can then be composited to construct integrals of length n_r. Since there are $2n_r$ integration values stored in each bin, and n_t/n_r bins, the total number of entries in each sub-integral table is $O(n_t)$. The geometric intuition behind the linear size of the sub-integral tables is that in cases where the length of integration n_r is small, each bin contains few entries, but there are many bins. Alternatively, as n_r approaches the transfer function length n_t, fewer bins are required, but each bin contain more entries. Two examples are shown in Figure 5.

Using the subrange-integral table, a complete pre-integrated lookup table can be computed in $O(n^2)$ time. Each bin requires $2n_r$ constant-time composite operations and there are n_t/n_r bins, so each range-integral table requires $2n_t$ composite operations. For an $n \times n$ pre-integrated lookup table, there are $2n$ diagonals, each of which requires a single range-integral table. The entire lookup table can therefore be computed in $(2n_t)(2n) = O(n^2)$ time.

4.3. Efficient Computation of Weighted Lookup Tables

The front-weighted and back-weighted lit pre-integration tables are like the standard pre-integration tables but with an additional scaling function applied to each integral. As each transfer function sample is evaluated and composited with the previous values, the (r,g,b) color (not opacity) is multiplied by a ramp function as illustrated in Figure 3 (c) and (d). For the two front-weighted lookup tables, this ramp function has a value of one at the front sample and a value of zero at the back sample. For the two back-weighted lookup tables, the ramp function is reversed. Since the front-weighted and the back-weighted scaling factors always sum to one for a given transfer function sample, adding the (r,g,b) components of the two lookup tables gives the standard pre-integrated lookup table.

The goal is to generate four tables, two specular and two diffuse. Only the two diffuse lookup tables will be discussed here; the specular tables are computed identically except with the appropriate specular rather than diffuse colors. For white specular lighting, the specular weighted lookup tables

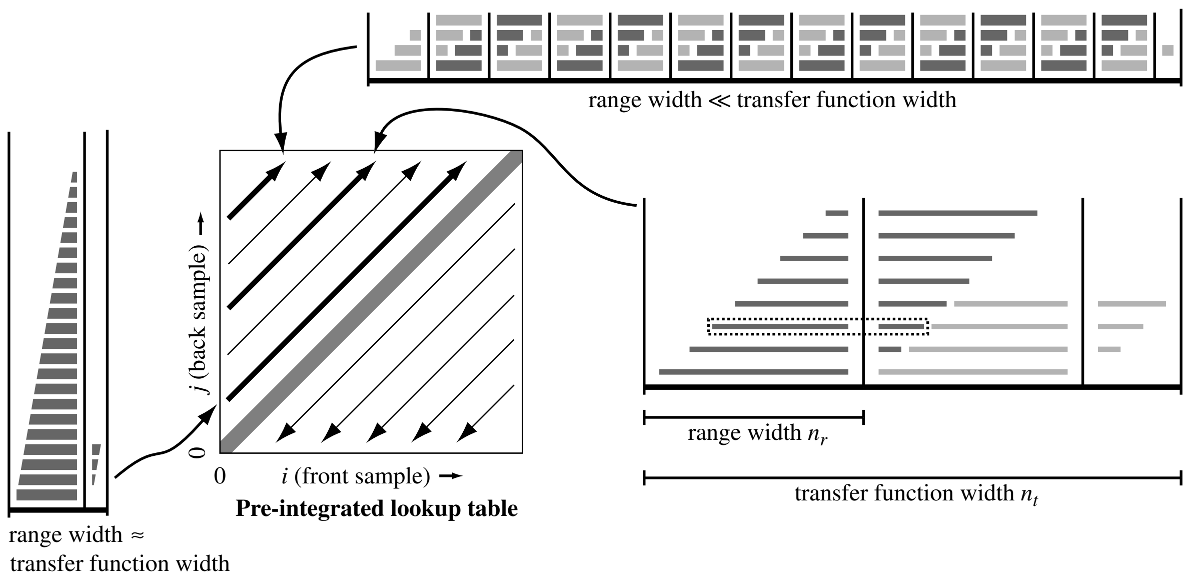

range width ≪ transfer function width
j (back sample) →
0
0
i (front sample) →
Pre-integrated lookup table
range width ≈
transfer function width
range width n_r
transfer function width n_t

Figure 5: *The pre-integrated lookup table showing the direction of computation along the table diagonals. Each diagonal (corresponding to integrals of length n_r) requires one subrange-integral table. Each subrange-integral table contains all length n_r integrals over the transfer function in one direction. One such integral is shown as a dotted box in the lower-right table.*

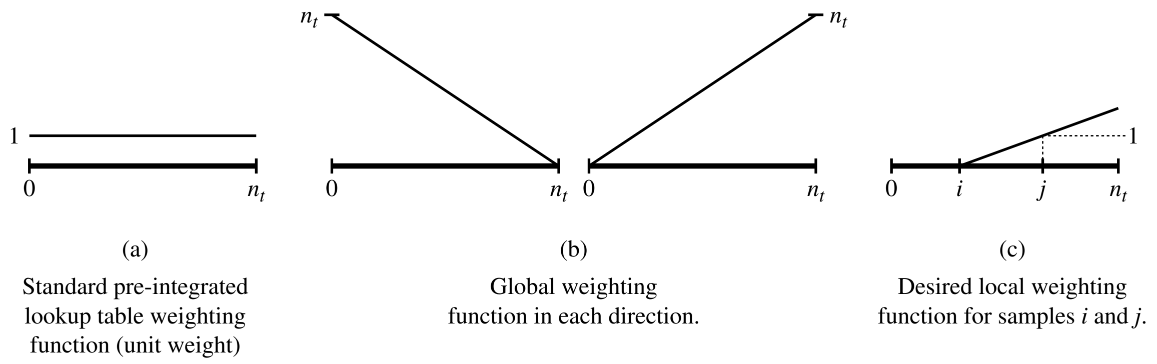

1
0
n_t
(a)
Standard pre-integrated lookup table weighting function (unit weight)
n_t
0
n_t
n_t
0
n_t
(b)
Global weighting function in each direction.
1
0
i
j
n_t
(c)
Desired local weighting function for samples i and j.

Figure 6: *The standard, unweighted pre-integrated lookup table is shown in (a) for a transfer function of n_t elements. The two modified weighting functions are shown in (b), which are used to generate two intermediate tables. The desired function shown in (c) uses weighting functions that depend on the range of integration for each element.*

are computed by replacing all transfer function colors with white.

Efficiently computing these weighted tables presents a challenge since the weighting function changes for every entry; it ramps from zero at one end of each integral to one at the other end, so it depends on both front and back sample values. We compute the weighted lookup tables for lighting in $O(n^2)$ time using three intermediate tables.

The first intermediate table is the standard pre-integrated lookup table discussed previously. The other two intermediate tables contain the integrals computed with the two modified weighting functions. The transfer function colors are multiplied with the ramped weighting functions shown in Figure 6(b). In more concrete terms, each color value is multiplied by its index (or n_t minus its index for the opposite weighting direction) into the table. The pre-integrated table computation procedure is run on these modified transfer functions to get globally-ramped weighted tables.

What we have now are two tables that use a globally ramped weighting function which does not depend on where the integration starts and ends. The final tables needed for rendering use weighting functions that *do* depend on the range of integration. Fortunately, the needed tables can be easily computed from the globally ramped integration tables.

Consider one case where $i < j$ and we are calculating the back-weighted table. The desired weighting function for this range ramps from 0 at $W(i)$ to 1 at $W(j)$, giving $W(x) = (x-i)/(j-i)$. To simplify the equations, the opacity factors of each term will be abbreviated A_x, where

$$A_x = (1-\alpha_i)(1-\alpha_{i+1})(1-\alpha_{i+2})...(1-\alpha_{x-1})\,\alpha_x$$

Our goal is to get the weighted color value

$$C^W_{i\to j} = \frac{0}{j-i}A_iC_i \;+\; \frac{1}{j-i}A_{i+1}C_{i+1} \;+ ... +\; \frac{j-i}{j-i}A_jC_j$$

Factoring out $1/(j-i)$, the goal becomes

$$C^W_{i\to j} = \frac{1}{j-i}\left(0A_iC_i \;+\; 1A_{i+1}C_{i+1} \;+ ... +\; (j-i)A_jC_j\right)$$

Two values have previously been computed. First, we have the unweighted color value C^1 representing the full pre-integrated range over $i \to j$:

$$C^1_{i\to j} = A_iC_i \;+\; A_{i+1}C_{i+1} \;+ ... +\; A_jC_j$$

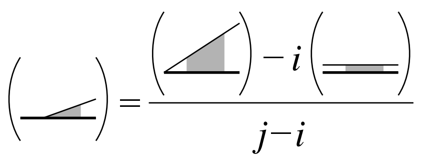
Second, we have the color value C^G which is integrated over $i \to j$ using the globally ramped weighting function:

$$C^G_{i\to j} = iA_iC_i \;+\; (i+1)A_{i+1}C_{i+1} \;+ ... +\; jA_jC_j$$

If we subtract iC^1 from C^G, the effect will be to remove the dependence on i from each term. Dividing the result by $j-i$ gives the desired C^W above:

$$C^W_{i\to j} = \frac{C^G_{i\to j} - iC^1_{i\to j}}{j-i}$$

Another way to look at the operation is geometrically:



For physical intuition as to why these operations can be performed, consider that only (r,g,b) light intensity values are being scaled or added, and not the non-linear opacity attenuation of light.

The cases where $i > j$ can be computed analogously. As an optimization, the front-weighted table can be computed by subtracting the back-weighted table from the full pre-integrated lookup table.

5. Implementation and Results

The proposed lighting method for pre-integrated volume rendering was implemented in a hardware-based volume renderer. It uses three-dimensional textures to store scalar and normal data, and rendering is performed back-to-front with view-aligned slices. It uses conditional execution based on early depth culling, or computation masking [SHN03], to reduce the number of calls to the fragment program for invisible voxels. The computer runs Windows XP on a 2.2 GHz Athlon FX-51 with 2 GB of main memory and has an ATI Radeon 9800 Pro graphics card with 256 MB of video memory.

5.1. Optimizing Texture Lookups

The rendering bottleneck for our implementation on current graphics hardware is the texture lookups. A straightforward approach requires one lookup for each of the two sample values and four lookups for the pre-integrated values: two each for the front and back specular and diffuse components. We reduce the number of texture lookups by two by packing the diffuse and specular pre-integrated results.

The lighting lookup results in five pieces of information: front- and back-weighted diffuse colors, front- and back-weighted specular colors, and the full pre-integrated opacity (lighting does not affect the opacity of the rendered result). All of this data can be packed into two 2D RGBA lookup tables. The front lookup table contains the diffusely-lit, front-weighted color in the (r,g,b) components, and the complete pre-integrated opacity in the α component. The back lookup table contains the diffusely-lit, back-weighted color in the (r,g,b) components, and the specularly-lit, back-weighted, result in the α component. Because the front-weighted and back-weighted integrals should always sum to the full pre-integrated value, the front-weighted specular color is computed by subtracting the back-weighted specular color intensity in the back table from the α stored in the front table (identical to the un-ramped pre-integrated specular color intensity).

5.2. Quality Results

For smooth surfaces with smoothly varying normals, our lighting algorithm produces results indistinguishable from the special-case isosurface rendering algorithm of Engel, et al. (see Color Plates). Both of these algorithms improve upon standard volume rendering and standard pre-integrated volume rendering, shown in Figure 8. Unlike special-case isosurface handling, our method allows an unlimited number of possibly transparent isosurfaces with no ordering problems or lighting discontinuities as in displayed in Figure 7.

When the normal changes rapidly or is poorly-defined, the proposed method can introduce minor lighting artifacts compared to special-case surface rendering. One such case occurs where there are two homogeneous materials separated

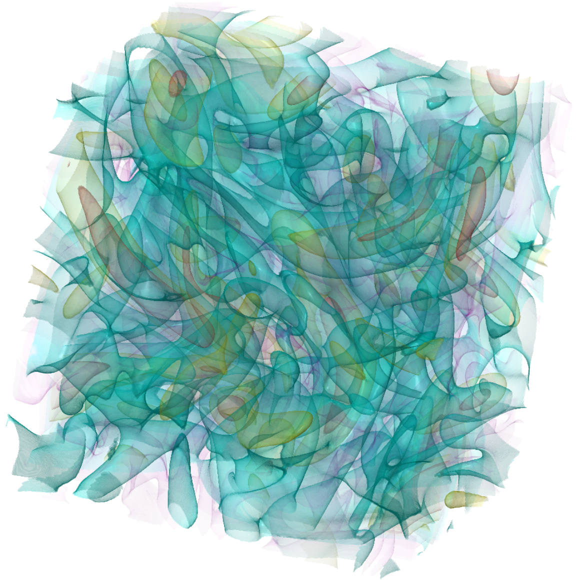


Figure 7: *Multiple transparent isosurfaces rendered with the pre-integrated lighting algorithm. The algorithm can render an unlimited number of isosurfaces with no lighting discontinuities.*

by a sharp boundary. While the normal at the boundary surface is well-defined, the front and back samples surrounding the boundary may occur inside the material where the normal is less well-defined. The resulting lighting may have some randomness associated with it. One such case is illustrated in Figure 8(d). Small horizontal striations are visible in the small surfaces connecting the roots of the upper teeth, and small specular highlights in the roots of the lower teeth deviate from the surface rendering. Fortunately, we found such deviations to be limited and only visible at high levels of magnification.

5.3. Performance Results

Our optimized $O(n^2)$ lookup table computation algorithm greatly improves upon the $O(n^3)$ brute-force method without the need for approximation or hardware-acceleration. The algorithm takes significantly less time than the brute force method, particularly as table size grows as seen in Table 1. Large table sizes are desirable because interpolating adjacent entries, as is often done when a sample value falls between two table entries, will not result in the exact pre-integrated result for the interpolated value. The larger the table, the smaller such interpolation error will be. Cell-projection methods, which require a larger three-dimensional lookup table, will benefit even more.

The rendering algorithm is somewhat slower than the traditional pre-integrated method as shown in Table 2. This is because, for every sampling slab, two lighting values must be computed and one extra texture read is necessary to retrieve the weighted pre-integrated data. Due to the complex fragment program, the algorithm is fill-rate dependent. As more data is drawn to the screen, performance drops relative to standard pre-integrated rendering.

n	**Brute force** $O(n^3)$	**New** $O(n^2)$
32	0.0025	0.00078
64	0.022	0.0033
128	0.18	0.015
256	1.57	0.057
512	12.4	0.22
1024	98.	0.88
2048	797.	3.70

Table 1: *Comparing computation time in seconds for an $n \times n$ pre-integrated lookup table. These times do not include the time required to compute the weighted versions of the table for lighting. The weighted tables require only a small constant factor of extra time to compute.*

		Our method	**Single sample**
Skull	With comp. masking	4.4 fps	5.0 fps
	No comp. masking	2.2 fps	2.6 fps
Vortex	With comp. masking	8.1 fps	10.5 fps
	No comp. masking	5.4 fps	8.1 fps

Table 2: *Comparing frames-per-second for the skull (256×256×256) and vortex flow (128×128×128) data sets rendered to a 512×512 window using interpolated pre-integrated lighting and the traditional method which uses a single sample for computing the lighting. Results are shown with and without computation masking, which eliminates calls to the fragment program for empty voxels.*

6. Conclusions

We have described a method for achieving high-quality lighting effects with pre-integrated volume rendering. Because a fully-lit pre-integrated lookup table is not practical, our method approximates lighting along the volume rendering integral through interpolation. The result is more accurate, smoothly changing lighting with minimal visual artifacts.

Interpolated pre-integrated lighting is appropriate for most types of data and transfer functions, including multiple transparent isosurfaces combined with semi-transparent volume rendering, without the need for supersampling or special handling of certain cases. It integrates well with all pre-

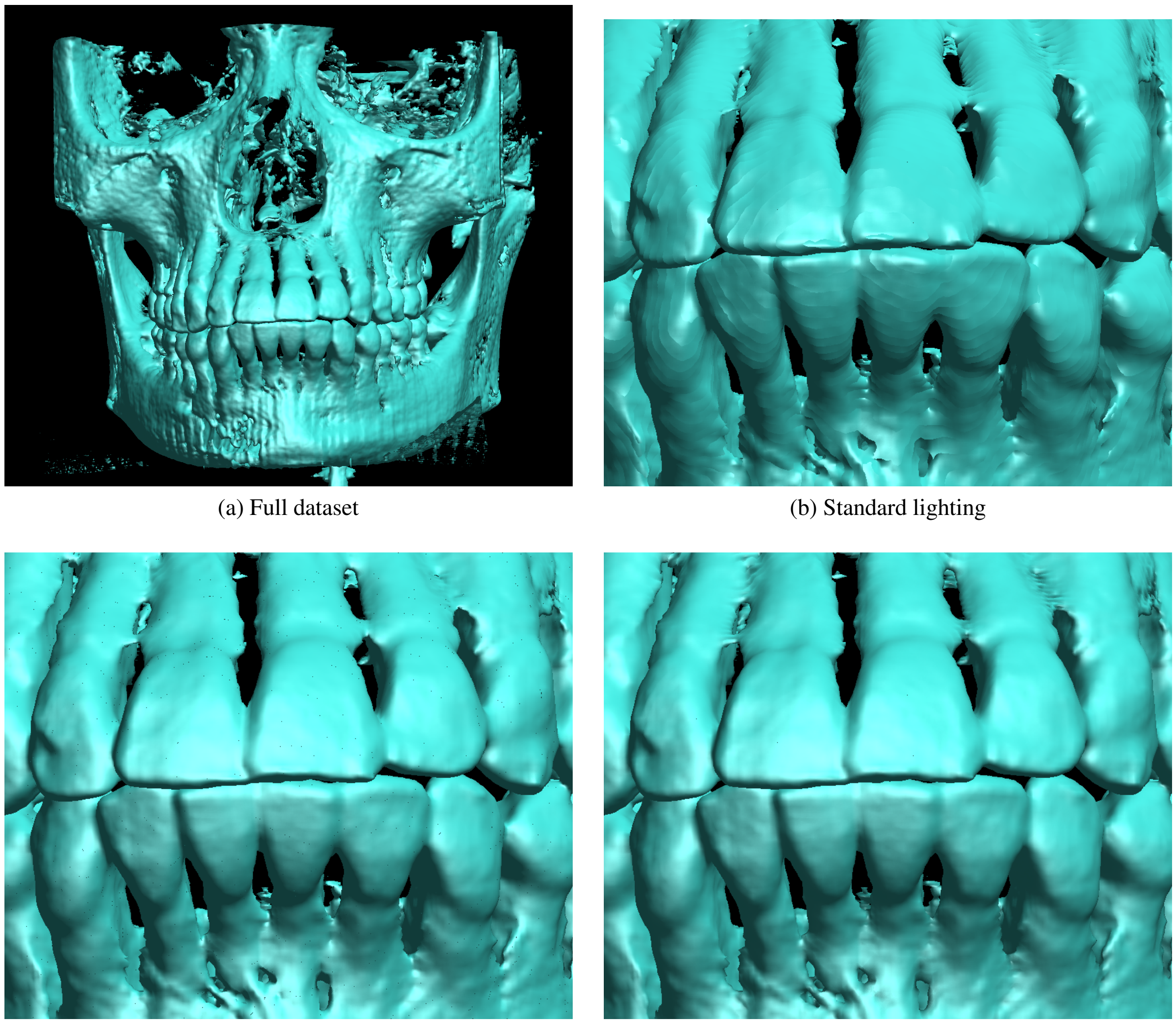

(a) Full dataset
(b) Standard lighting
(c) Special-case surface lighting
(d) Proposed lighting method

Figure 8: *Comparing rendering methods. Our general-purpose lighting method produces much better results than standard lighting. Compared to special-case isosurface rendering, it can introduce minor artifacts where the normal is poorly-defined which can be visible at high magnification.*

integrated rendering methods, including cell projection, ray casting, and hardware-accelerated volume rendering.

We also described an efficient $O(n^2)$ method for computing the lookup tables required to implement pre-integrated volume rendering both with and without lighting. Coupled with hardware-accelerated rendering, very high-quality, real-time exploration and transfer-function manipulation is possible.

Acknowledgments

This work has been sponsored in part by the U.S. National Science Foundation under contracts ACI 9983641 (PECASE award), ACI 0325934 (ITR), and ACI 0222991; the U.S. Department of Energy under Memorandum Agreements No. DE-FC02-01ER41202 (SciDAC program), and No. B523578 (ASCI VIEWS); the LANL/UC CARE program; the National Institute of Health through the Human Brain Project; and a United States Department of Education Government Assistance in Areas of National Need (DOE-GAANN) grant P200A980307. We obtained the turbulent vortex flow data set through the VIZLAB of CAIP at Rutgers University and the skull data set from Siemens Medical Systems in Forchheim, Germany. The authors would also like to thank Mark Segal at ATI and Martin Kraus at Purdue University.

References

[EKE01] ENGEL K., KRAUS M., ERTL T.: High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of Eurographics/SIGGRAPH Workshop on Graphics Hardware* (2001), pp. 9–16.

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

[MGS02] MEISSNER M., GUTHE S., STRASSER W.: Interactive lighting models and pre-integration for volume rendering on PC graphics accelerators. In *Proceedings of Graphics Interface 2002* (2002).

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

[RE02] ROETTGER S., ERTL T.: A two-step approach for interactive pre-integrated volume rendering of unstructured grids. In *Proceedings of the IEEE Symposium on Volume Visualization and Graphics* (2002), pp. 23–28.

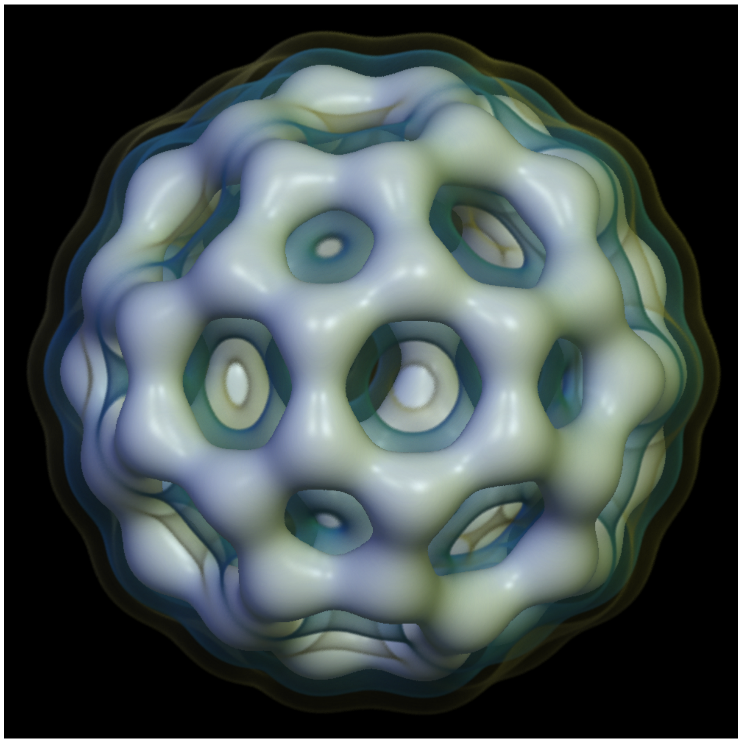
[RGW*03] ROETTGER S., GUTHE S., WEISKOPF D., ERTL T., STRASSER W.: Smart hardware-accelerated volume rendering. In *EUROGRAPHICS/IEEE Symposium on Visualization* (2003), pp. 231–238.

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

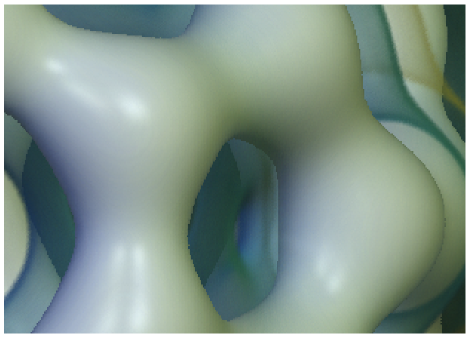
[SBM94] STEIN C., BACKER B., MAX N.: Sorting and hardware assisted rendering for volume visualization. In *Symposium on Volume Visualization* (1994), pp. 83–90.

[SHN03] SHERBONDY A., HOUSTON M., NAPEL S.: Fast volume segmentation with simultaneous visualization using programmable graphics hardware. In *Proceedings of IEEE Visualization* (2003), pp. 171–176.

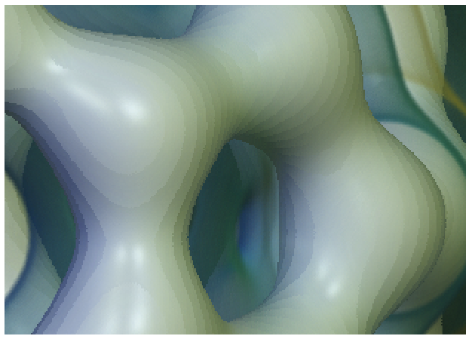
[SKLE03] SCHULZE J. P., KRAUS M., LANG U., ERTL T.: Integrating pre-integration into the shear-warp algorithm. In *Proceedings of the Eurographics/IEEE TVCG Workshop on Volume Graphics* (2003), pp. 109–118.



Color Plate 1: Buckyball dataset rendered with interpolated pre-integrated lighting.



Color Plate 2: Detail view using interpolated pre-integrated lighting.



Color Plate 3: Detail view using traditional sampled lighting.