Gaussian Transfer Functions for Multi-Field Volume Visualization

Joe Kniss[1] Simon Premože[2] Milan Ikits[1] Aaron Lefohn[1] Charles Hansen[1] Emil Praun[2]

[1] Scientific Computing and Imaging Institute, University of Utah
[2] School of Computing, University of Utah

Abstract

Volume rendering is a flexible technique for visualizing dense 3D volumetric datasets. A central element of volume rendering is the conversion between data values and observable quantities such as color and opacity. This process is usually realized through the use of transfer functions that are precomputed and stored in lookup tables. For multidimensional transfer functions applied to multivariate data, these lookup tables become prohibitively large. We propose the direct evaluation of a particular type of transfer functions based on a sum of Gaussians. Because of their simple form (in terms of number of parameters), these functions and their analytic integrals along line segments can be evaluated efficiently on current graphics hardware, obviating the need for precomputed lookup tables. We have adopted these transfer functions because they are well suited for classification based on a unique combination of multiple data values that localize features in the transfer function domain. We apply this technique to the visualization of several multivariate datasets (CT, cryosection) that are difficult to classify and render accurately at interactive rates using traditional approaches.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism I.3.7 [Computer Graphics]: Three-Dimensional Graphics

Keywords: Volume Rendering, Transfer Functions, Multi-field visualization

1 Introduction

Direct volume rendering is a flexible technique for visualizing arbitrary three-dimensional scalar and multi-field datasets. Other 3D visualization techniques require the computation of an intermediate geometric representation of the data prior to rendering (*e.g.* creating a polygonal mesh using isosurface extraction). In contrast, direct volume rendering does not require intermediate geometry; the data is resampled and converted to optical properties as it is being rendered. This conversion from data values to optical properties is represented using a transfer function, which is typically implemented as a lookup table.

One advantage of direct volume rendering is its ability to visualize multiple values, or fields, simultaneously. Multi-field volume rendering has been shown to dramatically improve our ability to classify subtle features that may not be well characterized by any single input field [Laidlaw 1995]. Even scalar datasets can benefit

IEEE Visualization 2003,
October 19-24, 2003, Seattle, Washington, USA
0-7803-8120-3/03/$17.00 ©2003 IEEE

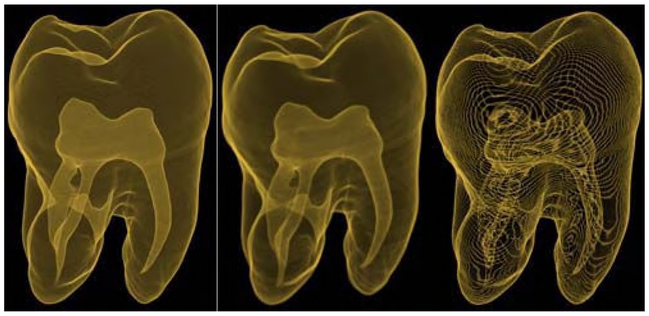

Figure 1: Volumetric rendering using Gaussian Transfer Functions (GTF). Left: analytic approximation of the GTF integral evaluated on graphics hardware (128 slices). Middle: numerical integration of the GTF using 368 slices. Right: numerical integration of the GTF using 128 slices.

from multi-field volume rendering techniques by adding fields for local derivative information [Kindlmann 1999]. For example, gradient magnitude characterizes the rate of change of values in some neighborhood and can help classify the input data set into homogeneous and transition regions [Kindlmann 2002]. Multiple data fields effectively place the ranges of data values representing different features at different locations in a multidimensional data space. Features may therefore be easier to classify in a multivariate dataset because ambiguities can be better resolved when different features share the same range of data values in an individual field.

Although a multi-field dataset can be visualized using separate transfer functions for each field, multidimensional transfer functions that specify the optical properties for each unique combination of data values are a more general and expressive representation [Kniss et al. 2002b; Kniss et al. 2002a]. A major limitation of multidimensional transfer functions using a lookup table is the increased storage requirement. Each additional field in the dataset increases the size of the transfer function lookup table. For instance, a 1D transfer function for eight bit data would require 256 entries, whereas a 2D transfer function requires 256^2 entries. In practice, we have found that it is not uncommon to encounter datasets that require 3D or even 4D transfer functions.

One approach for handling the exponential memory requirements of a multidimensional transfer function is to decompose it into multiple transfer functions of a lower dimension, *i.e.* implement it as a product of separable transfer functions. For instance, a 4D transfer function for data fields d_1, d_2, d_3, d_4 could be represented as a 2D transfer function for fields d_1 and d_2 multiplied with another 2D transfer function for fields d_3 and d_4. Alternatively, this 4D transfer function could be represented as four 1D transfer functions, one for each of the data fields, multiplied together. Although separable transfer functions may reduce the memory requirements of a high dimensional transfer function, they also dramatically limit the kinds of features that can be visualized when compared to a general multidimensional transfer function. Separable transfer functions also have the potential to erroneously clas-

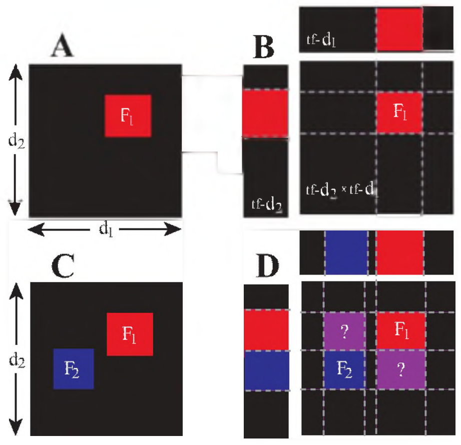

Figure 2: The limitations of separable transfer functions.

sify features. This occurs when the separable parts of a transfer function unintentionally interact when they are combined. Figure 2 illustrates how this unintentional interaction occurs in the transfer function domain. Figure 2(A) shows the desired 2D transfer function classifying some feature F_1 seen in red. Figure 2(B) shows 1D transfer functions for each of the data fields d_1 and d_2 that when multiplied together produce the desired transfer function. However, if we add a second classified feature to the 2D transfer function, F_2 in Figure 2(C), we can no longer represent this transfer function as a product of two 1D transfer functions. Figure 2(D) shows how the 1D transfer functions interact to produce an incorrect 2D transfer function.

Another issue affecting the size of a transfer function lookup table is the dynamic range of the data. The ideal transfer function lookup table should have the same number of entries as the number unique data values. Therefore, a typical transfer function for scalar 8-bit data has 256 entries. Similarly, a transfer function for scalar 16-bit data would need to have 65,536 entries. Most industrial and medical scanners acquire data with a fixed-point dynamic range of at least 12 bits. Nearly all modern numerical simulations produce 32-bit floating-point or double precision floating-point data. Using a transfer function with a different dynamic range and resolution than that of the source data requires scaling the source to the dynamic range and quantizing it to the resolution of the transfer function. It is not always clear how to appropriately map floating point data to a discrete transfer function with finite size.

The dynamic range and dimensionality of the transfer function affect how finely the data volume must be sampled during rendering. Engel *et al.* [2001] observed that the maximum frequency along the viewing ray in volume rendering is the product of the highest frequency in the source data and the highest frequency in the transfer function. Kniss *et al.* [2002b] observed that the maximum frequency along the viewing direction is also proportional to the dimension of the transfer function. These observations imply that it is not sufficient to sample the volume with the Nyquist frequency of the data field, because undersampling artifacts would become visible. This problem is exacerbated if non-linear transfer functions are allowed. That is, the narrower the peak in the transfer function, the more finely we must sample the volume to render it without artifacts. Similarly, as more dimensions are added to the transfer function, we must also increase the sampling rate of the volume rendering.

This paper develops a framework for the compact representation of multidimensional transfer functions. We choose a transfer function based on the Gaussian as the underlying primitive. While many other functions could be used in general, its capability for feature classification and several implementation issues justify our choice of the Gaussian.

First, we show that the Gaussian and its properties enable flexible feature classification. Second, the Gaussian can be efficiently evaluated even for multiple dimensions while preserving its generality and expressiveness. This function can be *explicitly* evaluated which also remedies the size and dynamic range issues of transfer functions that would ordinarily be implemented as lookup tables. Third, mathematical properties of the Gaussian allow us to approximate the volume rendering equation in closed form over a line segment with linearly varying multivariate data values. As shown in Figure 1 this approximation enables us to render high quality images with significantly fewer samples than are required for ordinary numerical integration techniques.

In Section 3 we present a class of transfer functions based on the Gaussian. Section 4 describes how this class of transfer function primitives can be analytically integrated over a line segment under the assumption that data values vary linearly between two sampled points. In Section 5 we describe the practical implementation details of Gaussian transfer functions and their analytic integration. In Section 6 we show examples of multivariate data visualization and compare the described approach with commonly used methods.

2 Previous Work

Visualization of volumetric datasets has been studied extensively. [Blinn 1982] introduced a particle model for computer graphics adapting the radiative transfer theory. [Kajiya and Von Herzen 1984] extended the particle model to inhomogeneous volumetric media. Modern direct volume rendering methods are based on work of [Drebin et al. 1988], [Sabella 1988] and [Levoy 1988]. [Sabella 1988] presented a simple raytracing algorithm for rendering three dimensional scalar fields. The illumination model was based on a varying density emitter and a simple transfer function was applied to the scalar data field for visualization. Accurate and convincing visualization of data sets can be achieved by direct volume rendering if the data is sampled at high rates. Unfortunately, high sampling rates result in performance penalties and therefore slow rendering times. Slow performance is exacerbated if non-linear transfer functions are used and complex optical models [Max 1995] are employed in volume shading computation. [Max et al. 1990] realized the need for pre-integration in the projected tetrahedra (PT) volume rendering algorithm [Shirley and Tuchman 1990]. They analytically integrate the opacity and intensity integrals in the sorted convex polyhedra cells comprising the volume. These individual segments are then composited together using standard compositing operators. Color sample compositing has been derived by [Blinn 1982] and [Porter and Duff 1984]. [Williams and Max 1992] described a simple analytic volume density optical model for piecewise linear transfer functions. [Williams et al. 1998] further improve the accuracy of visualization of unstructured grids by improving pre-integration and adding more sophisticated optical models. Previous volume rendering algorithms only allowed linear transfer functions [Stein et al. 1994]. Arbitrary transfer functions can also be pre-integrated by storing the integral in a three-dimensional texture and later used efficiently in the visualization step [Roettger et al. 2000; Roettger and Ertl 2002]. [Engel et al. 2001] applied the idea of volume pre-integration to regular meshes and employed programmable consumer hardware for visualization. They achieved high-quality visualizations of volume data even for coarse data sets and non-linear transfer functions without the performance penalty. Pre-integration can sometimes result in artifacts if preshaded colors and opacity values are interpolated separately. [Wittenbrink et al. 1998] improved the compositing step by interpolating opacity-weighted color.

Interactive direct volume rendering has become possible by exploiting graphics hardware support for texture mapping. [Cabral et al. 1994] cast the volume rendering problem into a 3D texture resampling problem that can be efficiently implemented in graphics hardware. Numerous other authors made significant improvements to texture based volume rendering such as performance optimizations, sophisticated light and shading models and improved quality [Westermann and Ertl 1998; Meißner et al. 1999; Rezk-Salama et al. 2000; Westermann and Sevenich 2001; Engel et al. 2001].

Transfer functions and methods for generating them have also been extensively studied. [Pfister et al. 2001] and [Kindlmann 2002] provide an excellent survey of existing methods and tradeoffs between them. While 1D and 2D transfer function have received much attention, true multidimensional transfer functions have not. [Laidlaw 1995] developed a framework for magnetic resonance imaging (MRI) classification and visualization using volume rendering algorithms that included 2D Gaussian transfer functions for data classification.

3 Gaussian Transfer Functions

In direct volume rendering, data points are directly mapped to optical properties such as color and opacity that are then composited along the viewing direction into an image. This mapping is achieved using *transfer functions*. These functions have to be able to efficiently classify data features and produce various different outputs such as color, opacity, emission, phase function, etc. Typically these functions have many parameters that have to be set by the user by hand or through interactive exploration of the volume data. As the survey by [Kindlmann 2002] on transfer functions and generation methods shows, the process of creating expressive transfer functions can be a very time consuming and frustrating task. For multivariate volumes, this problem becomes even more daunting since the number of parameters grows with the number of dimensions, sometimes exponentially. It is therefore important, especially for multivariate datasets, to have transfer functions with simple expressions that rely on a limited number of free parameters. We have found transfer functions based on the Gaussian primitive to be particularly useful.

3.1 General GTF

The *Gaussian Transfer Function* (*GTF*) is defined in one dimension as:

$$g(v,c,\sigma) = e^{-(v-c)^2/\sigma^2} \tag{1}$$

where v is the sampled data value, c is the data value that the Gaussian is centered over, and σ is the width of the Gaussian. Note that this function does not represent a probability distribution. The GTF is a scaled version of the normal distribution that does not integrate to one, yet retains its shape and simplicity.

While the above definition illustrates the shape and some desirable properties of the GTF in one dimension, we are interested in multidimensional transfer functions. The multivariate Gaussian transfer function is written as:

$$\mathrm{GTF}(\vec{v},\vec{c},\mathbf{K}) = e^{-(\vec{v}-\vec{c})^T\mathbf{K}^T\mathbf{K}(\vec{v}-\vec{c})} \tag{2}$$

where $\vec{v}$ is the sampled data vector of dimension n (the number of values at each sample in the data set), $\vec{c}$ is the vector data value that the Gaussian is centered over, and $\mathbf{K}$ is an $n \times n$ linear transformation matrix that can scale and rotate the Gaussian (see Table 1). For example, if $\mathbf{K}$ is a diagonal matrix, it scales the Gaussian along the primary axes of the data domain. In the more general case, the matrix $\mathbf{K}$ can rotate and scale the Gaussian about the center $\vec{c}$. As defined above, the GTF takes values between 0 and 1. We obtain achromatic opacity α by scaling the GTF with the maximum opacity value α_{max}.

To select several features from the data set and show each one in its own color, we build a transfer function by combining several

$\mathbf{K}$	Linear transformation matrix
$\vec{K}_V$	Vector representing scaling
K_S	Scalar representing uniform scaling
ρ	Density
τ	Extinction
$\vec{c}$	Center of the Gaussian
$\vec{c}'$	Transformed center of the Gaussian
$\vec{v}$	Vector data value
v	Scalar data value
erf	Error function
GTF	Gaussian transfer function
C	Color
α	Opacity
α_{max}	Maximum opacity

Table 1: Notation and important terms used in the paper

Gaussian primitives. We sum the opacities α_i and average the colors C_i together:

$$\alpha = \sum \alpha_i \quad and \quad C = \sum \alpha_i C_i. \tag{3}$$

where α and C are the resulting opacity and color contributions from all primitives. Note that these operators combine the individual contributions without taking into account the order in which they are specified.

Using Gaussian primitives is just one possible approach to building transfer functions. In the past, researchers have explored the use of precomputed lookup tables, piecewise linear or piecewise quadratic functions. While individual elements of these functions are simple and can be evaluated efficiently, the number of elements required to build transfer functions that can faithfully select fine features can grow very large for multivariate datasets. For example, representing an n-dimensional transfer function capable of selecting a neighborhood of size Δ around a data value v may require a lookup table with $1/\Delta^n$ entries. Commonly used datasets can have $1/\Delta = 256$ and $n = 4$, leading to table sizes larger than the available texture memory on current graphics cards. Piecewise linear and piecewise quadratic functions are more memory efficient than lookup tables but can still suffer from an exponential growth in the number of free parameters with respect to the number of dimensions. For rendering, we are interested in the transfer function applied to a line segment between two data values. Even for transfer functions that can be represented with relatively few segments along each of the primary data axes, the restriction of the function's domain to an arbitrary line through data space may be quite complicated (use a large number of segments). Our Gaussian primitives have higher expressive power while still being simple enough to allow intuitive parameter control and efficient hardware evaluation. In addition, multidimensional GTFs restricted to arbitrary lines result in a simple one-dimensional Gaussian that can be analytically integrated. We explore uses of this property in Section 4.

3.2 Triangular GTF

For visualizing boundaries between materials in scalar data, one can benefit from transfer functions that also depend on the data gradient magnitude. We extend 1D GTFs to 2D transfer functions using a construction introduced by [Levoy 1988]. A triangular transfer function primitive can be generated using Gaussians by adjusting their widths depending on the gradient magnitude $\|\nabla v\|$: $\sigma = \sigma'\|\nabla v\|$, where σ' is a free parameter (corresponding to the width of the triangular GTF for $\|\nabla v\| = 1$). The intuition behind this construction is that in regions of the volume with high gradient magnitude, the data values are changing fast, so it is likely that the region will contain the boundary (feature) that we are looking for. Therefore, it is advantageous to apply a wider Gaussian to increase the chance that the region will be selected by the transfer function. The use of the triangular classification function can also be easily extended for use with multi-field datasets by replacing the gradient magnitude from the univariate case with the L_2 norm of the matrix $D^T D$ where the rows of D are the gradients of each of the data fields.

4 Piecewise Analytic Integration

The intersection of the multidimensional GTF with an arbitrary line through data space results in one-dimensional Gaussian. This allows us to integrate the transfer function over line segments in the volume for which the data varies linearly. As shown in Figure 3 for narrow peaked transfer functions, this analytical integration is much more accurate than a numerical (Riemann sum) integration using the same number of samples for each viewing ray.

The *emission-absorption* volume rendering equation over a line segment is defined as [Sabella 1988]:

$$I(a,b) = \int_a^b C\rho(v(u))\, e^{-\int_a^u \tau\rho(v(t))dt} du \tag{4}$$

where τ is extinction (expressing attenuation along the ray), ρ is density, C is radiant intensity or color, $v(t)$ is the data value at the position along the ray parameterized by t starting at the spatial position $\mathbf{x}$ in direction $\vec{\omega}$. If we assume that the color C and extinction τ are constant over the segment, the intensity can be expressed as [Max et al. 1990]:

$$I(a,b) = \frac{C}{\tau}\alpha \tag{5}$$

where the opacity term α is:

$$\alpha = 1 - e^{-\tau \int_a^b \rho(v(t))dt}. \tag{6}$$

If we further assume that data values along the ray between parameters a and b vary linearly, the opacity term becomes:

$$\alpha(v_1, v_2, l) = 1 - e^{-\tau l \int_0^1 \rho(v_1 + t(v_2 - v_1))dt} = 1 - e^{-\tau l \rho'} \tag{7}$$

where $v_1 = v(a)$ is the data value at ray parameter a, $v_2 = v(b)$ is the value at ray parameter b, $l = b - a$, and ρ' is the density line integral along the segment. For arbitrary one dimensional transfer functions the integral can be expressed as [Williams and Max 1992]:

$$\rho'(v_1, v_2) = \int_0^1 \rho(v_1 + t(v_2 - v_1))dt = \frac{R(v_2) - R(v_1)}{v_2 - v_1} \tag{8}$$

where $R(v)$ is the integral function of the density:

$$R(v) = \int_{-\infty}^{v} \rho(x)dx \tag{9}$$

The opacity is computed similarly to (7) when ρ is a multidimensional function:

$$\alpha(\vec{v}_1, \vec{v}_2, l) = 1 - e^{-\tau l \int_0^1 \rho(\vec{v}_1 + t(\vec{v}_2 - \vec{v}_1))dt} = 1 - e^{-\tau l \rho'} \tag{10}$$

In general, the line integral ρ' has no analytic solution. In the companion paper [Kniss et al. 2003], we show that if we let $\rho(\vec{v}) = \mathrm{GTF}(\vec{v}, \vec{c}, \mathbf{K})$, $\rho\prime$ becomes:

$$\rho'(\vec{v}_1, \vec{v}_2) = \frac{\sqrt{\pi}}{2}\frac{S}{\|\vec{d}\|}(\mathrm{erf}(B) - \mathrm{erf}(A)) \tag{11}$$

where

$$A = \frac{\vec{d}\cdot\vec{v}_1'}{\|\vec{d}\|}, \qquad B = A + \|\vec{d}\|, \qquad S = e^{-\|\vec{v}_1'\|^2 + A^2}$$

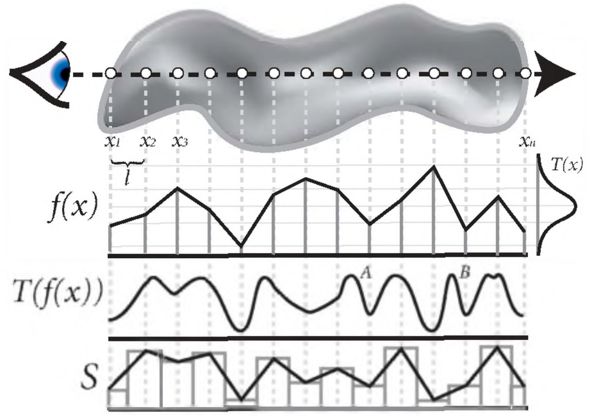

Figure 3: Setup for analytic integration using Gaussian transfer functions. The top image shows a parameterized ray going through a volume. The volume is sampled at points $x_1 \ldots x_n$ along this ray. A continuous function f is reconstructed from these samples using linear interpolation. A Gaussian transfer function T is then applied to the function f, and becomes $T(f(v))$. Traditionally, the integral of $T(f(v))$ is computed using a Riemann sum, seen at the bottom labeled S. Notice how the peaks A and B in $T(f)$ are missing in the Riemann sum. Piecewise analytic integration of $T(f)$ ensures that we do not miss these peaks.

$$\vec{v}_1' = \mathbf{K}(\vec{v}_1 - \vec{c}), \quad \vec{v}_2' = \mathbf{K}(\vec{v}_2 - \vec{c}), \quad \vec{d} = \vec{v}_2' - \vec{v}_1' \tag{12}$$

and erf(z) is the *error function*:

$$\mathrm{erf}(z) = \frac{2}{\sqrt{\pi}}\int_0^z e^{-x^2}dx. \tag{13}$$

Notice that the $\sqrt{\pi}/2$ in equation (11) cancels the $2/\sqrt{\pi}$ in equation (13). While erf has no explicit representation, it can be closely approximated with simple functions. We found the approximation of [Abramowitz and Stegun 1974] particularly useful and easy to implement.

Note that if $\vec{v}_1 = \vec{v}_2$, *i.e.*, when we have two samples in a homogeneous region, $\|\vec{d}\| = 0$ and we cannot use equation (11) directly. In this case the formula converges to:

$$\rho'(\vec{v}_1, \vec{v}_2) \rightarrow \rho(\vec{v}_1) \tag{14}$$

as $\|\vec{d}\| \rightarrow 0$, since ρ' becomes the derivative of the integral function, which is the integrand itself.

We use the following formulae to combine transfer function elements during piecewise analytic integration of each segment:

$$\begin{aligned} \rho_i' &= \int_0^1 \rho_i(v_1 + t(v_2 - v_1))dt \qquad \rho_i(v) = \mathrm{GTF}(v, c_i, \mathbf{K}_i) \\ \alpha &= 1 - e^{-l\sum\tau_i\rho_i'} \\ C &= \frac{\sum\tau_i\rho_i' C_i'}{\sum\tau_i\rho_i'} = \frac{\sum\rho_i' C_i}{\sum\tau_i\rho_i'} \qquad C_i' = \frac{C_i}{\tau_i} \end{aligned} \tag{15}$$

where the integrals in the sum for computing the opacity α are evaluated separately similarly to equation (11). Note that even though the sum of GTFs is not a GTF, we can still integrate them separately, scale them by τ_i and sum them in the exponent. Combining the color contributions employs a commonly used approximation that neglects the order in which the primitives appear along the line segment [Engel et al. 2001]. We also have to divide the input color C_i by the input extinction coefficient τ_i according to equation (5).

Note that in theory the extinction coefficient τ_i takes values between 0 and ∞. In practice the necessary upper limit is much lower because τ_i is integrated resulting in an opacity value that quickly reaches one.

5 Implementation

In this section, we describe the practical implementation details of Gaussian transfer functions from Section 3 and the implementation of the analytical integration described in Section 4.

5.1 Gaussian Transfer Function

The goal of Gaussian transfer functions is to provide a general and scalable class of transfer function primitives for specifying fully general multidimensional transfer functions. Rather than utilizing a lookup table to evaluate the transfer function for data values sampled within the volume, the Gaussian transfer function is evaluated as a true function for each sample. The down side of transfer functions that are evaluated explicitly is that the computational cost of evaluation is linearly proportional to the number of transfer function primitives used. Although it may seem that this computational cost would preclude the use of this class of transfer functions for interactive volume rendering, we have found them quite practical for a number of reasons. First, the simple and continuous form of the Gaussian transfer function makes it an efficient function to compute on modern graphics hardware. Second, we found that in practical applications we rarely use more than four or five transfer function primitives at a time.

The fragment processing pipeline on modern GPUs provides a rich set of SIMD vector operations such as component-wise arithmetic, vector dot products, exponentiation and trigonometric functions. The current generation of graphics hardware supports all of the necessary instructions to implement explicit evaluation of transfer functions at full 32-bit floating point precision. High precision is important since we would like the transfer function primitives to be general with respect to both dimension and dynamic range.

The GTF can be evaluated on modern graphics hardware in as few as four instructions; a multiply-add instruction, a dot product, an exponential, and a multiply. This holds for datasets with up to 4 fields, and each additional multiple of 4 fields adds 3 instructions; a multiply-add, a dot product, and an add. The algorithm for evaluating the GTF is shown below in pseudo code:

1	$\vec{r} = \vec{K}_{V,i} * \vec{v} - \vec{c}_i'$	Vector Multiply-Add
2	$r = \vec{r} \cdot \vec{r}$	Vector Dot Product
3	$r = exp(-r)$	Scalar Exponent
4	$\alpha_i = \alpha_{max,i} * r$	Scalar Multiply

Table 2: Fragment program for computing opacity using the GTF.

The GTF parameters $\vec{c}' = \vec{K}_V * \vec{c}$, $\vec{K}_V$, and h are stored as fragment program constants, while the sampled data value vector $\vec{v}$ can be read from a data texture and/or come from other variables such as a spatial position or the dot product of the view direction and normal. r is a temporary register. The program in Table 2 assumes that the matrix **K** only scales the GTF along the primary axes of the transfer function domain, so the diagonal matrix **K** can be represented with just a vector $\vec{K}_V$ with n elements, where n is the number of fields in the dataset. A general matrix representation for **K** would be more expressive allowing us to specify an arbitrary orientation for the GTF. However, it would significantly complicate any user interface for the transfer function, and the additional computational cost of evaluating the matrix-vector multiply may outweigh the benefits. An example of classification using a 3D GTF is seen in Figure 4.

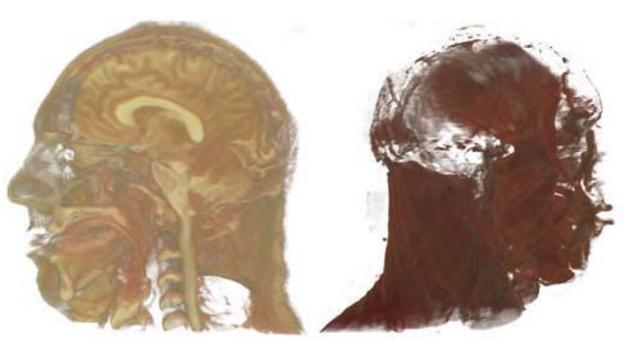
Figure 4: Examples of multi-field volume classified using a GTF. The dataset is the Visible Male Color Cryosection, courtesy of the National Institutes of Health. Dataset size is 256^3

1	$r_0 = 1/g$	Scalar Reciprocal
2	$\vec{r}_1 = \vec{K}_{V,i} * \vec{v} - \vec{c}_i'$	Vector Multiply-Add
3	$\vec{r}_1 = r_0 * \vec{r}_1$	Scalar Vector Multiply
4	$r_1 = \vec{r}_1 \cdot \vec{r}_1$	Vector Dot Product
5	$r_1 = exp(-r_1)$	Scalar Exponent
6	$\alpha_i = \alpha_{max,i} * r_1$	Scalar Multiply
7	$\alpha_i = \text{if}(g = 0)\{0\}\text{else}\{\alpha_i\}$	Scalar Conditional

where $g = \|\nabla v_i\|$ is the gradient magnitude of one of the data values, and r_0 and r_1 are temporary registers. $\vec{K}_V$ is the scaling vector that scales the Gaussians along the axes of the transfer function domain.

Table 3: Fragment program for computing opacity using the triangular GTF.

The triangular GTF described in Section 3.2 requires three additional instructions (a scalar reciprocal, a vector multiply, and a scalar conditional-move operation) compared to the general GTF (see Table 3). The conditional in line 7 of Table 3 is required because we have a divide by zero when $g = 0$. Note that we do not always have to check for division by zero if the graphics hardware architecture implements the IEEE floating point standard. If g is equal to zero in the first instruction, the result will be properly carried through subsequent instructions as expected, setting pixels with invalid values to zero. An example of classification using a 4D triangular GTF is seen in Figure 5.

Notice that the algorithm for computing the GTF involves two vector operations and two scalar operations. Similarly, the triangular GTF requires three vector and four scalar operations. This symmetry is important since modern programmable GPUs allow us to compute one vector and one scalar operation in parallel. We cannot exploit this parallelism in the computation of a single GTF because each operation is serially dependent on the previous one. However, the computation of multiple GTF primitives can be interleaved such that the vector operations for one are computed in parallel with the scalar operations for another. Therefore, we can effectively compute two transfer function primitives at once. Combining multiple GTFs requires us to keep track of the summed colors and GTFs, as in equation (3), adding an additional vector/scalar pair of instructions per GTF:

1	$C = C_i * \alpha_i + C$	Vector Multiply-Add
2	$\alpha = \alpha_i + \alpha$	Scalar Add

where C is the cumulative opacity weighted color, α is the cumulative opacity, α_i is computed using the program from Table 2 or Table 3 and C_i is the color for the current primitive.

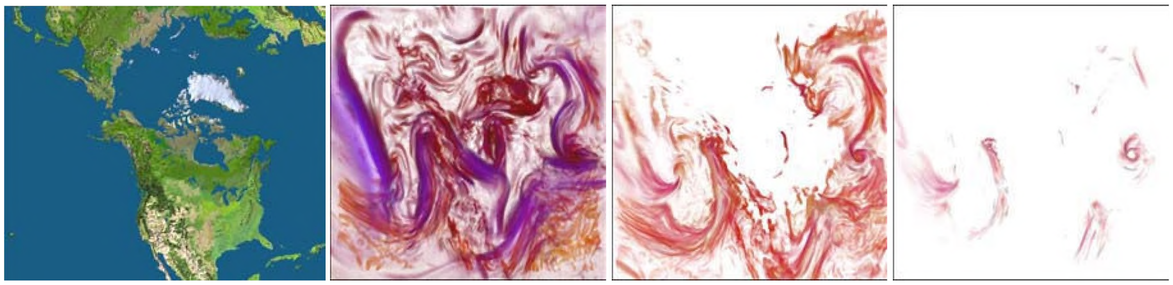

Figure 5: An example of classification using the triangular GTF. The data set is a numerical weather simulation courtesy of the Canadian Meteorological Centre and includes 4 fields, temperature, humidity, wind speed, and a derived multi-gradient. The left image identifies the simulation domain. The left-center image shows a *default transfer function* created by centering a triangular GTF at the median value for each axis and setting its width to one. The right-center image shows some of the airmass boundaries (fronts). The right image was created by modifying the triangular GTF's width along the wind speed axis to select only those portions of the previously classified airmass that have wind speeds greater than fifty percent of the maximum wind speed in the simulation. Dataset size is $256x256x64$.

5.2 Piecewise Analytic Integration

5.2.1 One Dimensional Case

The use of explicitly evaluated transfer functions as described in the previous section, assumes that the volume rendering equation is being solved by compositing color and opacity segments along the viewing ray using a Riemann sum. It is well known that this technique produces significant artifacts if the sampling rate is not high enough. We use analytic integration based on the equations derived in Section 4 in order to significantly reduce the number of samples required to reconstruct the data with good fidelity.

The analytic integral for scalar data and gradient magnitude can be implemented as a special case of equation (11). In this case we can simplify the general multidimensional case to 1D, because the gradient magnitude only modifies the width of a 1D GTF.

We use equation (8) to compute the density integral using the GTF. For the triangular version, ρ' will depend on the gradient magnitudes and we use the same formulation with the following input parameters:

$$v_j' = \frac{K(v_j - c)}{\hat{g}}, \quad \hat{g} = \frac{g_1 + g_2}{2}, \quad g_j = \|\nabla v_j\|.$$

We found that this approximation of the density integral works well in practice. Further justification for using the average of the gradient magnitudes can be found in [Kniss et al. 2003].

For this special case we precompute a 2D function:

$$\begin{aligned} \mathrm{IGauss}(x_1, x_2) &= \frac{\sqrt{\pi}}{2}\frac{\mathrm{erf}(x_1) - \mathrm{erf}(x_2)}{x_1 - x_2} & x_1 &\neq x_2 \\ \mathrm{IGauss}(x_1, x_2) &= e^{-x_1^2} & x_1 &= x_2. \end{aligned} \tag{16}$$

In our implementation we evaluate this function within a domain from $[-10, 10]$ in both x_1 and x_2, and store it as a 2D texture. Since the function is smooth we have found 128^2 16-bit samples to be sufficient. A scale and bias are required to access the texture correctly, since its texture coordinates are $[0, 1]$. The analytic piecewise integral of the triangular GTF for scalar data can be implemented with only four more instructions than the triangular GTF itself:

-	$c_i = \{C_i.x, C_i.y, C_i.z, \tau_i\}$	Color and extinction input
1	$\vec{r}.x = \vec{K}_{V,i} * v_1 - c_i'$	Scalar Multiply-Add
2	$\vec{r}.y = \vec{K}_{V,i} * v_2 - c_i'$	Scalar Multiply-Add
2(a)	$\vec{r} = \vec{r} * (1/\hat{g})$	Scalar Multiply
3	$\vec{r} = \vec{r}_0 * .05 + .5$	Vector Multiply-Add
4	$\vec{r}.x = \mathrm{IGauss}(\vec{r}.xy)$	2D Texture Read
5	$c = c + \vec{r}.x * c_i$	Vector Multiply-Add

where $c = (C, \tau)$ contains the combined color and extinction terms, and $c_i = (C_i, \tau_i)$. Step 2(a) is used for the triangular GTF only.

Once all primitive's color and extinction quantities have been computed, a final step is required to compute the opacity and the correctly weighted color:

-	$r = \{-l, 0, 0, 0\}$	Length input
1	$r.w = 1/c.w$	Scalar Reciprocal
2	$c = c * r.wwwx$	Scalar Multiply
3	$c.w = exp(c.w)$	Scalar Exponential
4	$c.w = 1 - c.w$	Scalar Add
5	$c.xyz = c.xyz * c.w$	Scalar Multiply

where r is a temporary register.

This algorithm leverages the fact that most modern graphics hardware architectures are capable of executing a texture read and a vector/scalar pair of operations simultaneously. For instance, the Nvidia FX series can handle two texture reads and two operations simultaneously. This algorithm also permits interleaving of instructions for the evaluation of multiple GTFs.

5.2.2 Multidimensional Case

The analytic integral of the general multidimensional GTF for linearly varying data, defined in equation (11), can be implemented entirely in a fragment program including an approximation of the erf function. We have however found that the large number of instructions required (over 30) affects performance dramatically. About half of these instructions are devoted to computing the two erf functions. Similarly to the one-dimensional case, we can precompute $\mathrm{erf}(x) - \mathrm{erf}(y)$ and store it into a 2D texture. Since this function asymptotically approaches constant values as the absolute value of the argument grows, we only need to represent a small interval around the origin and clamp to the edges of the texture when accessing values outside this interval. We have found that the domain $-3.6 \leq x, y \leq 3.6$, is adequate for a 16-bit lookup table.

Table 4 shows the fragment program used in the multidimensional case. Naturally, 20 fragment instructions are a lot for a rendering technique that is already fill bound. This computation takes

1	$\vec{v}_1 = \vec{K}_V * \vec{v}_1 - \vec{c}'$	Vector Multiply-Add
2	$\vec{v}_2 = \vec{K}_V * \vec{v}_2 - \vec{c}'$	Vector Multiply-Add
3	$\vec{d} = \vec{v}_2 - \vec{v}_1$	Vector Subtract
4	$r_0 = \vec{d} \cdot \vec{d}$	Vector Dot Product
5	$r_1 = 1/\sqrt{r_0}$	Scalar Recip. Sqrt.
6	$A\vec{B}G.x = \vec{d} \cdot \vec{v}_1$	Vector Dot Product
7	$A\vec{B}G.x = A\vec{B}G.x * r_1$	Scalar Multiply
8	$A\vec{B}G.y = r_0 * r_1 + A\vec{B}G.x$	Scalar Multiply-Add
9	$A\vec{B}G.z = \vec{v}_1 \cdot \vec{v}_1$	Vector Dot Product
10	$A\vec{B}G.w = A\vec{B}G.z - A\vec{B}G.x$	Scalar Move
11	$A\vec{B}G.z = exp(-A\vec{B}G.z)$	Scalar exponentiation
12	$A\vec{B}G.w = exp(-A\vec{B}G.w)$	Scalar exponentiation
13	$A\vec{B}G.xy = A\vec{B}G.xy * 1/7.2 + .5$	Vector Multiply-Add
14	$E = \mathrm{erf}(A\vec{B}G.xy)$	2D Texture Read
15	$I = hl * r_1$	Scalar Multiply
16	$I = E * I$	Scalar Multiply
17	$I = I * A\vec{B}G.w$	Scalar Multiply
18	$I = I * r_1$	Scalar Multiply
19	$LH = A\vec{B}G.z * hl'$	Scalar Multiply
20	$I = \mathrm{if}(r_0 = 0)\,\{LH\}\,\mathrm{else}\,\{I\}$	Scalar Conditional

Table 4: Fragment program for analytic integration in the multidimensional case.

more than twice as long as the point sampled multidimensional GTF, meaning that this analytic integral is as computationally expensive as the Riemann sum of a GTF with at least 2 extra samples over the same domain. In general, this analytic solution would only be required when one of the components of **K** is very large, *i.e.* the GTF is very narrow along one of the dimensions of the dataset, or the gradient magnitude is being used to emphasize boundaries.

6 Results

As can be seen in Figure 1, one advantage of the analytic triangular GTF is the ability to extract thin material boundaries with fewer slices than using Riemann sum integration methods. On the left is the CT tooth dataset extracting the enamel, dentin, and pulp boundaries rendered with the analytic triangular GTF using 128 slices. In the middle is an image with a non-analytic triangular GTF using 368 slices. On the right is an image rendered with a non-analytic triangular GTF but with only 128 slices. This example demonstrates the ability to extract thin material boundaries with similar quality using far fewer slices by employing the analytic GTF.

In evaluating performance, the computational cost of explicitly evaluating transfer functions must be considered. Each classified feature adds a new transfer function primitive, which increases the time for the evaluation. Table 5 lists the timings achieved for different numbers of transfer function primitives using the approaches presented in this paper, and compares them to the traditional approaches using table lookups. Separable 2D Textures use one pair of 2D textures for each transfer function primitive. This ensures we do not have classification errors detailed in Section 1. It is clear from the table that the cost of explicitly computing the analytic solution is higher, due to the long fragment program, than either separable transfer function or just the GTF.

Figure 6 shows another example of the tradeoff between quality and speed for the analytic triangular GTF and the Riemann sum integration methods. The data is a multi-modal MRI of a mouse brain, and the transfer function uses three primitives representing the cortex, white matter, and the ventricles. Figure 6 (A) shows the results of using the 4D triangular GTF (TGTF) rendered by Riemann sum integration with 270 slices. Figure 6 (B) is the same dataset and

Type	Number of Primitives			
	1	2	3	4
Separable 2D Texture	.20	.26	.39	.54
3D GTF (RGB only)	.21	.30	.39	.55
4D TGTF (Triangle)	.24	.33	44	.59
Analytic 3D GTF	.55	1.11	1.79	2.45
Analytic 4D TGTF	.57	1.23	2.00	2.60

Table 5: Time comparisons in seconds per-frame using the Duke Mouse Brain at 1.2 average samples per voxel on an Nvidia GeForceFX 5900. GTF computations were not interleaved in these timings.

number of slices but rendered with the analytic 4D TGTF. For figure 6 (C), the Riemann sum was again used but the number of slices was increased (1000 slices) to match the rendering speed required by figure 6 (B). Note the higher quality result of the analytic integration.

If speed is the primary concern in a volume rendering system, it is likely that a separable transfer function using table lookups will suffice. However, if accuracy and scalability are primary concerns, Gaussian transfer functions may be preferable over separable lookup tables. Although a general n-dimensional transfer function cannot be decomposed into n 1D transfer functions without errors in the classification, it is possible to decompose it if it consists of m separable transfer function primitives. In this case the transfer function could be represented by $n * m$ 1D transfer functions. Naturally, the computational cost of this decomposition could be much higher compared to an explicitly evaluated transfer function with the same characteristics. The cost for evaluation of the analytic GTF is high but is clearly effective for thin material boundaries.

User interface concerns are very important in multi-field transfer function design. Although direct manipulation of classified features is often necessary, we have found that a simple modification of dual-domain interaction [Kniss et al. 2002b] can provide the primary transfer function interface without an explicit representation of the transfer function domain. Our system allows the user to probe a simple color-mapped slice of the multi-field data. As the user probes a feature, all sample locations are recorded and basic statistics are performed on these samples to derive the mean, standard deviation, and covariance required for GTF specification. The user is then able to refine the GTF specification by modifying simplified and independent parameters like size, opacity, color, etc. Each classification element can be manipulated independently, or grouped with other elements representing the same feature and modified simultaneously. An explicit representation of the transfer function is required for fine tuning. Each classification element or group is modified independently of all others. The interface provides a number of $2D$ projections of the multi-dimensional transfer function space that allow the user to modify the position and size of GTFs along each axis simultaneously or independently.

7 Conclusions

We have presented a general and scalable framework for classifying and rendering multi-field datasets, using transfer functions based on Gaussian primitives. We have adopted these transfer functions because they are well suited for classifying narrow features in multidimensional domains. The triangular Gaussian transfer function is a simple modification of the basic GTF that makes it useful for classification based on gradient magnitude measures.

These transfer functions are efficient to implement, especially on the modern programmable graphics hardware and do not require the large amount of texture memory storage as do high dimensional lookup tables. With reasonable assumptions, *e.g.* data values only vary linearly between two sample points, they can be analytically

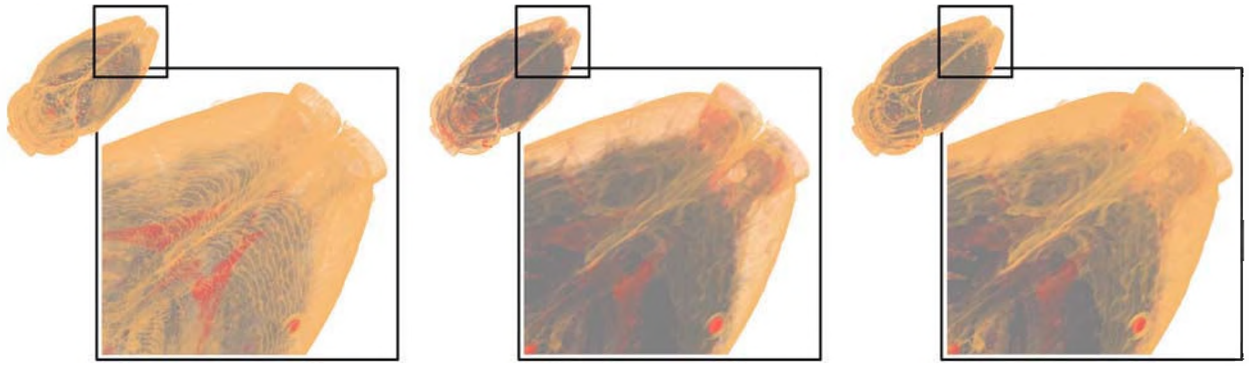
(A) 0.43 s/frame (B) 2.11 s/frame (C) 2.20 s/frame

Figure 6: A multi-field MR of a mouse brain, courtesy of the Duke Center for In Vivo Microscopy, consisting of four fields; PD, T2, Diffusion Tensor trace, and a derived multi-gradient. The images are rendered with the Riemann sum and analytically integrated TGTF (3 primitives: cortex (tan), white matter (gray), and ventricles (red)). The dataset size is 256^3.

integrated over a line segment even in multiple dimensions. Although functions would work as an underlying transfer function primitive, the GTF is valuable because it is easy to control, very simple to implement, and analytically integrable.

8 Acknowledgments

We would like to thank Will Nesse, Dmitriy Pavlov, and Frank Stenger for stimulating conversations about functional approximation, statistics, and analytic integration techniques, which aided greatly in the development of this work. Dave Weinstein and Helen Hu read drafts of the paper. This work was funded by the following grants: NSF ACR-9978032, NSF MRI-9977218, NSF ACR-9978099 and the US Department of Energy's VIEWS program.

References

ABRAMOWITZ, M., AND STEGUN, I. A. 1974. *Handbook of Mathematical Functions*. Dover, June.

BLINN, J. F. 1982. Light reflection functions for simulation of clouds and dusty surfaces. *Computer Graphics (SIGGRAPH '82 Proceedings) 16*, 3 (July), 21–29.

CABRAL, B., CAM, N., AND FORAN, J. 1994. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *1994 Symposium on Volume Visualization*, 91–98.

DREBIN, R. A., CARPENTER, L., AND HANRAHAN, P. 1988. Volume rendering. In *Computer Graphics (SIGGRAPH '88 Proceedings)*, J. Dill, Ed., vol. 22, 65–74.

ENGEL, K., KRAUS, M., AND ERTL, T. 2001. High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading. In *Eurographics / SIGGRAPH Workshop on Graphics Hardware '01*, 9–16.

KAJIYA, J. T., AND VON HERZEN, B. P. 1984. Ray tracing volume densities. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, vol. 18, 165–174.

KINDLMANN, G. 1999. *Semi-Automatic Generation of Transfer Functions for Direct Volume Rendering*. Master's thesis, Program of Computer Graphics, Cornell University.

KINDLMANN, G., 2002. Transfer functions in direct volume rendering: Design, interface, interaction. Siggraph Course Notes.

KNISS, J., HANSEN, C., GRENIER, M., AND ROBINSON, T. 2002. Volume rendering multivariate data to visualize meteorological simulations: A case study. In *Eurographics - IEEE TCVG Symposium on Visualization 2002*.

KNISS, J., KINDLMANN, G., AND HANSEN, C. 2002. Multi-dimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics 8*, 3, 270–285.

KNISS, J., PREMOŽE, S., IKITS, M., LEFOHN, A., AND HANSEN, C. 2003. Closed-form approximations to the volume rendering integral with Gaussian transfer functions. Tech. Rep. UUCS-03-013, School of Computing University of Utah.

LAIDLAW, D. H. 1995. *Geometric Model Extraction from Magnetic Resonance Volume Data*. PhD thesis, Department of Computer Science, California Institute of Technology.

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

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

MAX, N. 1995. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics 1*, 2 (June), 99–108.

MEISSNER, M., HOFFMANN, U., AND STRASSER, W. 1999. Enabling Classification and Shading for 3D Texture Mapping based Volume Rendering using OpenGL and Extensions. In *IEEE Visualization '99 Proc.*

PFISTER, H., LORENSEN, B., BAJAJ, C., KINDLMANN, G., SCHROEDER, W., AVILA, L. S., MARTIN, K., MACHIRAJU, R., AND LEE, J. 2001. The transfer function bake-off. *IEEE Computer graphics and Applications 21*, 3 (May/June), 16–22.

PORTER, T., AND DUFF, T. 1984. Compositing digital images. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, vol. 18, 253–259.

REZK-SALAMA, C., ENGEL, K., BAUER, M., GREINER, G., AND ERTL, T. 2000. Interactive Volume Rendering on Standard PC Graphics Hardware Using Multi-Textures and Multi-Stage-Rasterization. In *Eurographics / SIGGRAPH Workshop on Graphics Hardware '00*, 109–118,147.

ROETTGER, S., AND ERTL, T. 2002. A Two-Step Approach for Interactive Pre-Integrated Volume Rendering of Unstructured Grids. In *Proc. IEEE VolVis '02*.

ROETTGER, S., KRAUS, M., AND ERTL, T. 2000. Hardware-accelerated vollume and isosurface rendering based on cell-projection. In *Preceedings of IEEE Visualization*, 109–116.

SABELLA, P. 1988. A rendering algorithm for visualizing 3D scalar fields. In *Computer Graphics (SIGGRAPH '88 Proceedings)*, vol. 22, 51–58.

SHIRLEY, P., AND TUCHMAN, A. 1990. A polygonal approximation to direct scalar volume rendering. In *Computer Graphics (San Diego Workshop on Volume Visualization)*, vol. 24, 63–70.

STEIN, C., BECKER, B., AND MAX, N. 1994. Sorting and hardware assisted rendering for volume visualization. In *1994 Symposium on Volume Visualization*, 83–90.

WESTERMANN, R., AND ERTL, T. 1998. Efficiently using graphics hardware in volume rendering applications. In *SIGGRAPH 98 Conference Proceedings*, M. Cohen, Ed., 169–178.

WESTERMANN, R., AND SEVENICH, B. 2001. Accelerated volume ray-casting using texture mapping. In *Proceedings of IEEE Visualization*, 271–278.

WILLIAMS, P. L., AND MAX, N. 1992. A volume density optical model. *1992 Workshop on Volume Visualization*, 61–68.

WILLIAMS, P. L., MAX, N. L., AND STEIN, C. M. 1998. A high accuracy volume renderer for unstructured data. *IEEE Transactions on Visualization and Computer Graphics 4*, 1 (Jan. – Mar.).

WITTENBRINK, C. M., MALZBENDER, T., AND GOSS, M. E. 1998. Opacity-weighted color interpolation for volume sampling. In *IEEE Symposium on Volume Visualization*, 135–142.