

Interactive Volume Rendering Using Multi-Dimensional Transfer Functions and Direct Manipulation Widgets

Joe Kniss

Gordon Kindlmann

Charles Hansen

Scientific Computing and Imaging Institute
School of Computing, University of Utah
{jmk|gk|hansen}@cs.utah.edu

1 Abstract

Most direct volume renderings produced today employ one-dimensional transfer functions, which assign color and opacity to the volume based solely on the single scalar quantity which comprises the dataset. Though they have not received widespread attention, multi-dimensional transfer functions are a very effective way to extract specific material boundaries and convey subtle surface properties. However, identifying good transfer functions is difficult enough in one dimension, let alone two or three dimensions. This paper demonstrates an important class of three-dimensional transfer functions for scalar data (based on data value, gradient magnitude, and a second directional derivative), and describes a set of direct manipulation widgets which make specifying such transfer functions intuitive and convenient. We also describe how to use modern graphics hardware to interactively render with multi-dimensional transfer functions. The transfer functions, widgets, and hardware combine to form a powerful system for interactive volume exploration.

CR Categories: I.3.3 [Computer Graphics]— Picture/Image Generation, Computational Geometry and Object Modeling, Methodology and Techniques, Three-Dimensional Graphics and Realism

Keywords: volume visualization, direct volume rendering, multi-dimensional transfer functions, direct manipulation widgets, graphics hardware

2 Introduction

Direct volume rendering has proven to be an effective and flexible visualization method for three-dimensional (3D) scalar fields. Transfer functions are fundamental to direct volume rendering because their role is essentially to make the data visible: by assigning optical properties like color and opacity to the voxel data, the volume can be rendered with traditional computer graphics methods. Good transfer functions reveal the important structures in the data without obscuring them with unimportant regions. To date, transfer functions have generally been limited to one-dimensional (1D) domains, meaning that the 1D space of scalar data value has been the only variable to which opacity and color are assigned. One aspect of direct volume rendering which has received little attention is the use of multi-dimensional transfer functions.

Often, there are features of interest in volume data that are difficult to extract and visualize with 1D transfer functions. Many medical datasets created from CT or MRI scans contain a complex combination of boundaries between multiple materials. This situation is problematic for 1D transfer functions because of the potential for overlap between the data value intervals spanned by the different boundaries. When one data value is associated with mul-

tle boundaries, a 1D transfer function is unable to render them in isolation. Another benefit of higher dimensional transfer functions is their ability to portray subtle variations in properties of a single boundary, such as its thickness.

Unfortunately, using multi-dimensional transfer functions in volume rendering is complicated. Even when the transfer function is only 1D, finding an appropriate transfer function is generally accomplished by trial and error. This is one of the main challenges in making direct volume rendering an effective visualization tool. Adding dimensions to the transfer function domain only compounds the problem. While this is an ongoing research area, many of the proposed methods for transfer function generation and manipulation are not easily extended to higher dimensional transfer functions. In addition, fast volume rendering algorithms that assume the transfer function can be implemented as a linear lookup table (LUT) can be difficult to adapt to multi-dimensional transfer functions due to the linear interpolation imposed on such LUTs.

While this paper aims to demonstrate the importance and power of multi-dimensional transfer functions, our main contributions are two techniques which make volume rendering with multi-dimensional transfer functions more efficient. To resolve the potential complexities in a user interface for multi-dimensional transfer functions, we introduce a set of direct manipulation widgets which make finding and experimenting with transfer functions an intuitive, efficient, and informative process. In order to make this process genuinely interactive, we exploit the fast rendering capabilities of modern graphics hardware, especially 3D texture memory and pixel texturing operations. Together, the widgets and the hardware form the basis for new interaction modes which can guide users towards transfer function settings appropriate for their visualization and data exploration interests.

3 Previous Work

3.1 Transfer Functions

Even though volume rendering as a visualization tool is more than ten years old, only recently has research focused on making the space of transfer functions easier to explore. He *et al.* [8] generated transfer functions with genetic algorithms driven either by user selection of thumbnail renderings, or some objective image fitness function. The Design Gallery [19] creates an intuitive interface to the entire space of all possible transfer functions based on automated analysis and layout of rendered images. A more data-centric approach is the Contour Spectrum [1], which visually summarizes the space of isosurfaces in terms of metrics like surface area and mean gradient magnitude, thereby guiding the choice of iso-value for isosurfacing, and also providing information useful for transfer function generation. Another recent paper [15] presents a novel transfer function interface in which small thumbnail renderings are arranged according to their relationship with the spaces of data values, color, and opacity.

The application of these methods is limited to the generation of 1D transfer functions, even though 2D transfer functions were

introduced by Levoy in 1988 [18]. Levoy introduced two styles of transfer functions, both two-dimensional, and both using gradient magnitude for the second dimension. One transfer function was intended for the display of interfaces between materials, the other for the display of isovalue contours in more smoothly varying data. The previous work most directly related to this paper facilitates the semi-automatic generation of both 1D and 2D transfer functions [13, 26]. Using principles of computer vision edge detection, the semi-automatic method strives to isolate those portions of the transfer function domain which most reliably correlate with the middle of material interface boundaries.

Other scalar volume rendering research that uses multi-dimensional transfer functions is relatively scarce. One paper discusses the use of transfer functions similar to Levoy's as part of visualization in the context of wavelet volume representation [24]. More recently, the VolumePro graphics board uses a 12-bit 1D lookup table for the transfer function, but also allows opacity modulation by gradient magnitude, effectively implementing a separable 2D transfer function [25]. Other work involving multi-dimensional transfer functions uses various types of second derivatives in order to distinguish features in the volume according to their shape and curvature characteristics [11, 30].

Designing colormaps for displaying non-volumetric data is a task similar to finding transfer functions. Previous work has developed strategies and guidelines for colormap creation, based on visualization goals, types of data, perceptual considerations, and user studies [2, 29, 32].

3.2 Direct Manipulation Widgets

Direct manipulation widgets are geometric objects rendered with a visualization and are designed to provide the user with a 3D interface [4, 10, 28, 31, 34]. For example, a frame widget can be used to select a 2D plane within a volume. Widgets are typically rendered from basic geometric primitives such as spheres, cylinders, and cones. Widget construction is often guided by a constraint system which binds elements of a widget to one another. Each sub-part of a widget represents some functionality of the widget or a parameter to which the user has access.

3.3 Hardware Volume Rendering

Many volume rendering techniques based on graphics hardware utilize texture memory to store a 3D dataset. The dataset is then sampled, classified, rendered to proxy geometry, and composited. Classification typically occurs in hardware as a 1D table lookup.

2D texture-based techniques slice along the major axes of the data and take advantage of hardware bilinear interpolation within the slice [3]. These methods require three copies of the volume to reside in texture memory, one per axis, and they often suffer from artifacts caused by under-sampling along the slice axis. Trilinear interpolation can be attained using 2D textures with specialized hardware extensions available on some commodity graphics cards [5]. This technique allows intermediate slices along the slice axis to be computed in hardware. These hardware extensions also permit diffuse shaded volumes to be rendered at interactive frame rates.

3D texture-based techniques typically sample view-aligned slices through the volume, leveraging hardware trilinear interpolation [7]. Other proxy geometry, such as spherical shells, may be used with 3D texture methods to eliminate artifacts caused by perspective projection [17]. The pixel texture OpenGL extension has been used with 3D texture techniques to encode both data value and a diffuse illumination parameter which allows shading and classification to occur in the same look-up [22]. Engel *et al.* showed how to significantly reduce the number of slices needed to adequately sample a scalar volume, while maintaining a high quality

rendering, using a mathematical technique of pre-integration and hardware extensions such as dependent textures [6].

Another form of volume rendering graphics hardware is the Cube-4 architecture [27] and the subsequent VolumePro PCI graphics board [25]. The VolumePro graphics board implements ray casting combined with the shear warp factorization for volume rendering [16]. It features trilinear interpolation with supersampling, gradient estimation, shaded volumes, and provides interactive frame rates for scalar volumes with sizes up to 256^3 .

4 Multi-Dimensional Transfer Functions

The role of the transfer function in volume rendering is to map the voxel information to renderable properties of opacity and color. Since generating a volume rendering which clearly visualizes the features of interest is only possible with a good transfer function, transfer function specification is a crucial task. Unfortunately, it is difficult to accomplish. We have identified three reasons for this. First, the transfer function has an enormous number of degrees of freedom in which the user can get lost. Even using simple linear ramps, every control point adds two degrees of freedom. Second, the usual interfaces for setting transfer functions (based on moving control points defining a set of linear ramps) are not constrained or guided by the dataset in question.¹ The lack of guidance is what forces the user into a trial-and-error mode of interaction, in which the transfer function domain is explored only by observing changes in the volume rendering as a result of incremental adjustments. Third, transfer functions are inherently non-spatial, in the sense that their assignment of color and opacity does not include spatial position as a variable in their domain. This can be frustrating if the user is interested in isolating one feature of the volume which is *spatially* localized, but not distinguishable, in terms of data value, from the other regions.

Multi-dimensional transfer functions are interesting because they address the third problem, while greatly compounding the first and second problems. Transfer functions can better discriminate between various structures in the volume data when they have more variables—a larger vocabulary—with which to express the differences between them. These variables are the axes of the multi-dimensional transfer function. However, adding dimensions to the transfer function greatly exacerbates the already troublesome problems of unbounded degrees of freedom and lack of user guidance; these challenges are addressed in the next section. Below, we explain our choice of axes for multi-dimensional transfer functions by describing how they enhance the ability to visualize an important class of volume datasets: those in which the features of interest are the boundaries between regions of homogeneous value. Of course, other application areas and visualization goals may imply a different set of relevant data variables for the transfer function axes.

For scalar volume datasets, the gradient is a first derivative. As a vector, it gives the direction of fastest change [21], which motivates its use as the “surface normal” in shaded volume rendering. The gradient magnitude is another fundamental local property of a scalar field, since it characterizes how fast values are changing. Our belief is this: any volume rendering application (medical, industrial, meteorological, etc.) which benefits from using gradient direction for shading can benefit from using gradient magnitude in the transfer function. This does not assume any particular mathematical model of how physical quantities are measured or represented in the volume data; it assumes only that regions of change tend to be regions of interest. Using gradient magnitude as the second dimension in our transfer functions allows structure to be differentiated with varying opacity or color, according to the magnitude of change. For example, the GE Turbine Blade is a canonical

¹The Contour Spectrum is an exception in that it provides information about scalar value and derivative to assist in setting 1D transfer functions.

volume dataset which has a simple two-material composition (air and metal), easily rendered with an isosurface or 1D transfer function. A subtlety in the dataset is how air cavities within the blade have slightly higher values than air outside (perhaps because of tomography artifacts), leading to lesser edge strength for the internal air-metal boundaries. Thus, as seen in Figure 1, this dataset benefits from 2D transfer functions, which can selectively render the internal structures by avoiding opacity assignment at regions of high gradient magnitude.

Our choice for the third axis of the transfer function is more directly based on principles of edge detection, and is best suited for application areas concerned with the boundaries or interfaces between relatively homogeneous materials. Some edge detection algorithms (such as Marr-Hildreth [20]) locate the middle of an edge by detecting a zero-crossing in a second derivative measure, such as the Laplacian. In practice, we compute a more accurate but computationally expensive measure, the second directional derivative along the gradient direction, which involves the Hessian, a matrix of second partial derivatives. Details on these measurements can be found in previous work on semi-automatic transfer function generation [12, 13]. The usefulness of having a second derivative measure in the transfer function is that it enables more precise disambiguation of complex boundary configurations, such as in the human tooth CT scan, shown in Figure 2. The different material boundaries within the tooth overlap in data value such that the boundaries intersect when projected to any two of the dimensions in the transfer function domain. Thus, 2D transfer functions are unable to accurately and selectively render the different material boundaries present. However, 3D transfer functions can easily accomplish this.

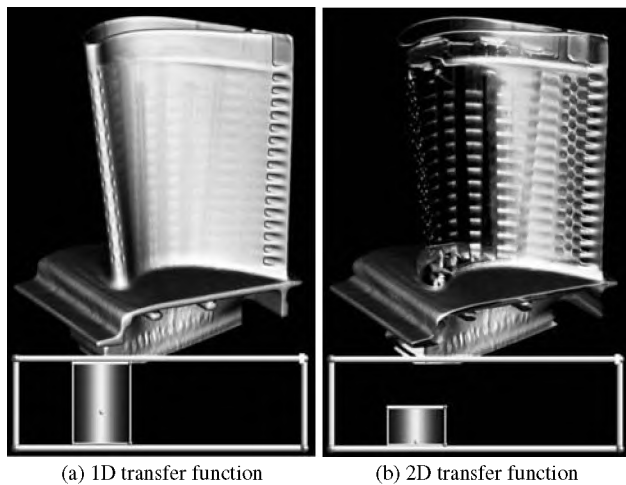


Figure 1: A 1D transfer function (a) is emulated by assigning opacity regardless of gradient magnitude (vertical axis in lower frame). A 2D transfer function (b) giving opacity to only low gradient magnitudes reveals internal structure.

5 Direct Manipulation Widgets

The three reasons for difficulty in transfer function specification, which were outlined in the previous section, can be considered in the context of a conceptual gap between the spatial and transfer function domains. Having intuition and knowledge of both domains is important for creating good transfer functions, but these two domains have very different properties and characteristics. The spatial domain is the familiar 3D space for the geometry and volume data being rendered, but the transfer function domain is more abstract. Its dimensions (data value and two types of derivative) are not spatial, and the quantity at each location (opacity and three



Figure 2: In (a), the 2D transfer function is intended to render all material interfaces *except* the enamel-background boundary at the top of the tooth. However, by using a 3D transfer function (b), with lower opacity for non-zero second derivatives, the previously hidden dentin-enamel boundary is revealed.

color channels) is not scalar. Thus, a principle of the direct manipulation widgets presented here is to link interaction in one domain with feedback in another, so as to build intuition for the connection between them. Also, the conceptual gap between the domains can be reduced by facilitating interaction in both domains simultaneously. We will provide a brief description of how a user might interact with this system and then describe the individual direct manipulation widgets in detail.

In a typical session with our system, a user might begin by moving and rotating a clipping plane through the volume, to inspect slices of data. The user can click on the clipping plane near a region of interest (for example, the boundary between two materials). The resulting visual feedback in the transfer function domain indicates the data value and derivatives at that point and its local neighborhood. By moving the mouse around, volume query locations are constrained to the slice, and the user is able to visualize, in the transfer function domain, how the values change around the feature of interest. Conversely, the system can track, with a small region of opacity in the transfer function domain, the data values at the user-selected locations, while continually updating the volume rendering. This visualizes, in the spatial domain, all other voxels with similar transfer function values. If the user decides that an important feature is captured by the current transfer function, he or she can effectively “paint” that region into the transfer function and continue querying and investigating the volume until all regions of interest have been made visible.

Another possible interaction scenario begins with a pre-determined transfer function that is likely to bring out some features of interest. This can originate with an automated transfer function generation tool [13], or it could be the “default” transfer function described in Section 7. The user would then begin investigating and exploring the dataset as described above. The widgets are used to adapt the transfer function to emphasize regions of interest or eliminate extraneous information.

The process outlined above for creating higher dimensional transfer functions is made possible by the collection of direct manipulation widgets described in the remainder of this section. Embedding the transfer function widget in the main rendering window (say, below the volume rendering, as has been done for all the figures in this paper) is a simple way to make interaction in either or

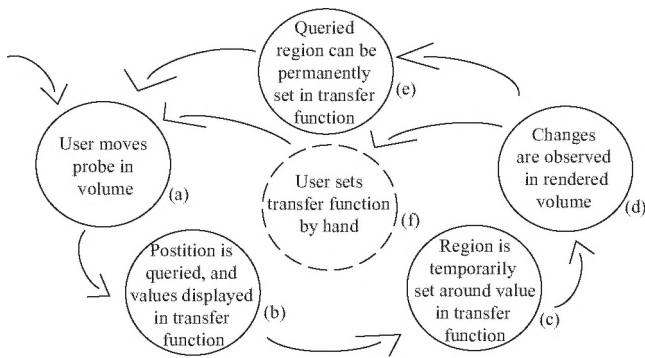


Figure 3: Dual-Domain Interaction

both domains more convenient. More importantly, our system relies on event-based inter-widget communication. This allows information generated by a widget in the spatial domain to determine the state of a widget in the transfer function domain, and vice versa. All widgets are object oriented and derived from a master widget class which specifies a standard set of callbacks for sending and receiving events, allowing the widgets to query and direct each other. Also, widgets can contain embedded widgets, which send and receive events through the parent widget.

5.1 Dual-Domain Interaction

In a traditional volume rendering system, the process of setting the transfer function involves moving the control points (in a sequence of linear ramps defining color and opacity), and then observing the resulting rendered image. That is, interaction in the transfer function domain is guided by careful observation of changes in the spatial domain. We prefer a reversal of this process, in which the transfer function is set by direct interaction in the *spatial domain*, with observation of the transfer function domain. Furthermore, by allowing interaction to happen in both domains simultaneously, the conceptual gap between them is significantly lessened. We use the term “dual-domain interaction” to describe this approach to transfer function exploration and generation.

Figure 3 illustrates the specific steps of dual-domain interaction. When a position inside the volume is queried by the user with the data probe widget (Figure 3a), the values associated with that position (data value, first and second derivative) are graphically represented in the transfer function widget (3b). Then, a small region of high opacity (3c) is temporarily added to the transfer function at the data value and derivatives determined by the probe location. The user has now set a multi-dimensional transfer function simply by positioning a data probe within the volume. The resulting rendering (3d) depicts (in the spatial domain) all the other locations in the volume which share values (in the transfer function domain) with those at the data probe tip. If the features rendered are of interest, the user can copy the temporary transfer function to the permanent one (3e), by, for instance, tapping the keyboard space bar with the free hand. As features of interest are discovered, they can be added to the transfer function quickly and easily with this type of two-handed interaction. Alternately, the probe feedback can be used to manually set other types of classification widgets (3f), which are described later. The outcome of dual-domain interaction is an effective multi-dimensional transfer function built up over the course of data exploration. The widget components which participated in this process can be seen in Figure 4 (on colorplate), which shows how dual-domain interaction can help volume render the CT tooth dataset. The remainder of this section describes the individual widgets and provides additional details about dual-domain interaction.

5.2 Data Probe Widget

The data probe widget is responsible for reporting its tip’s position in volume space and its slider sub-widget’s value. Its pencil-like shape is designed to give the user the ability to point at a feature in the volume being rendered. The other end of the widget orients the widget about its tip. When the volume widget’s position or orientation is modified the data probe widget’s tip tracks its point in volume space. The data probe widget can be seen in Figures 4 and 5 (on colorplate). A natural extension is to link the data probe widget to a haptic device, such as the SensAble PHANTOM, which can provide a direct 3D location and orientation [23].

5.3 Clipping Plane Widget

The clipping plane widget is a basic frame type widget. It is responsible for reporting its orientation and position to the volume widget, which handles the actual clipping when it draws the volume. In addition to clipping, the volume widget will also map a slice of the data to the arbitrary plane defined by the clip widget, and blend it with the volume by a constant opacity value determined by the clip widget’s slider. It is also responsible for reporting the spatial position of a mouse click on its clipping surface. This provides an additional means of querying positions within the volume, distinct from the 3D data probe. The balls at the corners of the clipping plane widget are used to modify its orientation, and the bars on the edges are used to modify its position. The clipping plane widget can also be seen in Figures 5 and 8 (on colorplate).

5.4 Transfer Function Widget

The main role of the transfer function widget is to present a graphical representation of the transfer function domain, in which feedback from querying the volume (with the data probe or clipping plane) is displayed, and in which the transfer function itself can be set and altered. The transfer function widget is shown at the bottom of all of our rendered figures. The backbone of the transfer function widget is a basic frame widget. Data value is represented by position along the horizontal axis, and gradient magnitude is represented in the vertical direction. The third transfer function axis, second derivative, is not explicitly represented in the widget, but quantities and parameters associated with this axis are represented and controlled by other sub-widgets. The balls at the corners of the transfer function widget are used to resize it, as with a desktop window, and the bars on the edges are used to translate its position. The inner plane of the frame is a polygon texture-mapped with a slice through the 3D lookup table containing the full 3D transfer function. The user is presented with a single slice of the 3D transfer function for a few reasons. Making a picture of the entire 3D transfer function would be a visualization in itself, and its image would visually compete with the main volume rendering. Also, since the goal in our work has primarily been the visualization of surfaces, the role of the second derivative axis is much simpler than the other two, so it needs fewer control points.

The data value and derivatives at the position queried in the volume (either via the data probe or clipping plane widgets) is represented with a small ball in the transfer function widget. In addition to the precise location queried, the eight data sample points at the corners of the voxel containing the query location are also represented by balls in the transfer function domain, and are connected together with edges that reflect the connectivity of the voxel corners in the spatial domain. By “re-projecting” a voxel from the spatial domain to a simple graphical representation in the transfer function domain, the user can learn how the transfer function variables (data value and derivatives) are changing near the probe location. The second derivative values are indicated by colormapping the balls: negative, zero, and positive second derivatives are

represented by blue, white, and yellow balls, respectively. When the projected points form an arc, with the color varying through the colormap, the probe is at a boundary in the volume. These can be seen in Figures 4a, 4c, 5a, and 5c (on colorplate). When the re-projected data points are clustered together, the probe is in a homogeneous region, as seen in Figures 4b and 5b. An explanation of the inter-relationships between data values and derivatives which underly these configurations can be found in [12, 13]. As the user gains experience with this representation, he or she can learn to “read” the re-projected voxel as an indicator of the volume characteristics at the probe location.

5.5 Classification Widgets

In addition to the process of dual-domain interaction described above, transfer functions can also be created in a more manual fashion by adding one or more classification widgets to the main transfer function window. The opacity and color contributions from each classification widget sum together to form the transfer function. We have developed two types of classification widget: triangular and rectangular.

The triangular classification widget, shown in Figures 2, 5, and 9, is based on Levoy’s “isovalue contour surface” opacity function [18]. The widget is an inverted triangle with a base point attached to the horizontal data value axis. The horizontal location of the widget is altered by dragging the ball at the base point, and the vertical extent is altered by dragging the bar on the top edge. As the height is modified, the angle subtended by the sides of the triangle is maintained, scaling the width of the top bar. The top bar can also be translated horizontally to shear the triangle. The width of the triangle is modified by moving a ball on the right endpoint of the triangle’s top bar. The classification can avoid assigning opacity to low gradient magnitudes by raising a gradient threshold bar, controlled by a ball on the triangle’s right edge. The trapezoidal region spanned by the widget (between the low gradient threshold and the top bar) defines the data values and gradient magnitudes which receive color and opacity. Color is constant; opacity is maximal along the center of the widget, and it linearly ramps down to zero at the left and right edges.

The triangular classification widgets are particularly effective for visualizing *surfaces* in scalar data. More general transfer functions, for visualizing data which may not have clear boundaries, can be created with the rectangular classification widget. This widget is seen in Figure 1. The rectangular widget has a top bar which translates the entire widget freely in the two visible dimensions of the transfer function domain. The balls located at the top right and the bottom left corners resize the widget. The rectangular region spanned by the widget defines the data values and gradient magnitudes which receive opacity and color. Like the triangular widget, color is constant, but the opacity is more flexible. It can be constant, or fall off in various ways: quadratically as an ellipsoid with axes corresponding to the rectangle’s aspect ratio, or linearly as a ramp, tent, or pyramid.

For both types of classification widget (triangular and rectangular), additional controls are necessary to use the third dimension of the transfer function domain: the second derivative of the scalar data. Because our research has focused on visualizing boundaries between material regions, we have consistently used the second derivative to emphasize the regions where the second derivative magnitude is small or zero. Specifically, maximal opacity is always given to zero second derivative, and decreases linearly towards the second derivative extremal values. How much the opacity changes as a function of second derivative magnitude is controlled with a single slider, called the “boundary emphasis slider.” Because the individual classification widgets can have various sizes and locations, it is easiest to always locate the boundary emphasis slider on the top edge of the *transfer function* widget. The slider controls

the boundary emphasis for whichever classification widget is currently selected. With the slider in its left-most position, zero opacity is given to extremal second derivatives; in the right-most position, opacity is constant with respect to the second derivative.

While the classification widgets are usually set by hand in the transfer function domain, based on feedback from probing and re-projected voxels, their placement can also be somewhat automated. This further reduces the difficulty of creating an effective higher dimensional transfer function. The classification widget’s location and size in the transfer function domain can be tied to the distribution of the re-projected voxels determined by the data probe location. For instance, the rectangular classification widget can be centered at the transfer function values interpolated at the data probe’s tip, with the size of the rectangle controlled by the data probe’s slider. Or, the triangular classification widget can be located horizontally at the data value queried by the probe, with the width and height determined by the horizontal and vertical variance in the re-projected voxel locations.

5.6 Shading Widget

The shading widget is a collection of spheres which can be rendered in the scene to indicate and control the light direction and color. Fixing a few lights in view space is generally effective for renderings, therefore changing the lighting is an infrequent operation.

5.7 Color Picker Widget

The color picker is an embedded widget which is based on the hue-lightness-saturation (HLS) color space. Interacting with this widget can be thought of as manipulating a sphere with hues mapped around the equator, gradually becoming black at the top, and white at the bottom. To select a hue, the user moves the mouse horizontally, rotating the ball around its vertical axis. Vertical mouse motion tips the sphere toward or away from the user, shifting the color towards white or black. Saturation and opacity are selected independently using different mouse buttons with vertical motion. While this color picker can be thought of as manipulating this HLS sphere, it actually renders no geometry. Rather, it is attached to a sub-object of another widget. The triangular and rectangular classification widgets embed the color picker in the polygonal region which contributes opacity and color to the transfer function domain. The shading widget embeds a color picker in each sphere that represents a light. The user specifies a color simply by clicking on that object, then moving the mouse horizontally and vertically until the desired hue and lightness are visible. In most cases, the desired color can be selected with a single mouse click and gesture.

6 Hardware Considerations

While this paper is conceptually focused on the matter of setting and applying higher dimensional transfer functions, the quality of interaction and exploration described would not be possible without the use of modern graphics hardware. Our implementation relies heavily on an OpenGL extension known as pixel textures, or dependent textures. This extension can be used for both classification and shading. In this section, we describe our modifications to the classification portion of the traditional hardware volume rendering pipeline. We also describe a multi-pass/multi-texture method for adding interactive shading to the pipeline.

The volume rendering pipeline utilizes separate data and shading volumes. The data volume, or “VGH” in Figures 6 and 7, encodes data value (“V”), gradient magnitude (“G”), and second derivative (“H”, for Hessian) in the color components of a 3D texture, using eight bits for each of these three quantities. The quantized normal volume, or “QN” in Figure 6, encodes normal direction as a 16-bit

unsigned short in two eight-bit color components of a 3D texture. The “Normal” volume in Figure 7 encodes the normal in a scaled and biased RGB texture, one normal component per color channel. Conceptually, both the data and shading/normal volumes are spatially coincident. A slice through one volume can be represented by the same texture coordinates in the other volume.

6.1 Pixel Texture

Pixel texture is a hardware extension which has proven useful in computer graphics and visualization [6, 9, 22, 33]. Pixel texture and dependent texture are names for operations which use color fragments to generate texture coordinates, and replace those color fragments with the corresponding entries from a texture. This operation essentially amounts to an arbitrary function evaluation via a lookup table. The number of parameters is equal to the dimension of components in the fragment which is to be modified. For example, if we were to pixel texture an RGB fragment, each channel value would be scaled to between zero and one, and these new values would then be used as texture coordinates into a 3D texture. The color values for that location in the 3D texture replace the original RGB values. Nearest neighbor or linear interpolation can be used to generate the replacement values. The ability to scale and interpolate color channel values is a convenient feature of the hardware. It allows the number of elements along a dimension of the pixel texture to differ from the number of bit planes in the component that generated the texture coordinate. Without this flexibility, the size of a 3D pixel texture would be prohibitively large.

6.2 Classification

Each voxel in our data volume contains three values. We therefore require a 3D pixel texture to specify color and opacity for a sample. It would be prohibitively expensive to give each axis of the pixel texture full eight-bit resolution, or 256 entries along each axis. We feel that data value and gradient magnitude variation warrants full eight-bit resolution. Because we are primarily concerned with the zero crossings in a second derivative, we choose to limit the resolution of this axis. Since the second derivative is a signed quantity, we must maintain the notion of its sign (with a scale and bias) in order to properly interpolate this quantity. We can choose a limited number of control points for this axis and represent them as “sheets” in the pixel texture. Specifically, we can exert linear control over the opacity of second derivative values with three control points: one each for negative, zero, and positive values. The opacity on the center sheet, representing zero second derivatives, is directly controlled by the classification widgets. The opacity on the outer sheets, representing positive and negative second derivatives, is scaled from the opacity on the central sheet according to the boundary emphasis slider. It is important to note here that if a global boundary emphasis is desired, *i.e.*, applying to all classification widgets equally, one could make this a separable portion of the transfer function simply by modulating the output of a 2D transfer function with the per-sample boundary emphasis value.

6.3 Shading

Shading is a fundamental component of volume rendering because it is a natural and efficient way to express information about the shape of structures in the volume. However, much previous work with texture-memory based volume rendering lacks shading. We include a description of our shading method here not because it is especially novel, but because it dramatically increases the quality of our renderings with a negligible increase in rendering cost.

Since there is no efficient way to interpolate normals in hardware which avoids redundancy and truncation in the encoding, normals

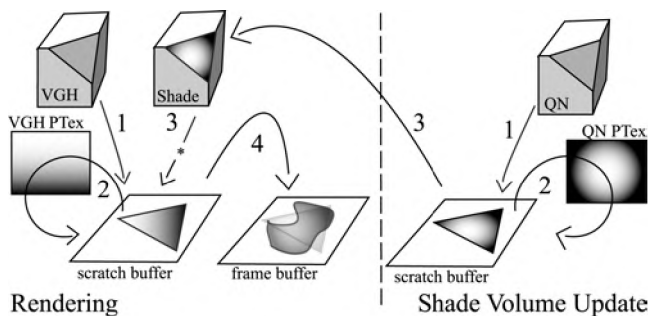


Figure 6: Octane2 Volume Rendering pipeline. Updating the shade volume (right) happens after the volume has been rotated. Once updated, the volume would then be re-rendered.

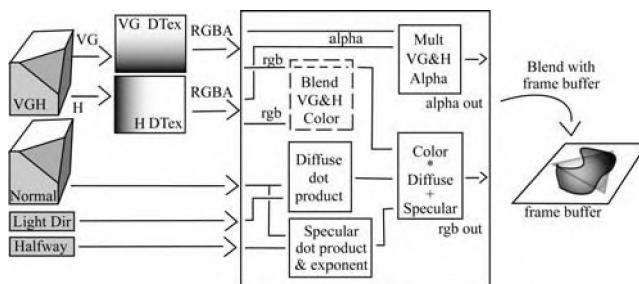


Figure 7: GeForce3 Volume Rendering pipeline. Four-way multi-texture is used. The textures are: VGH, VG Dependant Texture, H Dependant texture, and the Normal texture (for shading). The central box indicates the register combiner stage. The Blend VG&H Color stage is not usually executed since we rarely vary color along the second derivative axis. The Multiply VG&H Alpha stage, however, is required since we must compose our 3D transfer function separately as a 2D×1D transfer function.

are encoded using a 16-bit quantization scheme, and we use nearest-neighbor interpolation for the pixel texture lookup. Quantized normals are lit using a 2D pixel texture since there is essentially no difference between a 16-bit 1D nearest neighbor pixel texture and an eight-bit per-axis 2D nearest neighbor pixel texture. The first eight bits of the quantized normal can be encoded as the red channel and the second eight bits are encoded as the green channel. We currently generate the shading pixel texture on a per-view basis in software. The performance cost of this operation is minimal. It, however, could easily be performed in hardware as well. Each quantized normal could be represented as a point with its corresponding normal, rendered to the frame buffer using hardware lighting, and then copied from the frame buffer to the pixel texture. Some hardware implementations, however, are not flexible enough to support this operation.

6.4 Hardware Implementation

We have currently implemented a volume renderer using multi-dimensional transfer functions on the *sgi* Octane 2 with the V series graphics cards, and the *nVidia* GeForce3 series graphics adaptor. The V series platform supports 3D pixel texture, albeit only on either a *glDrawPixels()* or *glCopyPixels()* operation. Since pixel texture does not occur directly on a per-fragment basis during rasterization, we must first render the slice to a buffer, then pixel texture it using a *glCopyPixels()* operation. Our method requires a scratch, or auxiliary, buffer since each slice must be rendered individually and then composited. If shading is enabled, a match-

ing slice from a shading volume is rendered and modulated (multiplied) with the current slice. The slice is then copied from the scratch buffer and blended with previously rendered slices in the frame buffer. A key observation of this volume rendering process is that when the transfer function is being manipulated or changed, the view point is static, and vice versa. This means that we only need to use the pixel texture operation on the portion of the volume which is currently changing. When the user is manipulating the transfer function, the raw data values (VGH) are used for the volume texture, and a pre-computed RGBA shade volume is used.

The left side of Figure 6 illustrates the rendering process. The slices from the VGH data volume are first rendered (1) and then pixel textured (2). The “Shade” slice is rendered and modulated with the classified slice (3), then blended into the frame buffer (4). When the volume is rotated, lighting must be updated (shown on the right side of Figure 6). For interactive efficiency, we only update the shade volume once a rotation has been completed. A new quantized normal pixel texture (for shading) is generated and each slice of the quantized normal volume is rendered orthographically in the scratch buffer (1) and then pixel textured (2). This slice is then copied from the scratch buffer to the corresponding slice in the shade volume (3). The volume is then re-rendered with the updated shade volume. Updating the shade volume in hardware requires that the quantized normal slices are always smaller than scratch buffer’s dimensions.

The GeForce3 series platform supports dependent texture reads on a per-fragment basis as well as 4-way multi-texture, see Figure 7. This means that the need for a scratch buffer is eliminated, which significantly improves rendering performance by avoiding several expensive copy operations. Unfortunately, this card only supports 2D dependent texture reads. This constrains the 3D transfer functions to be a separable product of a 2D transfer function (in data value and gradient magnitude) and a 1D transfer function (in second derivative), but it also allows us to take full advantage of the eight-bit resolution of the dependent texture along the second derivative axis. The second derivative axis is implemented with the nVidia register combiner extension. Shading can either be computed as described above, or using the register combiners.

7 Discussion

Using multi-dimensional transfer functions heightens the importance of densely sampling the voxel data in rendering. With each new axis in the transfer function, there is another dimension along which neighboring voxels can differ. It becomes increasingly likely that the data sample points at the corners of a voxel straddle an important region of the transfer function (such as a region of high opacity) instead of falling within it. Thus, in order for the boundaries to be rendered smoothly, the distance between view-aligned sampling planes through the volume must be very small. Most of the figures in this paper were generated with sampling rates of about 6 to 10 samples per voxel. At this sample rate, frame updates can take nearly two seconds on the Octane2, and nearly a second on the GeForce3. For this reason, we lower the sample rate during interaction, and re-render at the higher sample rate once an action is completed. During interaction, the volume rendered surface will appear coarser, but the surface size and location are usually readily apparent. Thus, even with lower volume sampling rates during interaction, the rendered images are effective feedback for guiding the user in transfer function exploration.

One benefit of using our 3D transfer functions is the ability to use a “default” transfer function which is produced without any user interaction. Given our interest in visualizing the boundaries between materials, this was achieved by assigning opacity to high gradient magnitudes and low-magnitude second derivatives, regardless of data value, while varying hue along the data value. This de-

fault transfer function is intended only as a starting point for further modification with the widgets, but often it succeeds in depicting the main structures of the volume, as seen in Figure 8 (on colorplate). Other application areas for volume rendering may need different variables for multi-dimensional transfer functions, with their own properties governing the choices for default settings.

Dual-domain interaction has utility beyond setting multi-dimensional transfer functions. Of course, it can assist in setting 1D transfer functions, as well as isovalues for isosurface visualization. Dual-domain interaction also helps answer other questions about the limits of direct volume rendering for displaying specific features in the data. For example, the feedback in the transfer function domain can show the user whether a certain feature of interest detected during spatial domain interaction is well-localized in the transfer function domain. If re-projected voxels from different positions, in the same feature, map to widely divergent locations in the transfer function domain, then the feature is not well-localized, and it may be hard to create a transfer function which clearly visualizes it. Similarly, if probing inside two distinct features indicates that the re-projected voxels from both features map to the same location in the transfer function domain, then it may be difficult to selectively visualize one or the other feature.

A surprising variety of different structures can be extracted with multi-dimensional transfer functions, even from standard datasets which have been rendered countless times before. For instance, Figure 9 shows how using the clipping plane and probing makes it easy to detect and then visualize the surface of the frontal sinuses (above the eyes) in the well-known UNC Chapel Hill CT Head dataset, using a 3D transfer function. This is a good example of a surface that can not be visualized using isosurfacing or 1D transfer functions.

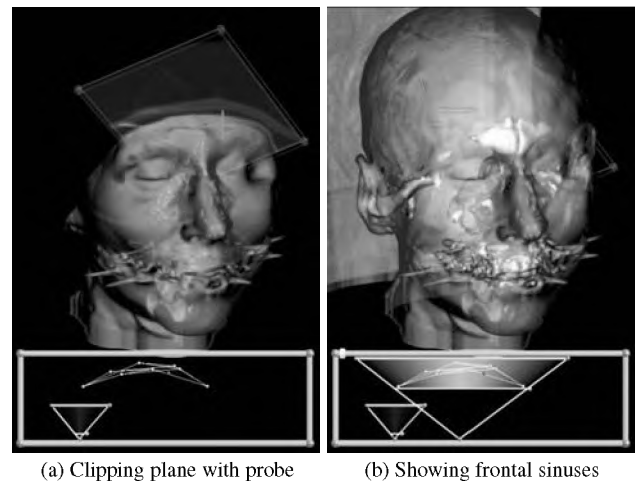


Figure 9: A clipping plane cuts through the region above the eyes. Probing in the area produces a re-projected voxel with the characteristic arc shape indicating the presence of a surface (a). A similarly placed triangular classification widget reveals the shape of the sinus (b).

8 Future Work

One unavoidable drawback to using multi-dimensional transfer functions is the increased memory consumption needed to store all the transfer function variables at each voxel sample point. This is required because a hardware-based approach can not compute these quantities on the fly. Combined with the quantized normal volume (which takes three bytes per voxel instead of two, due to pixel field alignment restrictions), we require six bytes per voxel

to represent the dataset. This restricts the current implementation with 104 MB of texture memory to $256 \times 256 \times 128$ datasets. Future work will expand the dataset size using parallel hardware rendering methods [14].

Utilizing multi-dimensional transfer functions opens the possibility of rendering multi-variate volume data, such as a fluid flow simulation or meteorological data. One challenge here is determining which quantities are mapped to the transfer function axes, and whether to use data values directly, or some dependent quantity, such as a spatial derivative.

Future commodity graphics cards will provide an avenue for expanded rendering features. Specifically, both the nVidia and ATI graphics cards support a number of per-pixel operations which can significantly enhance the computation of diffuse and specular shading (assuming a small number of light sources). These features, however, come at the expense of redundancy and truncation in normal representation. Pixel texture shading, on the other hand, allows arbitrarily complex lighting and non-photorealistic effects. The trade-off between these two representations is normal interpolation. Quantized normals do not easily interpolate; vector component normals do. Vector component normals, however, do require a normalization step after component-wise interpolation if the dot product is for accurately computing the diffuse and specular lighting component. This normalization step is not yet supported by these cards.

Direct manipulation widgets and spatial interaction techniques lend themselves well to immersive environments. We would like to experiment with dual-domain interaction in a stereo, tracked, environment. We speculate that an immersive environment could make interacting with a 3D transfer function more natural and intuitive. We would also like to perform usability studies on our direct manipulation widgets and dual-domain interaction technique, as well as perceptual studies on 2D and 3D transfer functions for volume rendering.

9 Acknowledgments

This research was funded by grants from the Department of Energy (VIEWS 0F00584), the National Science Foundation (ASC 8920219, MRI 9977218, ACR 9978099), and the National Institutes of Health National Center for Research Resources (1P41RR12553-2). The authors would like to thank sgi for their generous Octane2 equipment loan.

References

- [1] Chandrajit L. Bajaj, Valerio Pascucci, and Daniel R. Schikore. The Contour Spectrum. In *Proceedings IEEE Visualization 1997*, pages 167–173, 1997.
- [2] Lawrence D. Bergman, Bernice E. Rogowitz, and Lloyd A. Treinish. A Rule-based Tool for Assisting Colormap Selection. In *Proceedings Visualization 1995*, pages 118–125. IEEE, October 1995.
- [3] Brian Cabral, Nancy Cam, and Jim Foran. Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. In *ACM Symposium On Volume Visualization*, 1994.
- [4] D. Brookshire Conner, Scott S. Snibbe, Kenneth P. Herndon, Daniel C. Robbins, Robert C. Zeleznik, and Andries van Dam. Three-Dimensional Widgets. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, pages 183–188, 1992.
- [5] C.Rezk-Salama, K.Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive Volume Rendering on Standard PC Graphics Hardware Using Multi-Textures and Multi-Stage Rasterization. In *Siggraph/Eurographics Workshop on Graphics Hardware 2000*, 2000.
- [6] Klaus Engel, Martin Kraus, and Thomas Ertl. High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading. In *Siggraph/Eurographics Workshop on Graphics Hardware 2001*, 2001.
- [7] Allen Van Gelder and Kwansik Kim. Direct Volume Rendering with Shading via Three-Dimensional Textures. In *ACM Symposium On Volume Visualization*, pages 23–30, 1996.
- [8] Taosong He, Lichan Hong, Arie Kaufman, and Hanspeter Pfister. Generation of Transfer Functions with Stochastic Search Techniques. In *Proceedings IEEE Visualization 1996*, pages 227–234, 1996.

- [9] Wolfgang Heidrich, Rudiger Westermann, Hans-Peter Seidel, and Thomas Ertl. Applications of Pixel Textures in Visualization and Realistic Image Synthesis. In *Proceedings of the 1999 Symposium on Interactive 3D Graphics*, 1999.
- [10] Kenneth P. Herndon and Tom Meyer. 3D Widgets for Exploratory Scientific Visualization. In *Proceedings of UIST '94 (SIGGRAPH)*, pages 69–70. ACM, November 1994.
- [11] Jiř Hladůvka, Andreas König, and Eduard Gröller. Curvature-Based Transfer Functions for Direct Volume Rendering. In Bianca Falcidieno, editor, *Spring Conference on Computer Graphics 2000*, volume 16, pages 58–65, May 2000.
- [12] Gordon Kindlmann. Semi-Automatic Generation of Transfer Functions for Direct Volume Rendering. Master's thesis, Cornell University, Ithaca, NY, January 1999.
- [13] Gordon Kindlmann and James W. Durkin. Semi-Automatic Generation of Transfer Functions for Direct Volume Rendering. In *IEEE Symposium On Volume Visualization*, pages 79–86, 1998.
- [14] Joe Kniss, Patrick S. McCormick, Allen McPherson, James Ahrens, Jamie Painter, Alan Keahey, and Charles Hansen. Interactive Texture-Based Volume Rendering for Large Data Sets. *IEEE Computer Graphics and Applications*, 21(4):52–61, July/August 2001.
- [15] Andreas König and Eduard Gröller. Mastering Transfer Function Specification by Using VolumePro Technology. In Toshiyasu L. Kunii, editor, *Spring Conference on Computer Graphics 2001*, volume 17, pages 279–286, April 2001.
- [16] Philip Lacroute and Marc Levoy. Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transform. In *ACM Computer Graphics (SIGGRAPH '94 Proceedings)*, pages 451–458, July 1994.
- [17] Eric LaMar, Bernd Hamann, and Kenneth I. Joy. Multiresolution Techniques for Interactive Texture-Based Volume Visualization. In *Proceedings Visualization '99*, pages 355–361. IEEE, October 1999.
- [18] Marc Levoy. Display of Surfaces from Volume Data. *IEEE Computer Graphics & Applications*, 8(5):29–37, 1988.
- [19] J. Marks, B. Andalman, P.A. Beardsley, and H. Pfister et al. Design Galleries: A General Approach to Setting Parameters for Computer Graphics and Animation. In *ACM Computer Graphics (SIGGRAPH '97 Proceedings)*, pages 389–400, August 1997.
- [20] D. Marr and E. C. Hildreth. Theory of Edge Detection. *Proceedings of the Royal Society of London*, B 207:187–217, 1980.
- [21] Jerrold E. Marsden and Anthony J. Tromba. *Vector Calculus*, chapter 2.6, 4.2. W.H. Freeman and Company, New York, 1996.
- [22] Michael Meissner, Ulrich Hoffmann, and Wolfgang Strasser. Enabling Classification and Shading for 3D Texture Mapping based Volume Rendering using OpenGL and Extensions. In *IEEE Visualization 1999*, pages 207–214, 1999.
- [23] Timothy Miller and Robert C. Zeleznik. The Design of 3D Haptic Widgets. In *Proceedings 1999 Symposium on Interactive 3D Graphics*, pages 97–102, 1999.
- [24] Shigeru Muraki. Multiscale Volume Representation by a DoG Wavelet. *IEEE Trans. Visualization and Computer Graphics*, 1(2):109–116, 1995.
- [25] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler. The VolumePro Real-Time Ray-Casting System. In *ACM Computer Graphics (SIGGRAPH '99 Proceedings)*, pages 251–260, August 1999.
- [26] Hanspeter Pfister, Chandrajit Bajaj, Will Schroeder, and Gordon Kindlmann. The Transfer Function Bake-Off. In *Proceedings IEEE Visualization 2000*, pages 523–526, 2000.
- [27] Hanspeter Pfister and Arie E. Kaufman. Cube-4 - A Scalable Architecture for Real-Time Volume Rendering. In *IEEE Symposium On Volume Visualization*, pages 47–54, 1996.
- [28] James T. Purciful. Three-Dimensional Widgets for Scientific Visualization and Animation. Master's thesis, University of Utah, June 1997.
- [29] Penny Rheingans. Task-Based Color Scale Design. In *Proceedings Applied Image and Pattern Recognition*. SPIE, October 1999.
- [30] Yoshinabu Sato, Carl-Fredrik Westin, and Abhir Bhalerao. Tissue Classification Based on 3D Local Intensity Structures for Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):160–179, April-June 2000.
- [31] Paul S. Strauss and Rikk Carey. An Object-Oriented 3D Graphics Toolkit. In *ACM Computer Graphics (SIGGRAPH '92 Proceedings)*, pages 341–349, July 1992.
- [32] Colin Ware. Color Sequences for Univariate maps: Theory, Experiments, and Principles. *IEEE Computer Graphics and Applications*, 8(5):41–49, September 1988.
- [33] Rudiger Westermann and Thomas Ertl. Efficiently Using Graphics Hardware in Volume Rendering Applications. In *ACM Computer Graphics (SIGGRAPH '98 Proceedings)*, pages 169–176, August 1998.
- [34] R. C. Zeleznik, K. P. Herndon, D. C. Robbins, N. Huang, T. Meyer, N. Parker, and J. F. Hughes. An Interactive Toolkit for Constructing 3D Widgets. *Computer Graphics*, 27(4):81–84, 1993.