Interactive Rendering of Multivalued Volume Data with Layered Complementary Volumes

Category: Research

Abstract

We present a framework for rendering multivalued volumes interactively and flexibly. Users can explore these volumes more effectively, particularly relationships among the multiple data values defined at each spatial location, because the renderer provides interactive control over the visual relation of the values.

The four steps of our framework transform multivalued datasets into interactive visualizations that can be explored and manipulated. We first derive new values from the primary datasets. We then abstract both the primary and derived values into visual representations. In the third step we map the data through transfer functions that produce color and opacity. Finally, we render the layers interactively as users manipulate the transfer functions.

Contributions of this work include the conceptual framework, the interactive rendering of multivalued volumes, a thread-like density volume to represent continuous directional information, a complementary volume to generate a halo around each thread, a derived exploratory culling volume interactively to control control the complexity of a layer, and an interactive implementation using commodity PC graphics hardware. We demonstrate these contributions with a series of example visualizations of 2nd-order-tensor-valued MRI data and with simulated 3D fluid flow data.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism; I.4.10 [Computer Graphics]: Image Processing and Computer Vision—Image Representation Volumetric J.3 [Computer Applications]: Life and Medical Sciences

Keywords: Scientific Visualization, Diffusion Tensor Imaging (DTI), Diffusion, Fluid Flow, Medical Imaging, Direct Volume Rendering, Volume Graphics, Volume Shading, Multi-textures, PC Graphics Hardware

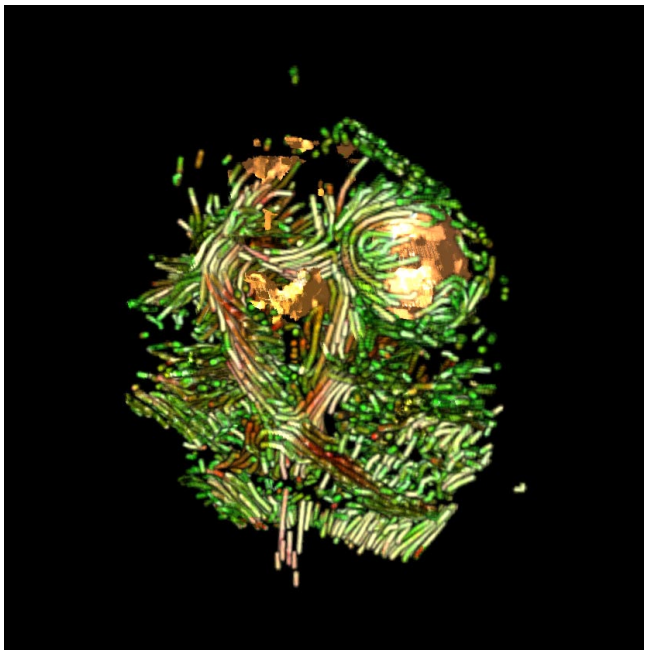

Figure 1: Interactive volume rendering of [update entire to detail info we see when we have real image] one hemisphere of a human brain dataset. At each point in the dataset a tensor-valued diffusion-rate measurement captures how fast water diffuses in all directions (different directions can have different rates). The rendering shows correlations among components of the tensor field and with an additional scalar field. The volume renderings permit interactive exploration of these multivalued volumes and enhance understanding of the data and the underlying white-matter structures.

1 Introduction

We describe a new multilayer interactive volume-rendering approach to exploring multivalued 3D scientific data. Our volume-rendering work has been driven primarily by two scientific applications: understanding anatomy, development, and pathology in the brain from neuroimaging and understanding blood flow in coronary arteries using direct numerical simulation data. These driving applications provide the

problems; the extent to which our application facilitates their solution helps to evaluate and guide the algorithm and tool development [Brooks 1996].

We believe that data exploration is essential to the scientific process. A scientific hypothesis can often be evaluated with a distillation of acquired data; but understanding the support for a hypothesis or developing a new one can best be done through a more complete understanding of the data. This is particularly true for multivalued scientific data, in which relationships among values make the problem more complex.

Creating comprehensive and accurate visualizations for exploring 3D multivalued data is challenging. The first challenge is to create visualizations in which the data nearer to the viewer does not excessively obscure that farther away: choices must be made about what to show and how to leave visual bandwidth for things farther away. The second challenge is to represent many values and their interrelationships at each spatial location. As an example, our neuroimaging data has seven primary values at each point and additional derived values can further illuminate the underlying biology. Similarly, the flow data has four primary values at each point, as well as another three to six useful derived values. The multiple values exacerbate the obscuration problem not only because more values must be shown at each spatial location, but also because important relationships among the different data values require even more of the precious visual bandwidth. The third challenge is to convey relationships among different spatial locations – arrows at different points can show velocity, but a streamline connecting these velocities at many points shows an important spatial relationship. The fourth challenge is to render the large datasets interactively – for our examples, datasets sufficient to address the scientific questions under study generally contain 256^3 samples. Polygonal representations conveying as much complexity as volume rendering for datasets that large are difficult to render interactively.

1.1 Interactive Complementary Volume Layering

We build on the idea of multilayer 2D visualizations [Laidlaw et al. 1998c; Laidlaw et al. 1998b; Kirby et al. 1999], implementing an interactive 3D analog to 2D compositing. Figure 1 shows a volume rendering of seven-valued brain-imaging data. One layer of this rendering is a direct volume rendering conveying tissue type; two complementary layers show connectivity information with a thread-like representation, and halos augmenting the thread that give a better sense of depth relationships. Figure 2 shows how layers

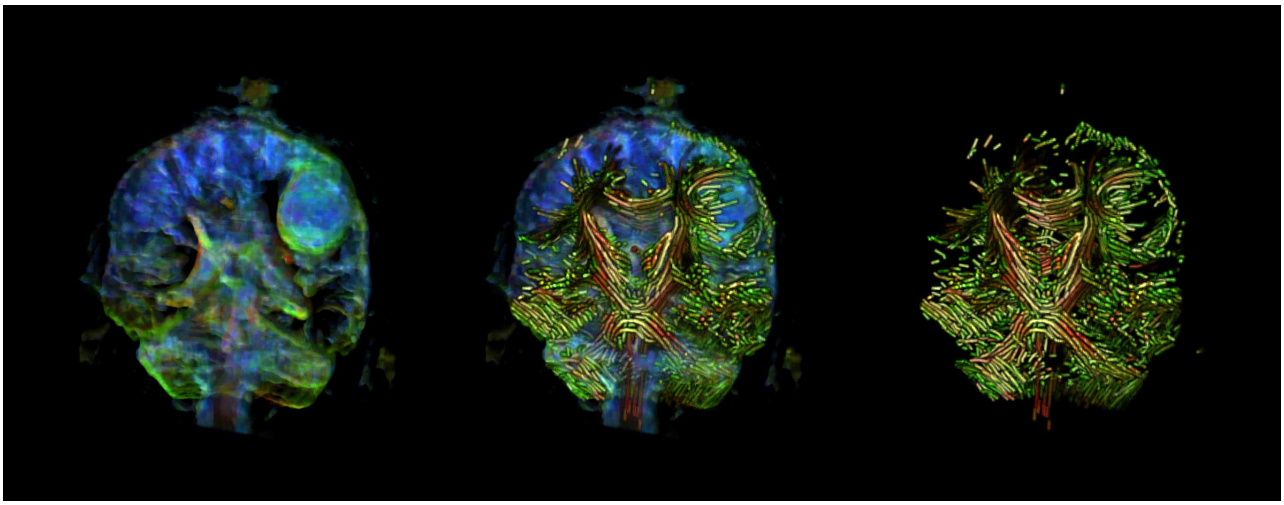

Figure 2: Left: a direct-volume-rendered layer showing regions with different diffusion anisotropy. Right: a hair layer showing both the anisotropy and diffusion direction. The two layers are combined in the center image. Combining the multiple layers requires little additional visual bandwidth and conveys significantly more information.

can combine. Our conceptual framework breaks the design process into four steps: deriving additional data values from the primary ones, abstracting the values visually, mapping the abstractions through transfer functions to hardware primitives, and interacting with the visualizations. We demonstrate images created from combinations of scalar, vector, and tensor fields, using a new 'thread and halo' implementation in our visual representation for vector fields. Figure 3 illustrates the thread represention with and without halos. We also describe data-driven culling of the visualization. The system is implemented on a PC using a commodity graphics card, and users can interactively control parts of the visual mapping through 1D and 2D transfer functions.

In the next section we discuss related work. We then describe each of the main contributions of the work in individual sections: layering complementary volumes, specifying and creating layers, complementary thread and halo layers, interactive culling, and editing one- and two-valued transfer functions. Results for our two driving applications are then presented and discussed, and a discussion of some of the lessons learned and issues raised precedes the summary and conclusions. Finally, we give hardware-specific details of the implementation.

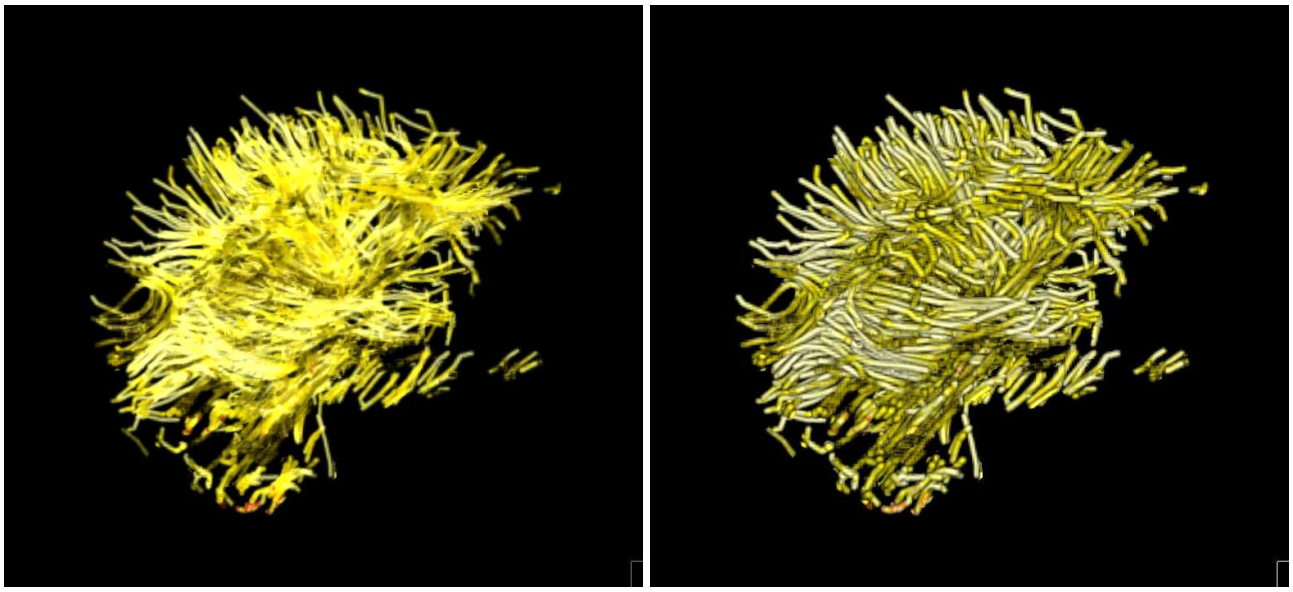

Figure 3: A threads volume showing brain connectivity information (the front of the head points right in both images). Halos, rendered only in the right image, clarify the depth ordering and 3D structure.

2 Related Work

Below we survey relevant work in diffusion tensor field visualization, vector field visualization, hardware-accelerated volume rendering, and thread rendering.

2.1 Visualization of Diffusion Tensor Fields

Several approaches have been made to visualizing diffusion tensor imaging (DTI) datasets. Since a diffusion tensor is a symmetric matrix with positive eigenvalues, an ellipsoid is a natural geometric representation for it. Pierpaoli *et al* [Pierpaoli and Basser 1996] used a 2D array of ellipsoids to visualize a 2D diffusion tensor field. To give a more continuous visual appearance, Laidlaw *et al* [Laidlaw et al. 1998a] normalized the ellipsoids. Additionally, they used concepts from oil painting – mapping data components onto brush strokes and building up the strokes in multiple layers – to represent more of the data, creating a second kind of 2D visualization showing all of the multiple values simultaneously.

None of the 2D methods generalize well to 3D. Placing the ellipsoids in 3D to visualize a 3D dataset has

two drawbacks. First, placing an ellipsoid at every data point in three-space means that only the outermost layer of ellipsoids is visible; all the others are obscured. Second, continuity cannot be shown in an ellipsoid representation, and without continuity information, neural connectivity is difficult to understand.

Kindlmann *et al* [Kindlmann and Weinstein 1999] overcame the problem of obscuring data points in 3D with a direct-volume-rendering approach: to every data point they assign a certain opacity and color based on the underlying diffusion tensor dataset, using the concept of a "barycentric map" for the opacity and "hue-balls" and "lit-tensors" for coloring and lighting. However, it is still difficult to pick out anatomically distinct regions and understand their connectivity. Our similar direct-volume-rendering technique makes connectivity more apparent by using a coloring and lighting concept based on the diffusion magnitude and the diffusion anisotropy. We also expand the use of the barycentric opacity map to generate color in addition to opacity.

Basser *et al* [Basser et al. 2000] calculated the trajectories of neural fibers in brain white matter that were generated from the diffusion tensor field by integrating along the eigenvector with the largest eigenvalue. Zhang *et al* [Zhang et al. 2000] used this method to generate streamtubes to visualize continuous directional information in the brain (the streamtubes are calculated during a preprocessing step). The streamtubes solve the problem of representing continuity but at the cost of the information in areas without streamtubes. We extend Zhang et al.'s algorithm to continue streamtubes through areas with planar anisotropy. Additionally, we filter the streamtube paths into a thread density volume rather than representing them as polygonal models.

2.2 Visualization of Vector Fields

Of the extensive work on creating effective visualizations of vector fields, the following two papers are closely related to our work. Interrante and Grosch [Interrante and Grosch 1997] visualized 3D flow with volume line integral convolution (LIC). Adapting artistic techniques, they introduced 3D visibility-impeding halos to improve the perception of depth discontinuities in a static volume renderer. As they demonstrated with offline rendering, halos improve depth perception and help make complex 3D structures easier to read. Our technique for achieving this halo effect in real time uses a texture-based volume renderer that makes complex 3D structures easier to read in the visualizations.

Zöckler *et al* [Zöckler et al. 1996] introduced illuminated field lines to visualize 3D vector fields. Our illuminated tube representation is similar, but our volumetric rendering approach renders at a rate independent of the tube or line complexity, and combines with our other volumetric layers to create visualizations that can convey more information.

2.3 Hardware-Accelerated Volume Rendering

Hardware-accelerated volume rendering often uses texture memory to store 3D volumes. The data is then mapped onto slices through the volume and blended together to render an image.

Cabral *et al* [Cabral et al. 1994] introduced a 2D texture approach for such rendering. Three stacks of 2D slices are generated through the volume, one perpendicular to each of coordinate axes. As the viewpoint changes, the renderer chooses the best stack to render. This approach exploits hardware-based bilinear interpolation in the plane of the slice. However, three copies of the volume must be stored in texture memory.

Van Gelder and Kim [Van Gelder and Kim 1996] avoid the redundant data copies by rendering with 3D textures. Generally view-aligned slices are used, exploiting trilinear interpolation in hardware. Our implementation uses 3D textures with view-aligned slices. In addition, we use hardware texture compression to further reduce texture memory consumption.

Several volume-rendering implementations use commodity graphics cards [**?**] or distributed hardware [**?**]. However, these approaches have not targeted visualizing multivalued datasets.

Kniss *et al* [Kniss et al. 2001] use interactive transfer functions operating on directional derivative values to select boundaries in scalar-valued datasets. We use this technique to visualize our scalar-valued datasets, although our interactive manipulation widgets are less sophisticated than theirs. We extend this technique to multivalued datasets by deriving a scalar value from the original dataset and then calculating the directional derivative from the derived value.

Engel *et al* [Engel et al. 2001] rendered more quickly with fewer slices by using additional precalculated volumes. We use the same hardware they do, but target multivalued datasets and do not use their precalculated volumes to reduce slice count.

Lum and Ma [Lum and Ma 2002] implemented a hardware-accelerated parallel nonphotorealistic vol-

ume renderer that uses multi-pass rendering on consumer-level graphics cards. Their system emphasizes edges or depth ordering using artistically motivated techniques. Our system also implements a multi-pass volume renderer; like Lum and Ma, we utilize multiple rendering passes to enhance visual cues, but our rendering is targeted to exploratory visualization of multivalued data.

2.4 Hair Rendering

Kajiya and Kay [Kajiya and Kay 1989] introduced texels to render realistic-looking fur. Texels are 3D texture maps in which both a surface frame (normal and tangent) and the parameters of a lighting model are distributed throughout a volume; they thus represent a complex collection of surfaces with a volumetric abstraction. Kajiya and Kay also developed a Phong-like lighting model for thread; our similar approach for our thread density volume defines free-floating threads. Instead of providing parameters for lighting, we store derived values from the multivalued datasets along with tangent and density values throughout the volume.

Lengyel [Lengyel 2000] uses a volumetric texture approach to render short threads in real time that is based on a stack of partially transparent 2D texture layers. In a preprocessing step, procedurally defined threads are filtered into layers and blended together. By contrast, our data-defined threads remain individually distinguishable. In Lengyel's approach, lighting calculations are performed at run time using Banks' [Banks 1994] hair-lighting model. We use the same lighting model with a different implementation appropriate for volume rendering.

3 A Layered Volume-Rendering Framework

Our visualization framework has four steps. We begin with primary multivalued volumetric data from two sources: the brain imagery datasets were acquired experimentally using DTI and the flow datasets were created using direct numerical simulation.

3.1 Calculate Derived Datasets

Since the primary data is often difficult to interpret directly, our first step is to calculate derived volumes of data with more intuitive interpretations. For example, DTI datasets are second-order tensor fields. It is often useful to decompose these into several scalar and vector fields to reduce the problem of rendering a tensor field into one of rendering several simpler fields. Speed, derived from velocity, is such a derived quantity. Some derived quantities are not simpler, but instead provide a different view of a dataset. Vorticity, a vector derived from velocity, is an example; it captures the rotational component of velocity.

3.2 Define Visual Abstractions

In the abstraction step, we group the data volumes into layers of volume-based visual representations. While some layers consist of a direct mapping of derived values to a volume, others consist of a calculated visual effect that helps portray the underlying data; the effect is filtered or scan-converted into a volume. The visual abstractions chosen depend on characteristics of the derived data and the problem at hand. For example, neural connectivity can be portrayed with a set of thread-like fibers derived from the data and then scan-converted into a volume. Anatomical references, such as the skull or eyes, are another example; they can help give context to a visualization.

Our layered approach provides flexibility in rendering and manipulating multivalued datasets. However, simultaneously visualizing two or more layers requires devising representations that complement one another. Even elegantly structured representations can be difficult to comprehend: they may be too densely populated to show regions of interest or may lack adequate depth cues. Designing good abstractions or visual effects and good ways to combine them is a difficult but essential part of the visualization process.

3.3 Map Data with Interactive Transfer Functions

The mapping step defines transfer functions that assign color an opacity to data values and shader programs that control lighting parameters. Color and opacity mappings can be defined independently from the lighting model for each layer. The mapping step allows us to decouple the data values from their visual representation, so that we can mainpulate the visual charateristics of the data without changing the data

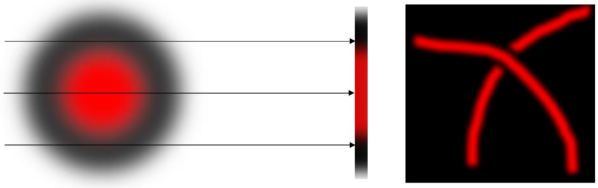

Figure 4: At left, the virtual light that passes through the red thread and its dark halo travels a long distance through the dark cloud of the halo towards the silhouette and a shorter distance through the thread itself. At right, using the same color for the halo as for the background generates a gap.

itself.

Our framework currently provides three types of transfer functions: 1D, 2D, and 2D barycentric. The 1D transfer function takes a scalar data value and returns a color and opacity. Both the 2D and 2D barycentric transfer functions take two input values and return a color and opacity.

3.4 Visualize and Explore

In the final step of the framework, we render the multiple volumes and use interaction widgets to control the visual appearance of each layer. Our texture-based volume renderer allows our system to run interactively on a PC with a commodity graphics card.

Our system uses view-aligned slices through a 3D texture. We need multiple rendering passes in order to draw all the volumetric layers. Thus, as we render the volume, we iterate through the slices from farthest to nearest and execute multiple shader programs per slice, one for each volumetric layer.

We use the mouse and 2D widgets placed in screen space to manipulate visual attributes of each layer while exploring the dataset. The widgets are linked directly to the transfer functions defined in the mapping step.

4 Threads and Halos

Inspired by hair-rendering techniques such as Kajiya and Kay's texels [Kajiya and Kay 1989] and Lengyel's shell textures [Lengyel 2000], we represent continuous directional information using thread (see Fig. 2

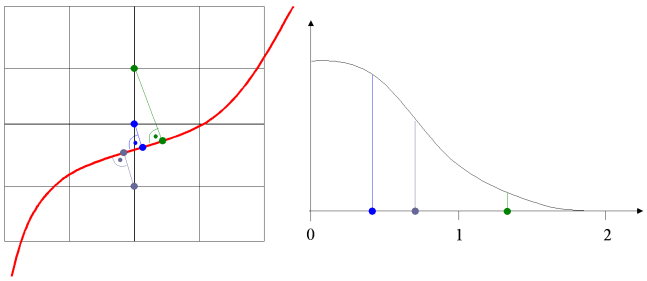

Figure 5: Filtering a path into a volume (2D view). For each voxel within a radius of two voxels from a path, we use the shortest distance to the path as the input to the filter (at right). The grid on the left shows single-voxel spacing, as does the horizontal axis on the right.

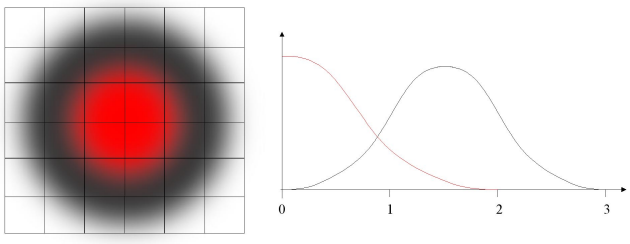

Figure 6: Filtering a halo into a volume (2D view); red depicts the thread and black the halo around it. The red curve is the filter for the thread and the black curve is the filter with which the halo is generated.

above). In general, we densely populate the volume with paths so as to represent as much of the underlying data as possible. The choice of a dense and representative set of paths displays the data faithfully and completely. To clarify individual paths, we visually augment each path with a "halo"; our interaction mode called exploratory culling selectively displays paths meeting certain criteria. Figure 3 above shows a thread density volume with and without halos: without the halos, it is particularly difficult to identify individual threads. Figure 4 shows the effect of a halo in clarifying the depth relationship.

The thread and halo volumes are precalculated and each thread in the volume corresponds to one path. Note that since the paths are represented in a volume, the rendering time is independent of the number of paths displayed. However, the diameter of the paths is limited by the resolution of the volume.

The paths and halos are filtered into volumes using a cubic B-spline filter that has a support of four voxels for the thread (see Fig. 5) and six voxels for the halos (see Fig. 6).

Lighting for the thread is calculated using the model in [Banks 1994], which defines intensity I as

$$I = k_d I_t \left(\sqrt{1-(T\cdot L)^2}\right)^p + k_s \left(\sqrt{1-(T\cdot H)^2}\right)^n. \quad (1)$$

Here I_t is the combined output color from the transfer functions, N the normal, T the tangent, L the light vector, H the vector halfway between the light vector and the view vector, and p the excess-brightness diffuse exponent.

5 Layering Volumes

Our volume-renderer implementation uses a single stack of view-aligned texture-mapped slices, rendered back to front. Lighting for the thread and halo layers is computed as in Eq.(1). For all other volume layers, a Phong lighting model (Eq.(2)) is used. As in the thread lighting model, I_t is the combined output color from the transfer functions, N is the normal, L the light vector, H the halfway vector, k_a the ambient contribution, k_d the diffuse contribution, and n the specular exponent:

$$I = k_a I_t + k_d I_t (N\cdot L) + k_s (N\cdot H)^n \quad (2)$$

Our implementation of texture-based volume rendering is embedded into a rendering framework built on top of the VR library, VR Juggler [Bierbaum et al. 2001]. VR Juggler provides a coherent application development environment for this and other related interactive virtual-reality applications. All our datasets are mapped to OpenGL 3D textures that have a resolution of 256^3 samples. We run the volume renderer on PCs equipped with either a NVidia GeForce4 Ti 4600 or a NVidia Quadro4 900 XGL. Both graphics cards have 128MB of texture memory, support 3D textures and texture compression, and provide hardware for programmable texture blending. Rendering is performed using OpenGL with several extensions, including NVidia's texture shader and register combiner and the generic texture-compression extension ARB_texture_compression. We also use NVidia's texture-shader and register-combiner specification tool, NVParse, which lets us quickly change the shader programs for the different layers without recompiling the volume renderer. This tool has been particularly helpful in iterating over the visual designs for the

Figure 7: A sequence of renderings of a thread density volume with increasing length threshold from left to right. The rightmost image shows only long paths.

example applications in this paper. More details on the implementation of the layered volume renderer appear in the Appendix.

Each slice is rendered multiple times, once for each volume layer. Since each layer-rendering pass for a slice can obscure parts of the previous one, the passes must be ordered carefully to ensure that the layers work together. We render the direct-volume-rendered layers first, the halos second, and the thread third. The direct-volume-rendered layers tend to reveal large structures and surfaces that are easy to make out underneath the more finely detailed thread and halo layers. Rendering the thread layer last keeps the threads slightly brighter.

6 Exploratory Culling

Interactively culling portions of the data using data-derived criteria has proven a powerful data-exploration tool in removing excessive detail and exposing structures of interest. By labeling structures in the visual representation layers and assigning those labels to transfer functions, we can control the color and opacity of entire structures. This approach is particularly useful for the thread paths. It is often unclear beforehand how many paths to generate for the thread density volume, as there is a tradeoff between generating too few paths to accurately represent data features and generating so many paths that the volumes become cluttered. By labeling each thread path with a parameter, e.g., path length, and selecting with a transfer function criteria based on these features, we can interactively cull out paths that meet certain criteria.

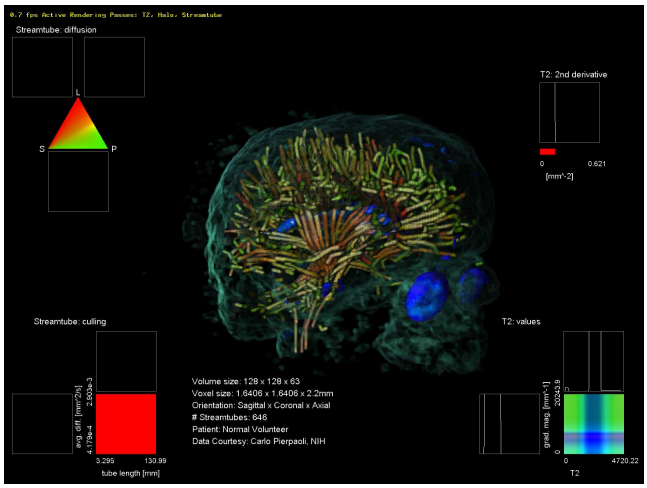

Figure 8: The interactive exploration tool. Clockwise from upper left are a 2D barycentric widget, a 1D widget, a 2D Cartesian widget, and a 2D Cartesian culling widget. The data are from a 3D second-order tensor field and a 3D scalar field acquired with MRI. Colored threads represent the tensor; the scalar is shown in a direct-volume-rendered layer.

In Zhang *et al*'s [Zhang et al. in review] approach, changing culling parameters required an entire preprocessing step taking minutes or even hours. With the exploratory culling mechanism, we can change those parameters interactively and better understand which parameter values are more effective. Figure 7 shows the effect of one of those parameters on a visualization.

7 Interactive Manipulation

We provide several on-screen widgets to control transfer functions and let us change the volume's appearance in real time. Figure 8 shows the interactive application. Each of the widgets defines a texture that corresponds to an underlying transfer function of the form 1D, 2D, or 2D barycentric. For all of the transfer-function manipulation widgets, color is HSV α. In all cases, color is manipulated along a 1D axis. For the transfer functions of dimension higher than one, two or three 1D color manipulators are used to specify colors across all the axes.

The 1D manipulation widget directly specifies the colors and opacities over the entire 1D domain. The 2D manipulation widget specifies color and opacity along each of the two axes (see Fig. 9 left). The colors

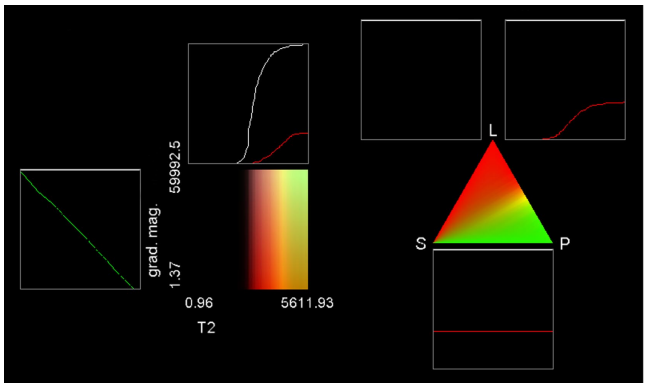

Figure 9: Two 2D transfer function manipulation widgets. On the left, the tensor-product widget combines the output from two 1D manipulators. On the right, the barycentric widget combines the output from three, one for each edge of the triangle. Color is specified in HSV α. The red curve shows hue, green saturation, blue value, and white α.

for the 2D domain are generated by averaging the colors of the two axes, while the opacities are combined multiplicatively. The 2D barycentric manipulation widget defines colors and opacities over a triangular space. We use this for the brain-diffusion-imaging examples because it manipulates the anisotropy metrics naturally. In this application, the vertices of the barycentric space represent spherical diffusion *S*, planar diffusion *P* and linear diffusion *L*. The user manipulates the color along each of the three edges (see Fig. 9 right). The color and opacity over the domain are generated by the weighted average of the colors and opacities on the three edges.

Typical rendering rates are four to five frames per second. Clipping planes help isolate regions of interest and increase the frame rate so that zooming in on those sections does not significantly slow down rendering. A high-resolution screen-capture feature lets scientists quickly navigate to interesting data views and capture high-resolution images of them with just a few extra seconds of waiting. (Features in high-resolution images can sometimes be more apparent than in the interactive application.)

8 Neuroimaging Results and Discussion

Our first scientific application involves using neuroimaging data to understand the brain. Our data are acquired using magnetic resonance imaging (MRI) and are of two types: second-order tensor-valued

water-diffusion-rate images and scalar-valued anatomical images. The scalar-valued data are typical T2-weighted MR images.

At each point in a volume, the tensor-valued data capture the rate at which water is diffusing through tissues. That rate is different in different areas – in regions of pure fluid, it is fast; in tissues like bone, it is slow. The rate of diffusion can also be directionally dependent, particularly in fibrous tissues like axon tracts and muscles, diffusing more quickly along the fibers than across them.

An ellipsoid is an effective visual representation for one diffusion-rate measurement. Its shape is analogous to that of a small ink spot that has diffused in water for a fixed time: fast, isotropic diffusion yields a big sphere; anisotropic diffusion, faster in one direction, yields a cigar-shaped ellipsoid aligned with that direction.

These diffusion-rate measurements provide a new view into the brain. Their correlation with white-matter structures is particularly intriguing to neurologists. This directionally dependent diffusion-rate information thus can potentially give neurologists insight into how different parts of the brain are connected, make possible a better understanding of neuropathology, and permit better treatment planning for neurosurgery [Zhang et al. 2001].

8.1 Primary and Derived Neuroimaging Data

The first primary dataset is a second-order tensor field measuring the water diffusion rate. Each value D is a symmetric tensor with real, non-negative eigenvalues. From D we derive several other measures. First, three scalar anisotropy measures introduced by Westin [Westin et al. 1997], c_l, c_p, and c_s, describe how close to a line, a plane, or a sphere the corresponding ellipsoid shape is for a given measurement. These measures are all positive and sum to one. Second, the trace of D, $\mathrm{Tr}(D)$, is equivalent to the sum of the eigenvalues of D and gives a scalar measure of the overall diffusion rate. Third, the gradient of the trace, $\nabla\mathrm{Tr}(D)$ and the magnitude of that gradient, $|\nabla\mathrm{Tr}(D)|$, describe how the diffusion rate is changing and in what direction; we use these quantities in lighting calculations.

The fourth and final derived data are a set of paths through the tensor field that represent the directions of diffusion. These paths are calculated and distributed within the volume as described by Zhang et al [Zhang et al. 2000; Zhang et al. 2001; Zhang et al. in review] and follow the direction of fastest diffusion in linear

regions. In planar regions, they stay within the plane formed by the major and medium eigenvectors, following whichever is more consistent with the path to that point. They are not present in isotropic regions.

The second primary dataset is a T2-weighted image scalar field showing anatomy. From it we derive the gradient of the value and the gradient magnitude, which help define how fast the value is changing and in which directions; once again, we use these quantities in lighting calculations. We also derive the second directional derivative to help define boundaries between homogeneous regions.

8.2 Neuroimaging Examples

In Fig. 1 above, data from a subject with a tumor is rendered so that the tumor is shown as an opaque mass surrounded by paths showing the diffusion structure. The color of the paths shows the type of diffusion: linear is red, planar is green.

In Fig. 8 above a diffusion dataset from a normal volunteer is rendered using three layers. The direction of fastest diffusion is shown via thread-like tubes. Traditional direct volume rendering provides a semi-transparent view of the T2-weighted volume in some regions and a relatively opaque portion in fluid-filled regions, e.g., the interior of the eyes and ventricles deep in the brain.

Figure 10 shows the mapping from the scalar- and tensor-valued volumes onto a direct-volume-rendered layer, a thread layer, and a halo layer. The first layer directly renders the T2-weighted image to give some anatomical context for the rest of the features. The hyper-intense fluid-filled regions were selected by interactively editing three transfer functions that are combined multiplicatively to give an α value. The transfer functions take as input three scalars: the image value, the magnitude of its gradient, and a directional second derivative (to give better control over the display of anatomical boundaries). The ventricles, a fluid-filled region in the center of the brain, have turned out to be an excellent landmark for neuroscientists studying these datasets. Color for this layer is specified through a transfer function based on the anisotropy metrics: isotropic regions are blue, regions with linear anisotropy red, and regions of planar anisotropy green. Lighting calculations are done using α and the gradient of the image value as a normal vector (see Eq. (2)).

The second layer renders a thread texture. The visible portions were interactively selected via an α-

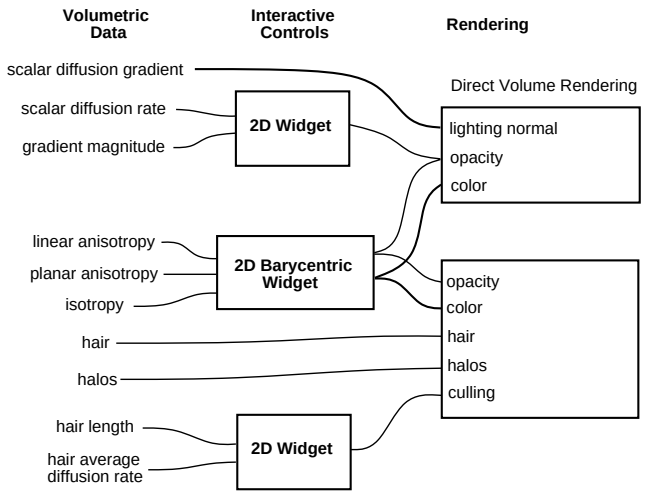

Figure 10: A data-flow diagram of the rendering setup for Fig. 8. Inputs are a tensor-valued diffusion rate image from which the volumes on the left are calculated. The interactive widgets in the center control how the quantities are displayed: some quantities are used to color layers, some to cull portions, some to show different types of diffusion, etc. The data, interactive controls, and layering are analogous for other examples.

value calculation based on three criteria. Each criterion is the basis for an α value; the three α values are combined multiplicatively. First, a transfer function maps the anisotropy metrics to α. For this rendering, areas of relatively high anisotropy are shown. Second, each path can be selected based on its length and on the average diffusion rate along it. In this rendering all paths are shown. Third, the thread density is directly provided as α. Lighting calculations are done using the α value created from the anisotropy metrics and the tangent of the thread density volume (see Eq. (1)). The third layer renders halos for the thread texture. α is calculated as for the thread except that the halo density volume is used in place of the thread density volume.

Figure 7 above shows how different length paths can be selectively displayed. All paths are present on the left, short paths removed in the center, and short and medium paths removed on the right. As the short paths are removed, it becomes easier to see some of the large white-matter structures, e.g., the internal capsule and its projection through the corona radiata. Controlling this parameter interactively has made it easier to build an overall understanding of the large structures, the more detailed small structures, and how

they interrelate.

In Fig. 2 above the left image shows a direct volume rendering with color coding the type of anisotropy (red for linear and green for planar) and opacity coding for the diffusion magnitude. The right image uses the same color and alpha information, but renders it using a thread layer that shows the direction of fastest diffusion; this additional information reveals much more of the neural structure. The figure shows a dataset from a patient with a tumor. The left image isolates the tumor in the direct-volume-rendered layer, the right image shows the thread and halo layers, and the center image combines both, showing a cradle of planar anisotropy around the tumor. Our visualization work has suggested several hypotheses for how tumor growth may create this previously unseen phenomenon.

9 Simulated Blood Flow in a Bifurcated Artery Model

Our second scientific application involves simulated fluid flow data on incompressible flow through a simplified model of a bifurcating coronary artery. We render one time-step of a pulsatile flow.

Biologically, we have been studying how the flow structure is related to atherosclerotic lesion formation. Lesions tend to form opposite branches in arteries. We hypothesize that upstream flow structure may provide important insight into why this happens. Exploration of the flow structure is facilitated by a visual representation displaying as much of the data as possible. Our volume visualization is particularly useful in this regard both because it displays multivalued volume-filling data in each image and because it is possible to interact with the 3D image and change how the different values are displayed.

9.1 Primary and Derived Fluid Flow Data

The primary data for this application area is a 3D velocity vector field. From this, a number of quantities can be derived. Speed, a scalar field, is the velocity's magnitude. Vorticity, a vector field, is a component of the first derivative of the velocity and captures the local rotational component of the flow. The vorticity vector points along the axis of rotation and its magnitude indicates the rate of rotation.

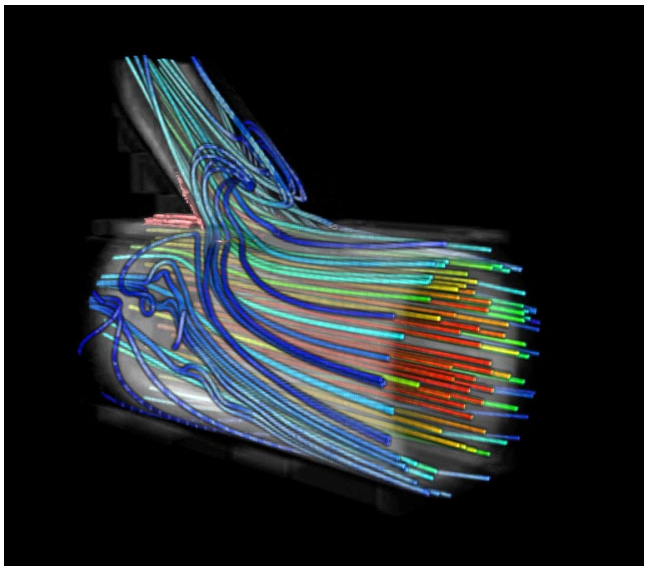

Figure 11: Simulated flow, from right to left, through a model of a branching coronary artery. Several complex structures can be seen, including reversal of flow direction, as illustrated by the blue-haloed hair streamlines in the side branch immediately downstream from the bifurcation. The semi-transparent white shell represents vorticity magnitude and gives near-the-wall context for the streamlines.

9.2 Flow Examples

Figure 11 shows an idealized model of a branching coronary artery. Flow is from right to left, starting in the open end and proceeding down into the two branches. Haloed hair streamlines colored according to speed are rendered together with a diaphanous shell showing relatively low-vorticity regions. A more opaque pink section right at the point of bifurcation shows the region of highest vorticity.

The same flow is rendered in Fig. 12. In this image yellow hairs show vortex lines integrated through the vorticity vector field. The semitransparent purple form shows low-speed regions.

These two images together show important flow features not seen with other visualization methods, including near-wall kinks in vortex lines and localized looping structure in the vorticity. The kinks tended roughly to fit into the upstream edges of separation pockets evident in velocity images. Velocity and vorticity lines are layered together in Fig. 13 (the input data was cropped to $128 \times 256 \times 256$ because of limited texture memory). Visualizing the vector fields simultaneously clarifies correlations among features quite dramatically.

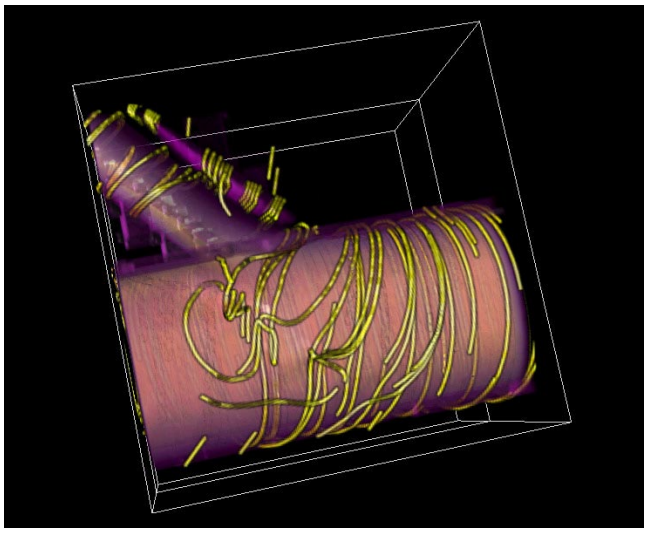

Figure 12: Integral curves through the vorticity vector field for the flow illustrated in Fig. 11. These vortex lines give additional clues to the flow structure, particularly in areas where it curves and varies in speed.

10 General Discussion

10.1 Exploratory Culling

Exploratory culling enables users of this system quickly to generate a broad spectrum of images from the same dataset. The three dramatically different views of the neuroimaging dataset in Fig. 7 above were created by culling paths by length. This has proven an important task in our example applications. Long paths often illustrate the major structures in a DTI dataset, and smoothly transitioning between a view of just long paths to a view including all paths has helped us contextualize and understand features in the data. Culling paths according to the average diffusion rate along each has also proven useful in helping distinguish the most coherent paths. Since the culling operations work at interactive rates, they facilitate quick exploratory experimentation with this relatively new kind of data.

Exploratory culling is implemented as a transfer function controlled by a manipulation widget, just like the other transfer functions in the system. However, unlike the other transfer functions, reasonable values for the α of the culling transfer function seem generally to be restricted to zero or one; intermediate values do not usually give useful results. Thus, it may be possible to reduce the culling transfer function to a single bit and still maintain most of its utility, a change that may simplify the implementation or let us

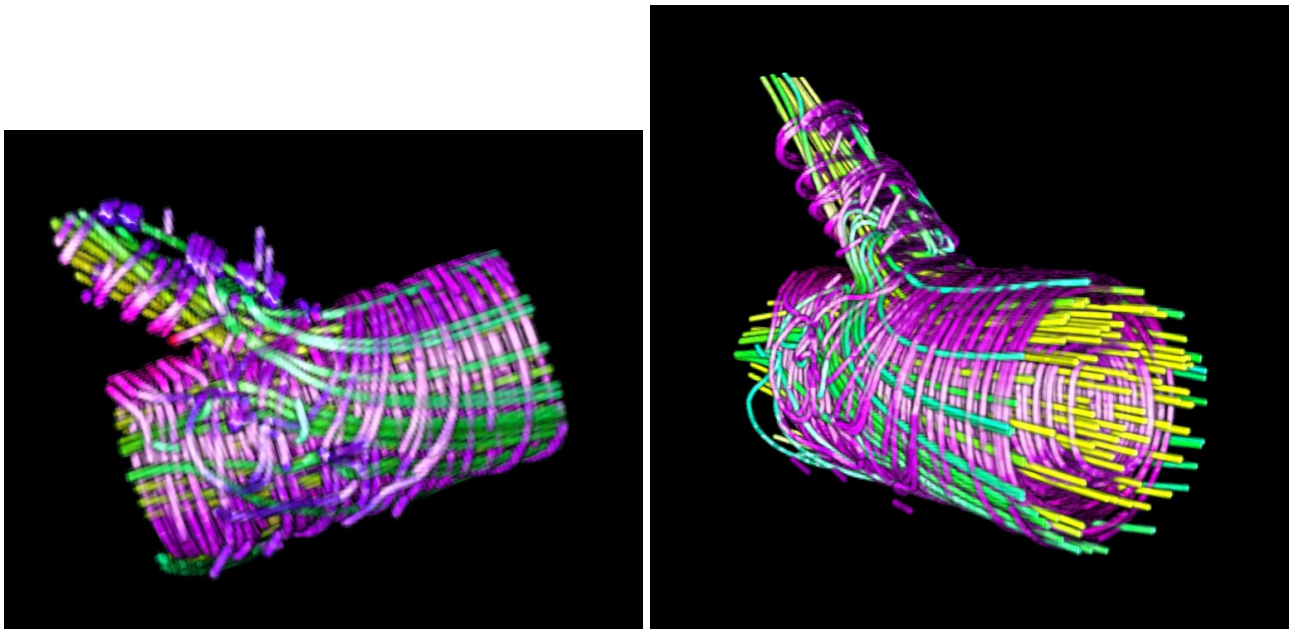

Figure 13: Integral curves through both the velocity (yellow and green) and vorticity (purple and pink) vector fields for the same flow as illustrated in Figure 11. Correlations among these vector fields are typically representative of important flow structures.

implement additional functionality using the additional bits.

Culling suffers from resolution limitations, a problem arising when paths are very closely packed within the volume. If a given voxel contains contributions from multiple paths, only one of them can be culled correctly. This typically happens in the periphery of two paths.

10.2 2D Transfer Functions

The 2D transfer-function editing we have implemented is a simple blend of two 1D transfer functions. More sophisticated transfer-function editing will give more control over the resulting images and is likely to generate better understanding of the data. We would like to be able to separate out regions of differing anisotropy more precisely or use the derivative information to display tissue boundaries more accurately.

10.3 Avoiding Thresholding

Thresholds in visualization are often problematic because they introduce apparent boundaries where none exist. In DTI visualization, the mapping of an anisotropy metric to a binary region of anisotropy introduces such a boundary, as evident in earlier DTI visualization work. We avoid these boundaries here by making smooth transitions in our transfer functions whenever possible. But some boundaries, e.g., those generated by culling, are difficult to avoid.

10.4 Threads and Halos

We have implemented several different thread lighting models, Kajiya and Kay [Kajiya and Kay 1989], and Banks [Banks 1994] with excess brightness exponents of $p = 2$ and $p = 4$. The actual value of p introduced by Banks is around 4.8. Kajiya and Kay's lighting model is similar to Banks' with an excess brightness exponent of $p = 1$. The smaller the exponent, the brighter the thread and the less dramatic the lighting. We found that the lighting model that uses $p = 2$ worked well: this brightens the threads and compensates for the overall darkening effect of the halos. Another advantage of using $p = 2$ is that it requires no square-root calculation, which significantly speeds up processing.

One of the drawbacks of using a volume to represent the threads is the limited resolution. This restricts the diameter and density of the thread paths that can be rendered without intersection, and also reduces our ability to store normal information for hair strands.

Our halo implementation darkens the threads significantly along the silhouettes; however, it also introduces some extraneous darkening along the thread centers, since it does not take the viewing direction into account. We experimented with halos that used outward-pointing normals from the thread paths to restrict the darkening to silhouettes. Unfortunately, the very limited spatial resolution (only a few voxels around the cross section of a halo) made this approach impossible, since the linear interpolation of the components of the normal was too inaccurate.

Shadows would improve visual perception of the thread volumes and thus offer an alternative to halos. Unfortunately, they are very expensive to calculate and still would not always make possible a clear perception of the depth ordering.

10.5 Design

When rendering layers of volumes, it is important to design the layers to complement one another visually. Just as the layers of a painting work together to convey a coherent theme, volume layers must be carefully designed so that they function together: each one must be designed with a sense of its impact on the appearance and effectiveness of the others.

Figure 2 shows a deconstructed layered volume. On the left is a direct-volume-rendered layer for a brain visualization and on the right is a complementary thread layer. The center image shows the composite rendering of both layers. In order to produce a composite rendering as effective as this one, care must be taken throughout the design process to insure that the final rendering has an effective composition and balance. At the abstraction level of design, the thread-and-halo effect was carefully constructed to represent directional information but avoid overwhelming the image, so that a direct-volume-rendered layer would not be obstructed. At the interactive level of design, effective composition and balance are achieved through manipulating transfer function widgets. Careful use of color, transparency, and culling is essential in preparing a view of the data that illustrates or locates a particular trend or anomaly. For example, a transfer function for a direct-volume-rendered layer that uses bright orange to indicate tumorous regions in the brain should not be used in conjunction with a transfer function that uses bright orange in a thread layer to represent paths typically found around a tumor – the two layers will be too difficult to distinguish. In general, transfer functions that work well in representing a single variable do not necessarily work without modification within a layered rendering approach. Thus, the visual design aspect of scientific visualization, with special emphasis on color and balanced composition, is particularly important within this framework.

10.6 Hardware Observations

We constantly work against the limits of texture memory and eagerly await new hardware. In the meantime, texture compression has been very helpful. We also found that the approaches we are interested in require more hardware arithmetic support. We anticipate that nine combiner stages are likely to be sufficient for these new approaches.

11 Summary and Conclusion

We have presented a volume-rendering approach for visualizing multivalued datasets. Our framework has four steps: calculating derived datasets, defining visual abstractions, mapping the datasets through interactive transfer functions, and exploring interactively. A key component of our approach is the abstraction of the derived values into separate layers of visual representations. This allows us to provide interactive manipulation of the visual attributes of each volume layer.

We also introduced thread and halo density volumes as an example of complementary volumes. Together, they provide an interactive visual representation of continuous directional information.

Labeling the thread volume according to some culling criterion, e.g., path length, allows us to control culling interactively. By manipulating a transfer function we can remove portions of a volume to expose interesting structures that might otherwise be occluded. Users can quickly generate many different images of multivalued datasets using this mechanism. Thus, scientists can quickly explore large regions of a volumetric dataset and, through this exploration, arrive at a better overall understanding of complex multivalued data.

We also introduced a technique for rendering visibility occluding halos within a volume in real time. These halos increase the depth perception of our thread path representations and help achieve the 3D understanding that is so crucial when examining volumetric data.

We showed that we can interactively render and manipulate multivalued volumetric datasets on PCs equipped with commodity graphics cards. We implemented a texture-based multi-pass volume renderer that composites the different layers together in a final image. To visualize and interactively manipulate the individual layers we rely heavily on programmable texture blending.

Additionally, we showed how two or more complementary visual representation layers, for example the direct-volume-rendered DTI and the thread paths, can be used to effectively explore multivalued volumetric datasets. The anatomical context provided by multiple layers enhances understanding of the data. Understanding is also achieved through interaction; providing manipulation widgets for each of the layers makes possible exploration of the individual layers and, more importantly, of the relationships among the data values of the individual layers.

Our layered complementary volumes approach was successful in both application areas we explored. For the visualization of diffusion characteristics of the human brain, we can represent both planar and linear anisotropy at the same time as well as smooth transitions between the two. We also provide an anatomical reference for the data that is essential in contextualizing the diffusion information.

For the exploration of blood flow in a bifurcated artery, our layered volumes approach was successful in representing anatomical references. The thread paths and halos representation was particularly useful in representing streamlines and vorticity lines. Correlating this layer of representation with a direct-volume-rendered layer helped locate regions of vorticity change and other interesting flow features. As in the brain diffusion application, exploratory culling and the interactive manipulation widgets were instrumental in quickly creating many different images of the data, leading to a better understanding of the 3D flow.

Feedback from fluids researchers and neuroimaging researchers suggests that they will find our interactive volume renderer effective in exploring both kinds of multivalued datasets. Currently these data are often not well understood and exploring them will help develop and evaluate new scientific hypotheses about the data and the physical phenomena they represent.

12 Acknowledgments

The authors thank the Brown Scientific Visualization Group, the Brown Graphics Group, and the Brown Technology Center for Advanced Scientific Computing and Visualization. Special thanks go to George Karniadakis and his group at Brown, Peter Richardson at Brown, Susumu Mori from Johns Hopkins, Mark Bastin from the University of Edinburgh, and Thomas Diesboeck from MGH for their data and feedback. Thanks also to Katrina Avery and Moriah Horani for superb editorial input. This work was partially supported by NSF (CCR-0093238) and the Human Brain Project (NIDA and NIMH).

T2-weighted MRI	
gradient x	R
gradient y	G
gradient z	B
gradient magnitude	R
2nd directional derivative	G
T2 value	A
Direct-volume-rendered DTI	
gradient x	R
gradient y	G
gradient z	B
gradient magnitude	R
c_l	G
c_p	B
diffusion magnitude	A
Streamtubes	
tangent x	R
tangent y	G
tangent z	B
density	A
average diffusion	R
c_l	G
c_p	B
length	A
Direct-volume-rendered fluid flow	
Halos	
density	A

Table 1: Mapping from data values onto texture channels.

A Implementation Specifics

A.1 Mapping Visual Representations into 3D Textures

To make volumetric data available for hardware-accelerated rendering it must be packaged into OpenGL 3D textures. Even though textures normally consist of the three colors RGB and sometimes an additional transparency or alpha value A, we need not store color information in the four channels. In fact, we use the four channels to store the data values from our visual representations. We are restricted to eight bits per channel due to texture-memory limitations on commodity PC graphics cards. Therefore, the data must be quantized to fit in the range $0 \ldots 255$. Table 1 shows how the data values from our examples are mapped into the RGBA channels.

A.2 Compression of Volumetric Datasets

The volumetric data is stored in compressed OpenGL 3D textures. For a 256^3 RGBA texture with eight bits per channel the data is 64MB. A GeForce 4 has available 128MB of texture memory that must be shared with the frame buffer, the textures for the transfer functions, and display lists. It would thus be impossible to show more than one layer at a resolution of 256^3 without some type of data compression. Using the generic OpenGL texture compression extension ARB_texture_compression provides a 4:1 compression ratio, which reduces the 64MB to 16MB for a single 256^3 3D texture. This improved memory efficiency allows us to store multiple layers in texture memory, which is imperative for interactive rendering.

A.3 Texture-Shader Program

The texture-shader program specifies the textures to fetch, how to fetch them, and the assignment of the interpolated fragments to registers. Four textures and fetched and assigned to registers `tex0` to `tex3`.

In order to achieve interactive frame rates, manipulating large, compressed 3D textures must be avoided. Thus, we do all our texture manipulation with transfer function textures. We rely heavily on the dependent texture lookup capabilities of modern graphics cards, i.e. the capability to use the output of one texture lookup as the coordinates to another texture lookup. The following texture-shader program in the NVParse

format is typical for our system.

```
!!TS1.0
texture_3d(); // load texture 0
texture_3d(); // load texture 1
dependent_gb(tex1);
dependent_ar(tex1);
```

First we fetch two 3D textures and assign the interpolated fragment to the registers `tex0` and `tex1`. Then we perform two dependent-texture lookups, the first based on the green and blue channels of register `tex1` and the second based on the alpha and red channels of register `tex1`. As an example, the green and blue channels of the 3D texture assigned to the register `tex1` might hold the two diffusion anisotropy metrics. We use those two values as the coordinates for a texture lookup into the texture of the 2D barycentric transfer function assigned to register `tex2`. The output is a color and opacity for the current fragment based on diffusion anisotropy.

A.4 Register-Combiner Program

The register-combiner program defines how textures are blended together. The register-combiner stage on a GeForce3 or higher provides eight general combiners and one final combiner to calculate the final fragment color from the data in the register set. For each combiner we can define input mappings on the input registers and scale and bias on the output registers. The combiner itself lets us perform simple operations such as dot product, elementwise multiplication, and sum.

The volume renderer performs texture blending as well as lighting calculations using the register combiners. We have implemented two lighting models in the register combiners, one based on Eq. (1) and one based on Eq. (2). The light vector L and halfway vector H needed in the lighting calculations are passed in through the constant registers `const0` and `const1` of the first general combiner. These two registers must be processed in the first combiner if we want to use the light and halfway vector in any of our calculations. This is the only convention necessary for our shader programs and introduces no major restrictions.

A.5 Compositing Volume Layers

The pseudocode below describes the rendering process.

```
for all slices back to front
    for all layers/shader programs
        activate the appropriate texture units
        bind textures
        bind texture shader program
        bind register combiner program
        set current light and half-way vector
        push texture transform on the texture stack
        render the slice
    end
end
```

It is particularly important that the pseudocode be executed on an ordered sequence of view aligned slices. The slices must be arranged in order from back to front. When the blending function is set to `GL_SRC_ALPHA` and `GL_ONE_MINUS_SRC_ALPHA`, this sequence produces the desired rendering containing information from all layers.

References

BANKS, D. C. 1994. Illumination in Diverse Codimensions. In *Proceedings SIGGRAPH '94*, ACM, 327–334.

BASSER, P., PAJEVIC, S., PEIRPAOLI, C., DUDA, J., AND ALDROUBI, A. 2000. In Vivo Fiber Tractography Using DT-MRI Data. *Magnetic Resonance in Medicine 44*, 625–632.

BIERBAUM, A., JUST, C., HARTLING, P., MEINERT, K., BAKER, A., AND CRUZ-NEIRA, C. 2001. VR Juggler: A Virtual Platform for Virtual Reality Application Development. In *Proceedings of IEEE VR 2001*.

BROOKS, F. P. 1996. The computer scientist as toolsmith [ii]. *CACM 39*, 3 (March), 61–68.

CABRAL, B., CAM, N., AND FORAN, J. 1994. Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. In *ACM Symposium On Volume Visualization*, ACM.

ENGEL, K., KRAUS, M., AND ERTL, T. 2001. High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading. In *Siggraph/EurographicsWorkshop on Graphics Hardware 2001*, ACM.

INTERRANTE, V., AND GROSCH, C. 1997. Strategies for Effectively Visualizing 3D Flow with Volume LIC. In *Proceedings of the conference on Visualization '97*, 421.

KAJIYA, J. T., AND KAY, T. L. 1989. Rendering Fur with Three Dimensional Textures. In *Proceedings SIGGRAPH '89*, ACM, 271–280.

KINDLMANN, G., AND WEINSTEIN, D. 1999. Hue-Balls and Lit-Tensors for Direct Volume Rendering of Diffusion Tensor Fields. In *Proceedings IEEE Visualization '99*, IEEE, 183–189.

KIRBY, R. M., MARMANIS, H., AND LAIDLAW, D. H. 1999. Visualizing multivalued data from 2D incompressible flows using concepts from painting. In *IEEE Visualization '99*, D. Ebert, M. Gross, and B. Hamann, Eds., 333–340.

KNISS, J., KINDLMANN, G., AND HANSEN, C. 2001. Interactive Volume Rendering Using Multi-Dimensional Transfer Functions and Direct Manipulation Widgets. In *IEEE Visualization 2001*, IEEE.

LAIDLAW, D. H., AHRENS, E. T., KREMERS, D., AVALOS, M. J., JACOBS, R. E., AND READHEAD, C. 1998. Visualizing Diffusion Tensor Images of the Mouse Spinal Cord. In *Proceedings IEEE Visualization '98*, IEEE, 127–134.

LAIDLAW, D. H., AHRENS, E. T., KREMERS, D., AVALOS, M. J., READHEAD, C., AND JACOBS, R. E. 1998. Visualizing diffusion tensor images of the mouse spinal cord. In *Proceedings Visualization '98*, IEEE Computer Society Press.

LAIDLAW, D. H., KREMERS, D., AHRENS, E. T., AND AVALOS, M. J. 1998. Visually representing multi-valued scientific data using concepts from oil painting. In *Visual Proc. SIGGRAPH '98*.

LENGYEL, J. E. 2000. Real-Time Fur. In *Eurographics Rendering Workshop 2000*, ACM, 243–256.

LUM, E. B., AND MA, K.-L. 2002. Hardware-Accelerated Parallel Non-Photorealistic Volume Rendering. In *Proceedings of the Non-Photorealistic Animation and Rendering Conference 2002*.

PIERPAOLI, C., AND BASSER, P. 1996. Toward a Quantitative Assessment of Diffusion Anisotropy. *Magnetic Resonance Magazine*, 893–906.

VAN GELDER, A., AND KIM, K. 1996. Direct volume rendering with shading via three-dimensional textures. In *1996 Volume Visualization Symposium*, IEEE, 23–30. ISBN 0-89791-741-3.

WESTIN, C.-F., PELED, S., GUBJARTSSON, H., KIKINIS, R., AND JOLESZ, F. 1997. Geometrical Diffusion Measures for MRI from Tensor Basis Analysis. In *Proceedings of ISMRM 1997*.

ZHANG, S., CURRY, C. T., MORRIS, D. S., AND LAIDLAW, D. H. 2000. Streamtubes and Streamsurfaces for Visualizing Diffusion Tensor MRI Volume Images. In *IEEE Visualization 2000 Work in Progress*, IEEE.

ZHANG, S., ÇAGATAY DEMIRALP, KEEFE, D., DASILVA, M., GREENBERG, B. D., BASSER, P. J., PIERPAOLI, C., CHIOCCA, E. A., DEISBOECK, T. S., AND LAIDLAW, D. 2001. An immersive virtual environment for DT-MRI volume visualization applications: a case study.

ZHANG, S., ÇAGATAY DEMIRALP, AND LAIDLAW, D. H. in review. Visualizing diffusion tensor mr images using streamtubes and streamsurfaces. *Transactions on Visualization and Computer Graphics*.

ZÖCKLER, M., STALLING, D., AND HEGE, H. 1996. Interactive visualization of 3D-Vector fields using illuminated streamlines. In *IEEE Visualization '96*, IEEE. ISBN 0-89791-864-9.