Visualization of Fibrous and Thread-like Data

Zeki Melek , *Student Member, IEEE*, David Mayerich, *Student Member, IEEE*, Cem Yuksel, and John Keyser , *Member, IEEE*

Abstract— Thread-like structures are becoming more common in modern volumetric data sets as our ability to image vascular and neural tissue at higher resolutions improves. The thread-like structures of neurons and micro-vessels pose a unique problem in visualization since they tend to be densely packed in small volumes of tissue. This makes it difficult for an observer to interpret useful patterns from the data or trace individual fibers.
In this paper we describe several methods for dealing with large amounts of thread-like data, such as data sets collected using Knife-Edge Scanning Microscopy (KESM) and Serial Block-Face Scanning Electron Microscopy (SBF-SEM). These methods allow us to collect volumetric data from embedded samples of whole-brain tissue. The neuronal and microvascular data that we acquire consists of thin, branching structures extending over very large regions. Traditional visualization schemes are not sufficient to make sense of the large, dense, complex structures encountered.
In this paper, we address three methods to allow a user to explore a fiber network effectively. We describe interactive techniques for rendering large sets of neurons using self-orienting surfaces implemented on the GPU. We also present techniques for rendering fiber networks in a way that provides useful information about flow and orientation. Third, a global illumination framework is used to create high-quality visualizations that emphasize the underlying fiber structure. Implementation details, performance, and advantages and disadvantages of each approach are discussed.

Index Terms—neuron visualization, GPU acceleration, global illumination, orientation filtering

1 Introduction

The segmentation and visualization of structures in large volumetric data sets continues to pose significant problems. New methods for data acquisition are creating data sets of increasing resolution, requiring newer algorithms to extract fine details. Recent data acquisition techniques, such as Knife-Edge Scanning Microscopy (KESM) [12] and Serial Block Face Scanning Electron Microscopy (SBF-SEM) [5], allow the sectioning and scanning of high-resolution neural and vascular data. These datasets contain dense fibrous structures that pose a unique problem in volumetric visualization. The small radius of individual fibers makes them difficult to differentiate from high-frequency noise. In these cases, image processing can be difficult because low-pass filters are a common technique used to prepare data for segmentation. Instead, vector tracing algorithms are used to locate and label fibers through a dataset. These techniques produce large numbers of line segments in three-dimensional space that represent the trajectories of individual fibers.

Interpreting volumetric data requires that the user is able to display and manipulate the data set interactively. Even for dense fibrous structures, this could be accomplished using line segments rendered on modern graphics hardware. Unfortunately, the density of neuronal fibers and microvasculature makes them difficult to visualize, resulting in an over-lapping tangle of lines that would be extremely hard for a user to interpret. Of course, one could always display randomly sampled subsets of thread data, but this would cause the loss of connectivity information between fibrous units, such as neuronal synapses from brain data.

- *Zeki Melek is in Computer Science at Texas A&M University, E-mail:melekzek@tamu.edu*
- *David Mayerich is in Computer Science at Texas A&M University, E-mail:david@quantumkingdom.com*
- *Cem Yuksel is in Visualization Science at Texas A&M University, E-mail:cem@viz.tamu.edu*
- *John Keyser is in Computer Science at Texas A&M University, E-mail:keyser@cs.tamu.edu*

Manuscript received 31 March 2006; accepted 1 August 2006; posted online 6 November 2006.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

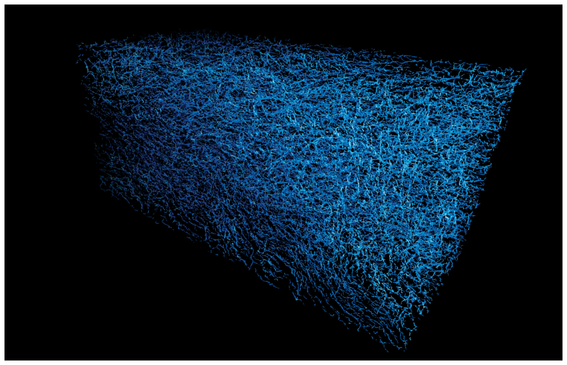
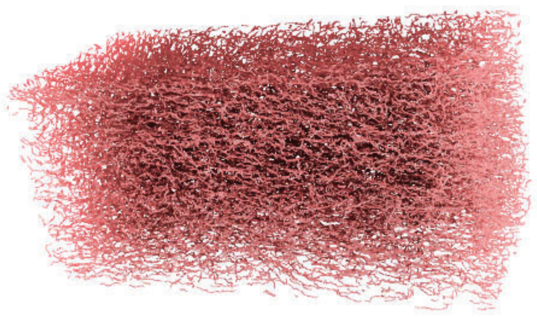

Fig. 1. Two visualizations of the neuronal threads, (top) interactive GPU SOS, and (bottom) hair-like rendering using fake skylight.

Incorporating shading information into on-screen threads allows a user to differentiate between individual fibers, but generally requires the creation of a polygonal surface. Although the thread data retrieved from vector tracing can be used to help create a polygonal isosurface, maintaining the continuity of the thread information requires a surface that tends to be extremely triangle-heavy, making it difficult to render a reasonable data set at interactive rates, even on modern graphics hardware. Furthermore, the basic shading that can be applied at interactive rates is sometimes insufficient for understanding the full complexity of the data.

1.1 Main Results

In this paper, we present a set of practical techniques that improve our ability to visualize dense thread-like structures extracted from large volumetric datasets. We present two techniques for visualizing such data interactively. First, in Section 3 we describe a way of implementing self-orienting surfaces on the GPU in order to quickly render individual thread-like paths at a minimal polygon cost. We also discuss a method for shading based on thread orientation, allowing us to visualize oriented fiber bundles that are difficult to see within the dense fiber set. Second, in Section 4 we describe a non-interactive method of rendering images of a thread-like volumetric data sets using global illumination techniques that make the trajectories of fiber bundles easier for a user to interpret. We also provide some brief background (Section 2) and an evaluation of our approach (Section 5).

2 Background

The visualization techniques we present here, though they have more general application, are geared toward a particular type of data. Note that this data is of a completely different scale and with very different characteristics than that of prior fiber visualization work (e.g. [2]).

2.1 Prior Visualization Work

Our data shares many characteristics of field lines, which have been thoroughly investigated in the flow visualization literature. Line segments are fast to display, but do not provide sufficient perceptual cues for complex geometric structures. Polygonal tubes can provide shading and radius information but become too expensive on larger datasets. Stream -polygons, -ribbons, -tubes, surfaces, and -balls have been proposed to decrease poly count while providing sufficient visual cues. A thorough discussion of these methods has been given by Schussman and Ma [16].

Schussman and Ma also present a new representation for field lines, the Scalable Self-Orienting Surface (SOS). These surfaces resemble the impostors and billboards commonly used in computer graphics, but are more flexible. SOS consists of polygon strips along a streamline, where the vertices rotate axially along a curve and always face the camera. The thickness of the strip gives a depth cue, and hardware bump mapping using normals set along the strip yields shading and specular lighting along the strip. Additional visual cues are provided using textures and per vertex fog (as provided in OpenGL extensions). SOS uses far less memory than display lists of polygon cylinders, and has been reported to work roughly 25 times faster.

An approach similar to SOS, stylized line primitives, has been implemented for vector field and flow visualization on the graphics processor unit (GPU) by Stoll et al. [17]. They provide additional visual cues by adding halo and shadow maps on the GPU. They propose a hybrid GPU-CPU approach to handle visual problems around singularity points such as silhouette vertices. A more detailed comparison to our method could be found at section 5.2. Kondratieva et al. [10] also make use of the GPU by using particle tracing to visualize 3D tensor fields. They demonstrate visualization of volumetric diffusion tensor images of human brain and canine heart.

Spline techniques [4, 7] are a well-known method for creating smooth interpolations through sampled data sets. Rossl et al. [15] use quadric super splines to interactively visualize a number of well-known volume datasets.

2.2 Data acquisition

The need for efficient methods to visualize thread-like structures follows from two recent developments in microscopy.

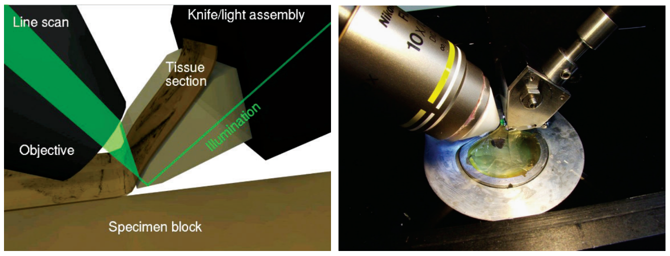

Fig. 2. Diagram of Knife Edge Scanning Microscopy, and a photo of our machine.

2.2.1 Knife Edge Scanning Microscopy

Knife-Edge Scanning Microscopy (KESM), developed at Texas A&M, involves an optical microscope/microtome designed to image thin sections of embedded tissue in order to reconstruct anatomical information at the cellular level. The use of KESM produces high-resolution data sets by imaging at 0.6μm x 0.6μm and cutting serial sections less than 1.0μm thick. Imaging and cutting are simultaneous, thereby maintaining registration between sequential sections.

The KESM consists of an optical microscope mounted perpendicular to the cutting surface of a translucent diamond knife (see figure 2). Light from a fiber-optic illuminator is refracted through the knife, which acts as both a cutting tool and high-intensity light source. Light is then transmitted through the tissue being cut and into the objective for imaging. A high-speed line scan camera mounted behind the objective images the tissue as it is being cut. The camera we are currently using operates at 45kHz, where each sample is made up of a 4096x1 pixel line. This allows us to retrieve data at a maximum rate of over 200 megabytes/second at a resolution of 0.6μm x 0.6μm x 0.8μm. KESM currently offers the only feasible method for scanning an entire mouse brain at such a resolution within a month, although our experimental datasets are not currently this large.

2.2.2 Serial Block Face Scanning Electron Microscopy

Serial Block-Face Scanning Electron Microscopy (SBF-SEM), developed by Denk and Horstmann [5], is another microscopy technique for acquiring volumetric data. In this approach, a scanning electron microscope is used to image the face of a block of prepared tissue. A microtome located within the microscope then removes a thin portion of the surface material, and the process is repeated. The result is a serial set of images of the tissue block at a much higher resolution (approximately 10nm x 10nm by 50nm) than available through light microscopy, however the slower imaging speed and staining techniques limit the size of the tissue sample. Although KESM and SBF-SEM provide different magnifications, the thread densities and dataset sizes are similar.

2.2.3 Our Data Sets

We use three different data sets as sources of our fiber-like data. From the KESM, we use two different staining techniques. With one technique (Golgi), we stain complete neurons, but only a small fraction of them. The vector tracing reconstructs the neuron structure. A different stain (Nissl) is used to stain cell bodies only (without the processes - the axons and dendrites), and thus does not produce thread-like data on its own. However, Nissl staining can be used to identify (by lack of stain) the microvasculature (small blood vessels) in the brain tissue. When vector tracing is applied, the microvasculature produces similar data to that of the Golgi stain. Finally, we use a set of neuronal data scanned by an SBF-SEM method. Tracing again follows pathways of neuron processes. As this produces the densest packing of threads, it is also the most

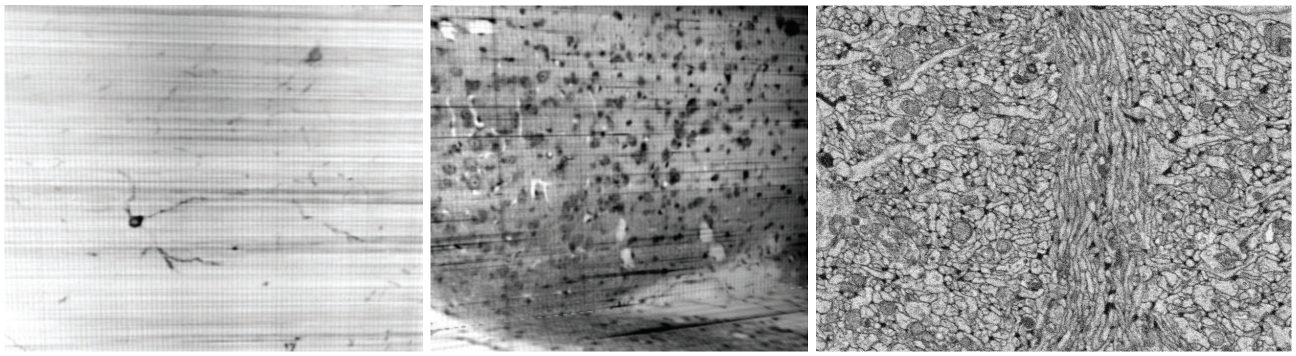

Fig. 3. From left to right, Golgi KESM data, Nissl KESM data, and SBF-SEM data.

dramatic in terms of our visualization. Examples of raw data of each of these types is shown in figure 3. The Golgi and Nissl stained volumes are 800x800x300 voxel blocks while the SBF-SEM dataset is 800x800x500 voxels. These are relatively small subsets of the available datasets but provide enough density to show the effectiveness of our visualization techniques.

2.3 Vector tracing

In order to deal with these large volumes of data, the datasets are split into smaller volumes and processed in parallel. The data we show is a collection of several such data sets processed independently.

The initial compression and segmentation of volumetric data is done using the L-block data structure proposed by McCormick et al.[13]. This technique allows us to achieve a high degree of compression by taking advantage of the thread-like nature of the structures stored in the volumetric data set. Initially, a conservative threshold is used to eliminate unstructured information that is highly unlikely to be of interest. The remaining information, including the desired threads and nearby pixels, are stored in 2x2x2 voxel axis-aligned bounding boxes. Each bounding box is connected to its neighbors on each side. Next, a greedy approach is used to combine neighboring bounding boxes to achieve the highest compression ratio for stored voxel information and connection information. This L-block data can be used as an initial approximation for the thread structures.

Extracting microvasculature information from Nissl stain and neuronal processes from Golgi is done using a modification of the vector technique originally developed by Can et al. [3] to trace blood vessels in two-dimensional images. This technique uses a correlation kernel of the form [1 2 0 -2 -1] in order to locate the boundary of a blood vessel. Two kernels can then be used to track both boundaries of a 2D fiber. By averaging across several pixels, the trajectory of the fiber structure can be estimated. As small steps are taken along the fiber, the estimated trajectory can be updated, allowing the structure to be traced. This technique can be extended into three dimensions as shown by Al-Kofahi et al. [1] for confocal image stacks.

We extend the 3D vector tracing algorithm by using additional correlation kernels to extract more accurate 3D radius information and to better track the individual thread-like paths. In addition, the algorithm requires the selection of initial points and direction vectors in order to begin tracing. These seed points are selected using the initial L-block construction as described by Doddapaneni [6]. Seed points are scattered through the structure and individually traced. As the tracing algorithm encounters other seed points, they are systematically eliminated. Once an entire thread has been traced, a new seed point is selected from the remaining set. Our implementation of the vector-tracing algorithm does not process branching or make special exceptions for cell bodies. Thus, the output of our vector tracing is a set of polylines in 3D that approximately follow the medial axis of the neuron/vascular structure, and contain associated radius information. This collection of line segments forms the “track” data our system relies on.

Since the processing of vector data just requires local information, the algorithm is highly parallel. In order to process our large data sets, the volumetric data was broken up into 100x100x100 voxel sections. The vector tracing algorithm can then be run on each section individually using separate processors. The running time of the algorithm on separate blocks varied with the number of threads but took approximately 10 minutes per 100x100x100 segment on a 1.8 GHz P4 processor. Additional stitching between inter-block segments is done as an additional postprocess.

3 Interactive Visualization

The fibrous neuronal data sets we obtain can be large and complex. Besides looking at local structures and individual threads, one also wants to explore the larger organizations that are particularly interesting because of the flow-like behavior of axonal and dendritic arbors. These arbors often appear to be tightly bundled, and understanding the “flow” of this bundle throughout the brain can potentially give improved insight into the organization of neural pathways, and thus better understanding of neural computation. We describe here a set of interactive techniques found useful for visualizing our data interactively.

3.1 Fast Visualization of Threads

We have created an efficient GPU-based implementation for displaying self orienting surfaces. SOS is a memory efficient and fast representation, where every vertex needs to be rotated axially along the curve. Traditional implementations [16] are CPU bound and vertex data needs to be transferred to the graphics card every frame after rotation to face to the camera; furthermore, such an approach does not exploit display lists. Taking advantage of the advancement of graphics hardware, we have implemented this process on the GPU using a vertex shader. This enables us to use display lists, and transfer the vertex data only once to the graphics card. Additionally, we can add additional visual cues with only a negligible performance loss without the need to retransfer the vertex data. Note that Stoll et al. have previously developed a similar GPU extension (stylized line primitives) for visualizing vector fields and flow lines [17].

We have adapted the SOS approach to be used with neurons. We start by generating a degenerated triangle strip along the thread, allocating two vertices of the triangle strip at each vertex of the polyline defining the thread, i.e. for every segment $\overline{\mathbf{pq}}$, two vertices are generated at the endpoint $\mathbf{p}$ (and a final pair of vertices is generated for the final point in the polyline).

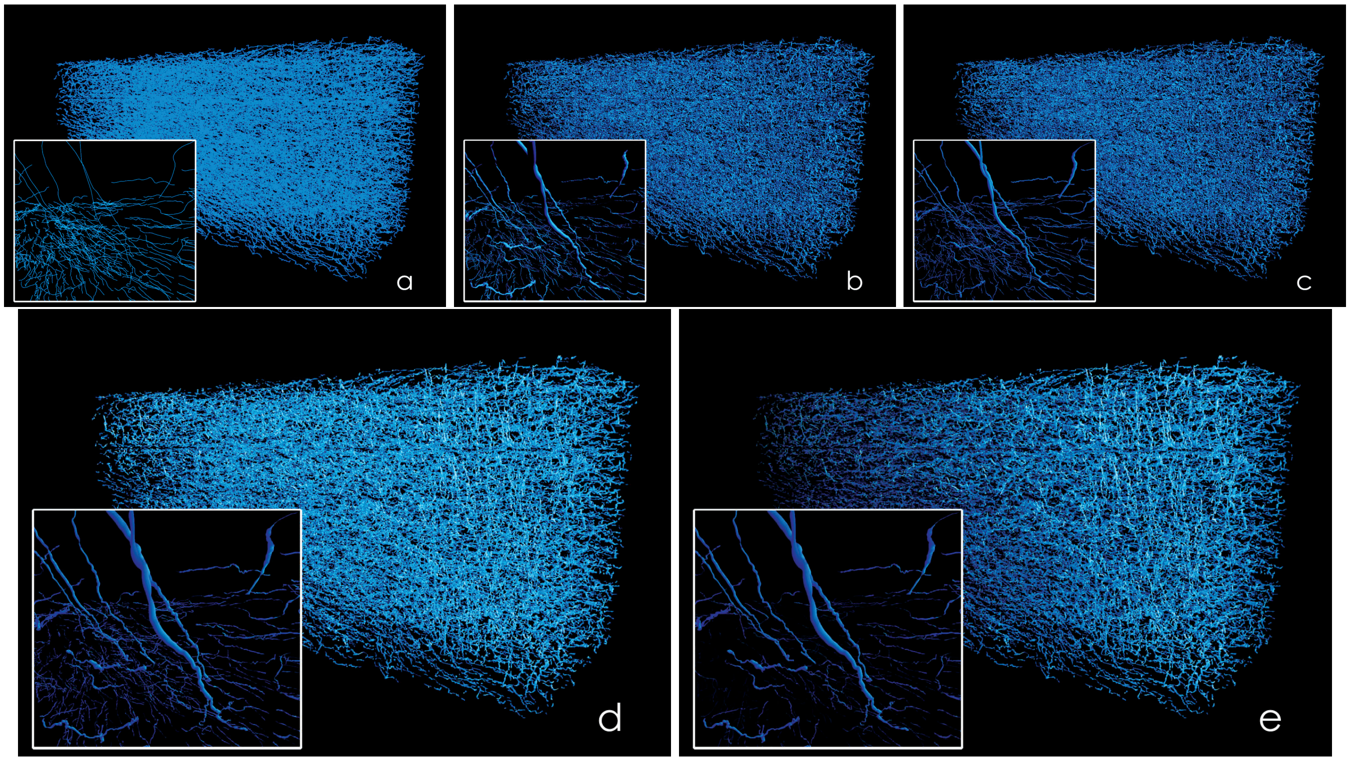

Fig. 4. SBF-SEM dataset with 240000 threads is visualized using (a) lines (b)polygon cylinder (c) SOS (d) GPU SOS (e) GPU SOS with depth attenuation

The position of the vertices will be set at runtime by the vertex shader. For each vertex, we need to send the direction of the segment $\vec{d} = \mathbf{q} - \mathbf{p}$, and the radius r at point $\mathbf{p}$, along with a tag to distinguish these vertices from one another (i.e. we set one r positive, and the other negative). All this information can be stored in a single four-channel texture coordinate (i.e. RGBA is actually d_x, d_y, d_z, r). Note that we do not need to explicitly send the position of the axis, since the vertices are initially positioned on the axis. The vertex shader does the following:

- Transforms vertex to view space
- Finds vector perpendicular to the view direction and local thread orientation
- Displaces vertex according to the perpendicular vector and radius at that vertex
- Sets vertex normal according to the perpendicular vector and direction of radius at that vertex
- Applies shading using distance and orientation filtering

The vertex shader applies a view space transform MV (and its inverse transpose, MV^{IT}) to the vertex position and thread direction (note: in OpenGL, MV is the ModelView matrix) as follows:

$$\mathbf{p}' = MV\mathbf{p}, \quad \vec{d'} = \vec{d}MV^{IT} \tag{1}$$

After the transform, the direction, $\vec{d_\perp}$, perpendicular to both view and $\vec{d'}$ is found easily using:

$$\vec{d_\perp} = \begin{pmatrix} -d'_y \\ d'_x \\ 0 \end{pmatrix} \tag{2}$$

and the final position for each of the vertices is then determined

$$\mathbf{p}'' = \mathbf{p}' + r\frac{\vec{d_\perp}}{|\vec{d_\perp}|} \tag{3}$$

Note that for segments (nearly) parallel to the view direction, equation 3 is ill-defined. This is a fundamental problem for any SOS approach. Stoll et al. [17] deal with this problem (their method uses cross products that are ill-defined as well) by adding a geometry "patch" in the problematic area during a CPU pass in their CPU-GPU hybrid approach. In our case, we use an ad-hoc GPU-based approach. GPU programming languages such as CG allow normalization of a zero-vector, usually returning a zero vector. In such a case, the quadrilateral edge collapses to a point, but this does not create major visual artifacts, as the quadrilateral is viewed edge-on.

Normals $\vec{n}$ are defined at the vertices facing outward to give the illusion of curvature to the flat strips after shading.

$$\vec{N} = \begin{pmatrix} -d'_y sign(r) \\ d'_x sign(r) \\ 1 \end{pmatrix}, \quad \vec{n} = \frac{\vec{N}}{|\vec{N}|} \tag{4}$$

Note that all the equations are applied after view space transformation, and all the coordinates are in view space coordinates. Shading and specular lighting is applied in the vertex shader in the usual way [14]. Additional visual cues are also created in the vertex shader. Depth based attenuation is applied using $\mathbf{p}'_z$, which is already calculated above. Focus point centered attenuation is applied using only one additional vector subtraction operation, requiring only that the focus point be sent as a uniform variable. Thickness of the segments can be varied interactively by using another uniform variable. None of these additional visual cues require rebuilding the display list or resending vertex data to the graphics card, hence they can be applied uninterrupted and interactively by sending only a small number of global parameters to the vertex shader. We observed that such additional features required only negligible additional processing that did not significantly affect the framerate. Note that more complex shading, such as a halo effect [17], could easily be done in the fragment shader.

3.2 Orientation and Radius Filtering

Although the neuronal threads tend to cluster into flow-like structures, they behave differently from traditional flow patterns, and traditional flow visualizations are not very useful. Differently oriented groups of threads can pass through each other or merge into bundles just to separate later. A key motivation for our work is to provide tools to allow researchers to investigate these structures and their locality. Note that we are not interested in individual threads but rather in bundles of similarly behaving threads. Orientation filtering enables the viewer to select different orientations and color the threads accordingly.

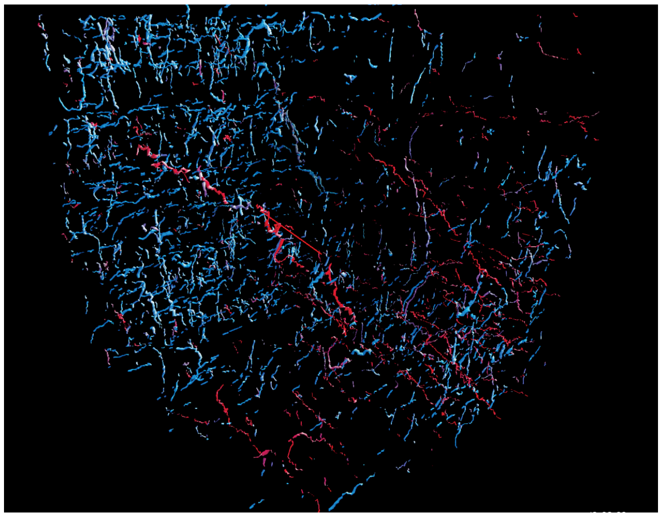

Fig. 5. Orientation filtering is used to find a large vascular structure in a Nissl KESM dataset.

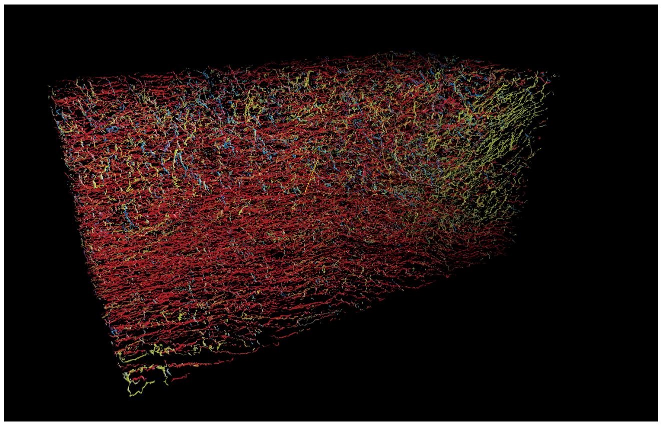

Fig. 6. Orientation filtering through SBF-SEM data shows two separate dendritic flows in red and yellow. Blue is the base color.

We allow a user to specify an orientation interactively. Inside the vertex shader, the dot product of the segment direction and the selected orientation $\vec{o}$ is used to linearly interpolate between a base color c_b and an orientation color c_o to find the output color C. However, due to imaging and tracing noise, this scheme did not give satisfying results. Also, neuronal threads tend to "wander", such that individual (local) segments may vary wildly in orientation, even while the thread as a whole has a well-defined direction. Thus, we use the thread direction $\vec{d_t}$ between the two endpoints of the thread, instead of the endpoints of the individual segment, to perform the coloring. This requires the thread direction to be stored at each vertex, and this is used in the dot product instead of the segment direction. This scheme adds one more per-vertex vector to the display list.

$$\alpha = |\vec{d_t} \cdot \vec{o}| \tag{5}$$

$$C = (1 - \alpha)c_b + \alpha c_o \tag{6}$$

One should note that, once this direction is known, multiple orientations could be filtered at the same time in the vertex shader interactively. Using two user defined orientations, one can easily display the major flow direction and a separate local flow in our most complex -to date- dataset of 240, 000 segments from SBF-SEM data.

Following the suggestions from an evaluation by expert neuroscientists, we added additional filtering capabilities using radius and thread length. By adding total thread length and maximum thickness of the thread into the vertex data the user can quickly filter out thin or short threads to better understand the major structures (see Figure 7).

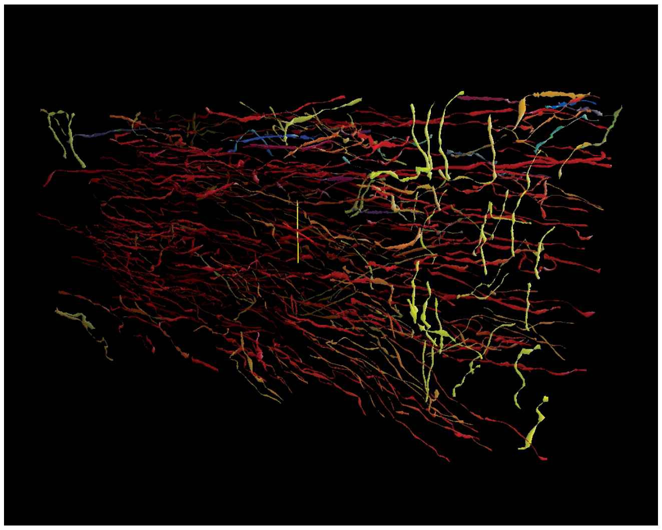

Fig. 7. Orientation filtering using the SBF-SEM data with only longer threads.

3.3 Smoothing the Axis

Neuronal threads tend to have a very jagged structure. Noise in the imaging system and our vector tracing algorithms tend to enhance this. Sharp corners are a challenge in SOS, since sharp changes in direction can result in a flattened SOS strip in places. Also, the jagged structures are visually distracting when trying to understand underlying organization. Although vector tracing could be arranged to return smoother results, this might not be desirable since it changes the output of a lengthy process of tracing. We instead provide smoothing at the visualization stage.

To make the SOS strip behave smoothly through the corners we define segments as cubic Bezier curves. The endpoints are maintained, and two additional points are needed; these will control the first derivative at the start and end points of the segment. These derivatives are set at the start point by

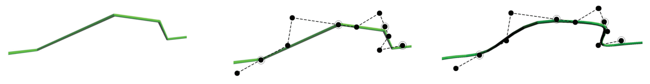
Fig. 8. Smoothing through segments

extending the parent segment; at the end point, the segment direction itself is used. Thus the derivative at each point is always set parallel to the direction of the previous line segment. This scheme ensures C1 continuity between segments (Fig. 8) and even branches (Fig. 9). All sibling segments at a branch will share the same starting direction.

Note that one could also use quadratic splines instead of cubic, however this becomes problematic when dealing with branching points, and tends to make the spline deviate much more from the original segmented shape. While a quadratic spline between two segments can easily be defined by averaging points, this is not feasible at branching points. If instead we extend a "parent" (similar to what we propose) to define the starting derivative, then this information must "cascade" down to later segments - i.e. the smoothing will become a global rather than local operation. Furthermore, the quadratic spline shape will tend to deviate much more from the original polyline shape. A hybrid quadratic-cubic spline approach might prove most helpful, but we prefer to go with the simplicity of only the single cubic approach.

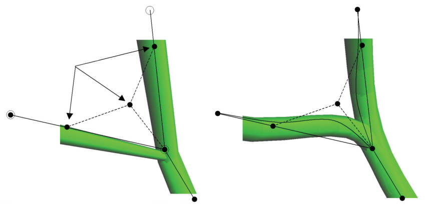
Fig. 9. Interpolating branch segments into a single branch point. (left) derivatives of the end points of each branch (right) the single derivative at the branch point guarantees continuity

We subdivide each segment into subsegments using the cubic Bezier curve formed by these four control points. We also interpolate the radius along the curve. The results are smooth and more visually satisfying. Any error introduced in this smoothing is likely to be minimized, as the majority of each segment still follows the original thread path. However, note that this does increase the number of vertices depending on the number of sub-segments used when rendering. This number can be minimized by varying the sub segment count according to the angle between segments.

Note that currently subdivision is performed on the CPU, as GPUs have not supported vertex creation. This should change in the near future, making it possible to send the curve directly, and allow the GPU to subdivide as necessary.

4 Visualizing Thread Data as Hair

The interactive methods described above allow us to display our 3D data effectively, and obtain some important information, such as thread orientation over a large region. However, we would also like to have a higher-quality rendering (such as might be obtained by global illumination) that enables us to better visualize and understand the underlying structure.

However, the dense fibrous data sets we are dealing with do not lend themselves well to techniques that might work well for simpler data, such as that from flow lines.

Because of the somewhat similar structure, we have chosen to treat our thread data as a set of hair strands. Hair is a structure and material that we all see quite often in our lives. Therefore, when thread data is visualized as hair, this familiar form improves the perception of the thread structure. We make use of high-quality hair illumination and shading techniques to allow better perception of depth and structure in the image (see Figure 10).

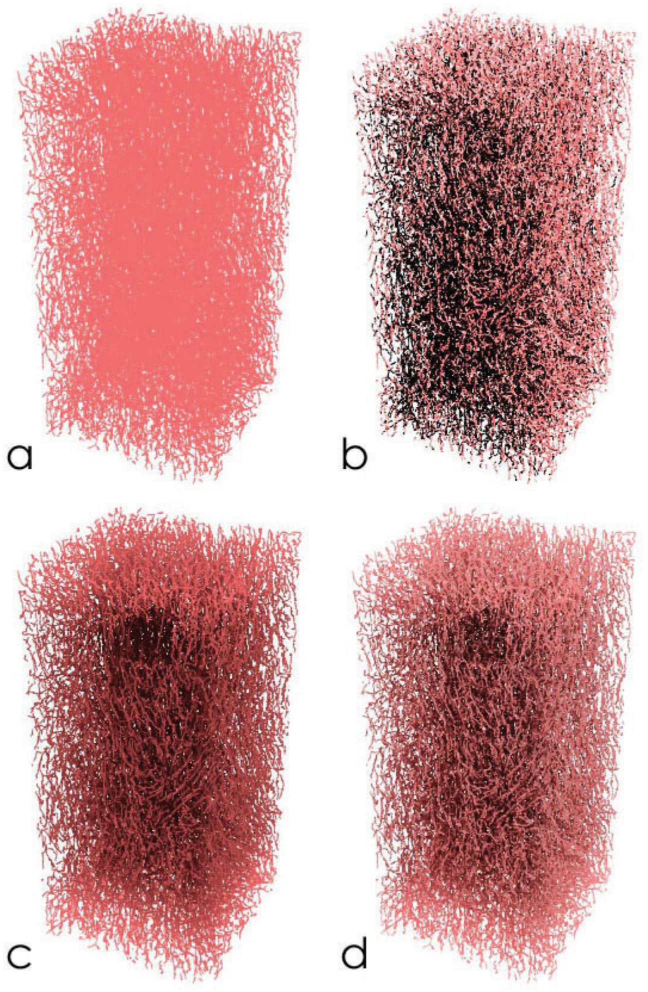

Fig. 10. Hair rendering on SBF-SEM data. (a) ambient only (b) spot light with shadows (c) fake skylight (d)fake skylight with spot light

4.1 Illumination and Shading

Using photo-realistic rendering techniques has an important effect on the visual perception of the thread data. Illumination and shading are the key elements to achieve this goal. Still, our concerns about realism are fairly limited - we are not viewing the data in its "real" form, and the only reason we wish to use realistic rendering techniques is in order to improve the perceptual quality of the image.

For our higher-quality images, we use both point lights and skylight. To compute the self shadows from point light sources we use the opacity shadow maps [9] algorithm. The illumination computation is slower for a skylight, which requires a very expensive occlusion calculation for each shading point, as we need to determine the amount of light coming from every direction. To reduce this vast computation, we use an occlusion estimation function to compute the amount of light at each shading point. In our system, the occlusion estimation of a point inside the thread volume is a linear function of the distance of the point to the nearest point outside the thread volume. This approach yields a simple 3D gradient function for the rectangular thread volumes that we visualize.

For shading thread lines we use the Kajiya-Kay hair shading equation [8]. Though Marschner et al.[11] allow a more realistic shading solution, the Kajiya-Kay equation is widely accepted by the graphics community and provides enough realism for our purposes.

4.2 Neuron Visualization as Hair

Unlike opaque surfaces, hair can be effectively illuminated from behind. This property of hair is very useful for thread visualization, since it makes lighting design easier, and allows us to target regions that are in the interior of a dense volume. We used two separate lighting techniques for visualizing both large sets and small subsections of the data.

When visualizing the whole data set at once, we use a skylight and point lights that are outside the thread volume. This lighting design helps to differentiate the threads that are inside the thread volume from the ones that are near the surface, and makes it easier to perceive the volumetric structure of threads. Figure 10 shows a comparison of this lighting scheme to constant illumination. As can be seen, the proposed lighting model improves the visual perception of the thread structure significantly.

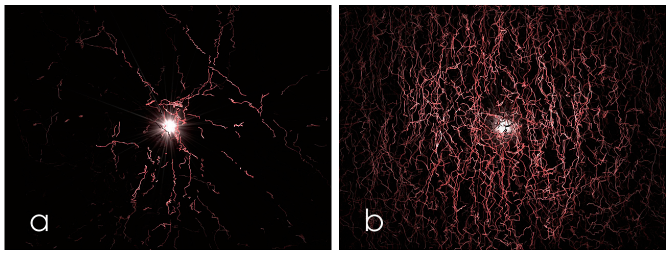

Fig. 11. Point light source inside (a) golgi (b) SBF-SEM dataset

Another technique we used is particularly designed for visualizing a small volume inside the thread data. In this case we place a point light source with strong falloff inside the thread volume. The point light illuminates the volume of interest, while the other areas are kept dark. This visualization technique is especially useful to analyze low-level thread connections. Figure 11 shows examples of this technique.

5 Performance and Evaluation

5.1 Domain Expert Evaluation

Our visualization has been evaluated by an expert neuroanatomist we collaborate with, as well as one of her experienced graduate students (hereafter referred to as "the users"). The users said that the vector-traced data visualization produced the results that they expected for fiber bundles in the SBF-SEM data and for neuronal processes in the Golgi stained tissue samples, and furthermore that they could find structures in the Golgi data more easily by using the interactive 3D system. That is, our visualizations allowed the users a 3D visual representation of what had previously been a conceptual image of the structures.

The users felt that a major advantage of the visualization was the ability to visualize the large volume of data at once. As such, the efficiency of our GPU-SOS implementation seems critical. The orientation filtering was seen as particularly useful for differentiating the fiber bundles that intersect or change direction rapidly; this is a common problem in traditional microscopy. Also, as a direct result of the users' evaluation, we implemented radius filtering, to help in identifying anatomical structure.

As far as we are aware, these datasets are currently the only such reconstructions of large-scale neuronal data in 3D. Although our current data sets are limited to scans of "normal" neuronal tissue, the users felt that our visualization approach would be particularly helpful for understanding differences between standard tissue and pathological specimens (when they become available), and for comparing anatomical structures across multiple specimens.

5.2 Performance Results and Discussion

By moving the SOS computation to the GPU, we offload the calculations of camera facings. Display lists are also now available to minimize per frame data transfer. In our measurements, GPU-based SOS performed around 10 times faster than CPU-based SOS. Our SBF-SEM dataset consists of 240000 threads; CPU-based SOS gives 2 fps whereas the GPU implementation runs at 22 fps (Table 1). Polygon rendering of this dataset is extremely slow. Note that the GPU-SOS figure includes depth attenuation and orientation filtering; additional visualizations could also be performed interactively on the vertex shader with negligible cost. Figure 4 compares several results with our method.

The recent work of Stoll et al. [17] provides a GPU approach to line drawing that is similar to ours. They apparently use a double cross product to find the orthogonal direction; our use of a view space approach provides a significant speedup. The hybrid CPU-GPU approach they propose to deal with singularities could also be applied to some (though not all) of the singularities in our data. For our applications, we find that the additional visual improvement we would get from patching the data is not sufficient to outweigh the need for additional speed.

Clearly, self-orienting surfaces are particularly effective for allowing visualization at interactive rates, particularly as they can easily be adapted to a GPU use. For very large data sets, such an approach is the only currently feasible approach we are aware of. However, it should be noted that there are some drawbacks to using this approach for thread-like data. In particular, surfaces can potentially twist along the pathway or at branching points, giving undesirable appearances.

6 Conclusion and Future Work

In this paper, we have described the production of fibrous data, including smoothing data by approximation by splines, a simple method for visualizing such fibrous data quickly using GPU-supported self-orienting surfaces, a simple method for exploring orientation information in such fibrous data, and a high-quality rendering approach that allows one to better see the structure of individual fibers.

There are several avenues open for future work. First, while the orientation filter has proved useful, a number of other filtering enhancements could be provided, and have been suggested by our evaluators. This includes multi-dimensional color mapping, individual thread targeting/highlighting, and tools to support or visualize merging or branching of thread data. Integrating such filters into a GPU-based setting will be critical. Second, while hair rendering seems an obvious analogy for the fibrous data structures we render, a different approach to high-quality global illumination might be even more appropriate. Third, while our current test data sets are very large by current standards, it is not clear how well these techniques will scale up to the massively larger (two orders of magnitude larger) data sets expected in the next few years. Finally, while the routines here have been developed for our particular data sets (currently the only data sets of this type in existence), and evaluated by our collaborators, it would be useful to get an evaluation over more samples by a wider range of neuroanatomists.

7 Acknowledgements

This work was supported in part by NIH/National Institute for Neurological Disorders and Stroke grant #R01-NS54252 and by NSF grant CCF-0220047. KESM data was obtained in the Brain Networks Lab, headed formerly by Bruce McCormick,

	threads	Wireframe	Polygon	SOS	GPU-SOS without any shading	GPU-SOS using depth and orientation filtering
Simple Neuron	29	1250 fps	201 fps	582 fps	1640 fps	1600 fps
Vascular	46513	29.1 fps	1.9 fps	10.4 fps	566 fps	166 fps
Golgi	56690	24.9 fps	1.27 fps	9.1 fps	478 fps	134 fps
SBF-SEM	241407	4.7 fps	0.99 fps	1.7 fps	80 fps	22.3 fps

Table 1. Test results on a 256 MB NVIDIA GeForce 6600

and currently by Yoonsuck Choe. Fiber tracing routines were developed in the BNL by Brad Busse and Purna Doddapaneni. SBF-SEM data was obtained by and provided courtesy of Winfried Denk[5]. Expert evaluation was provided by Louise Abbot and Kerry Thuett.

References

[1] K. Al-Kofahi, S. Lasek, D. Szarowski, C. Pace, G. Nagy, J. Turner, and B. Roysam. Rapid automated three-dimensional tracing of neurons from confocal image stacks. *IEEE Transactions on Information Technology in Biomedicine*, 6(2):171–186, June 2002.

[2] H. Axer, G. Berks, and D. G. V. Keyserlingk. Visualization of nerve fiber orientation in gross histological sections of the human brain. *Microscopy Research and Technique*, 51(5):481–492, 2000.

[3] A. Can, H. Shen, J. N. Turner, H. L. Tanenbaum, and B. Roysam. Rapid automated tracing and feature extraction from retinal fundus images using direct exploratory algorithms. *IEEE Transactions on Information Technology in Biomedicine*, 3(2):125–138, June 1999.

[4] C. De Boor. *A Practical Guide to Splines*. Springer, 1978. de Boor.

[5] W. Denk and H. Horstmann. Serial block-face scanning electron microscopy to reconstruct three-dimensional tissue nanostructure. *PLoS Biology*, 2(11):e329, Nov. 2004.

[6] P. Doddapaneni. Segmentation strategies for polymerized volume data sets. Master's thesis, Department of Computer Science, Texas A&M University, 2004.

[7] G. E. Farin. *Curves and Surfaces for CAGD: A Practical Guide*. Morgan Kaufmann Publishers, San Franscisco, CA, USA, 5th edition, 2002.

[8] J. T. Kajiya and T. L. Kay. Rendering fur with three dimensional textures. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 271–280, New York, NY, USA, 1989. ACM Press.

[9] T.-Y. Kim and U. Neumann. Opacity shadow maps. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 177–182, London, UK, 2001. Springer-Verlag.

[10] P. Kondratieva, J. Krüger, and R. Westermann. The application of gpu particle tracing to diffusion tensor field visualization. In *IEEE Visualization*, page 10, 2005.

[11] S. R. Marschner, H. W. Jensen, M. Cammarano, S. Worley, and P. Hanrahan. Light scattering from human hair fibers. *ACM Trans. Graph.*, 22(3):780–791, 2003.

[12] B. McCormick. Development of the brain tissue scanner. Technical Report 18, Department of Computer Science, Texas A&M University, College Station, TX, Mar. 2002.

[13] B. McCormick, B. Busse, P. Doddapaneni, Z. Melek, and J. Keyser. Compression, segmentation, and modeling of filamentary volumetric data. In *Proceedings of Symposium on Solid Modeling and Applications '04*, pages 333–338, 2004.

[14] NVIDIA. *CG Toolkit User's Manual 1.2*, January 2004.

[15] C. Rössl, F. Zeilfelder, G. Nürnberger, and H.-P. Seidel. Visualization of volume data with quadratic super splines. In *IEEE Visualization*, pages 393–400, 2003.

[16] G. Schussman and K.-L. Ma. Scalable self-orienting surfaces: A compact, texture-enhanced representation for interactive visualization of 3d vector fields. pages 1–10. IEEE, Oct. 2002.

[17] C. Stoll, S. Gumhold, and H.-P. Seidel. Visualization with stylized line primitives. In *IEEE Visualization*, page 88, 2005.