Eurographics/ IEEE-VGTC Symposium on Visualization (2006)
Thomas Ertl, Ken Joy, and Beatriz Santos (Editors)

GPU-Based Hyperstreamlines for Diffusion Tensor Imaging

G. Reina[1†] and K. Bidmon[1†] and F. Enders[2‡] and P. Hastreiter[2‡] and T. Ertl[1†]

[1]Visualization and Interactive Systems Group (VIS), University of Stuttgart, Germany
[2]Neurocenter, Dept. of Neurosurgery and Computer Graphics Group, University of Erlangen-Nuremberg, Germany

Abstract

We propose a new approach for the visualization of hyperstreamlines, which offers potential for better scalability than the conventional polygon-based approach. Our method circumvents the bandwidth bottleneck between the CPU and GPU by transmitting a small set of parameters for each tube segment and generates the surface directly on the GPU using the classical sphere tracing approach. This reduces the load on the CPU that would otherwise need to provide a suitable level-of-detail representation of the scene, while offering even higher quality in the resulting surfaces since every fragment is traced individually. We demonstrate the effectiveness of this approach by comparing it to the performance and output of conventional visualization tools in the application area of diffusion tensor imaging of human brain MR scans.

The method presented here can also be utilized to generate other types of surfaces on the GPU that are too complex to handle with direct ray casting and can therefore be adapted for other applications.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism J.3 [Computer Graphics]: Life and Medical Sciences

1. Introduction

Diffusion tensor imaging (DTI) is a magnetic resonance imaging (MRI) technique with increasing popularity especially in neurosurgery. Instead of information about quantity and linkage of hydrogen, it provides information about the diffusion properties of molecules in tissue. This is of special interest since the underlying cell structure influences these properties in a way such that strongly aligned cells restrict the diffusion to an anisotropic behavior. As important neuronal pathways feature these cell characteristics, one can infer about the occurrence of such major white matter tracts based on DTI data. From the medical point of view, this is particularly interesting since the protection of such major white matter tracts is of utmost importance during intervention. Therefore, visualization of the tracts is getting increasingly relevant to diagnosis and planning. However, many visualization methods are not capable of representing all tensor information in a comprehensive manner. This is disadvantageous since diagnosis is improved by the availability of additional information.

Our approach uses the diffusion tensor eigenvectors to generate oriented ellipses at the sampled points, which are then linearly interpolated to form tubes connecting the sampling points. Together the tube sections form hyperstreamlines as shown in [DH93]. Instead of creating a mesh from these tubes on the CPU, we just upload the orientation and size parameters of each tubelet to the graphics card and use sphere tracing [Har96] to generate the surface on the GPU. Since hyperstreamlines in medical visualization are most useful when displayed in the context of a volume representation of the surrounding tissue, we want to minimize data transfer as much as possible. That way, the system bus bandwidth can be used to upload volume data in case of time-dependent or bricked datasets. Since our approach does not access textures at all, the volume visualization can use all of the available bandwidth on the graphics card to keep performance as high as possible. On the other hand, our approach to the rendering of hyperstreamlines relies on the computational power that is available on current GPUs but is not subject to heavy load from direct volume rendering approaches. This paper is structured as follows: in Section 2 we summa-

† e-mail: {reina | bidmon | ertl}@vis.uni-stuttgart.de
‡ e-mail: {enders | hastreiter}@informatik.uni-erlangen.de

rize related work, Section 3 describes the MR data we use as a base for our visualization. Section 4 details the graphical representation of the tube parts as well as the mathematical background, and Section 5 shows how the surface rendering works on the GPU. In Sections 6 and 7 we give some results and discuss the performance of our approach. Section 8 concludes this work.

2. Related Work

Several strategies have been applied to the field of DTI visualization. Mapping tensors to scalar values and displaying them in a separate slice is a well-known method for diagnosis purposes [ERMB00, PAB02]. For a more comprehensive visualization of DTI data, volume rendering was proposed by Kindlmann *et al.* [KW99, KWH00] who utilized textures and transfer functions to represent the tensor properties. To investigate the features of the tensor per voxel directly, glyph-based approaches have been employed [WMM*02, Kin04]. They represent each tensor independently by shape, size, color, and other attributes of a glyph placed at the corresponding voxel position. These direct visualization techniques are useful for a detailed inspection of the DTI dataset. However, their weakness is the missing connectivity of the data which prevents the analysis of complete pathways.

To overcome this problem, streamline tracking techniques were adapted to DTI processing [BPP*00, LAS*02]. Additional improvements have been achieved by the use of streamtubes [ZDL03] and hyperstreamlines [DH93, WL01] since streamline techniques are limited to vector fields which can be partly compensated by the use of three-dimensional objects. Finally, neuronal pathways can be extracted and visualized as surfaces which support an intuitive understanding of the data [ESM*05].

Generating parametric surfaces on the GPU has been investigated in several publications, be it for explicitly generating geometry in vertex buffers [MP03, LH04], be it for ray casting the surfaces directly on the GPU, as has been done for ellipsoids [KE04, Gum03] and application-specific glyphs [RE05], to mention just a few. Sphere tracing [Har96] has been recently reproposed for displacement mapping on graphics hardware [Don05].

An approach very closely related to our own was proposed by Stoll *et al.* [SGS05], however the tubes generated by their method are limited to circular cross-sections as opposed to our elliptical cross-sections. This trade-off allows for the rendering of the resulting tubes using splatting techniques that yield much higher frame rates than our approach at the cost of a lower information density.

3. Data

For the computation of diffusion tensor data, six diffusion-weighted images with different gradient directions were measured in combination with a reference image, measured without any gradient. Based on this set of images, the real-valued symmetric second-order tensor can be determined [WMM*02]. This tensor represents the diffusion characteristics of hydrogen averaged over the volume of the corresponding voxel. The eigensystem of this tensor can be evaluated, resulting in three real eigenvalues and the corresponding orthogonal eigenvectors.

The first step towards hyperstreamlines is the tracking of the related streamlines. Integration schemes known from flow visualization, such as Euler or Runge-Kutta are applied to determine streamlines through a vector field. The required input vector field is derived by a simple reduction of the tensor to its principal eigenvector. The loss of information is partly compensated by introducing a threshold for tracking. A possible parameter is fractional anisotropy (FA) which is a measurement for anisotropy [BMP*01]. It serves as stop criterion to terminate tracking when running into areas of reduced anisotropy with low probability for neuronal pathways. In our approach all voxels above the specified FA threshold were used as seed regions. As soon as the tracking enters a voxel with an FA value below the threshold it stops. A resulting whole-brain tracking can be seen in Figure 1. Afterwards, subsets of fibers were selected using manually defined regions of interest (ROI).

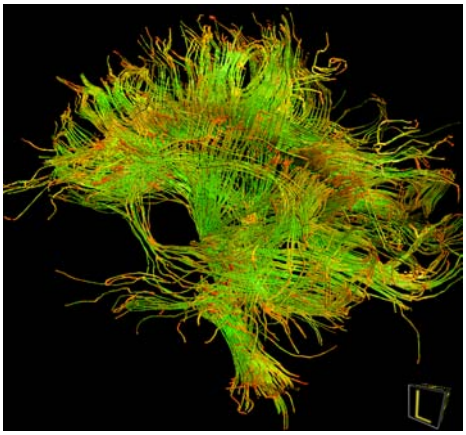

Figure 1: *Hyperstreamline visualization of a whole-brain tracking. The viewpoint is positioned left, posterior above the brain. Green segments indicate tracking in reliable regions with high anisotropy. Yellow tube sections suggest fiber crossings. The absence of red segments in the central parts of the hyperstreamlines documents the correctness of the fiber tracking algorithm. Dataset as used for performance measurements in Table 1.*

The drawback of streamlines in the context of DTI is certainly the restriction to a vector field. Features of the tensor field like torsion or the minor eigenvalues cannot be displayed with this method. To overcome this problem, hyperstreamlines are applied which are capable of visualizing

such attributes. The eigenvalues and eigenvectors of the tensor at each base point of the streamline serve as basis for the twist and the semi-major axes of the segments of the hyperstreamline.

4. Tubelets

To visualize the tensor field, the data is mapped to a special kind of tubelet shaped by ellipses within a slice plane at each end. These ellipses are defined by their semi-major and semi-minor axis and its rotation around the x axis. The shape along the tubelet is determined by linear interpolation as further described in Section 4.2. Taking into account the special needs of our given data, a local coordinate system is introduced in Section 4.1, and some more complicated customizations on distance measuring and rotation interpolation are required as described in Section 4.3.

4.1. Definition of the Local Coordinate System

For each tubelet a local coordinate system is defined, depending on the two ellipses that define the tubelet's shape. The ellipses are defined by an eigensystem each, where the normalized eigenvectors $\vec{e}_i$ are sorted by their corresponding eigenvalues v_i in descending order: $v_1 > v_2 > v_3$. As $\vec{e}_1$ is the hyperstreamline's direction, the two minor eigenvalues define the length of the ellipse's semi axes and the corresponding eigenvectors define the corresponding direction of each semi axis. The signs of the eigenvectors are chosen in a way such that the eigensystem is a right-handed orthonormal basis (as mentioned in Section 3) that can be used directly to define the tubelet's local coordinate system.

The local x-axis is given by the tubelet's axis $\vec{x}_t$, connecting the midpoints $\vec{p}_l$ and $\vec{p}_r$ of the two ellipses ($\vec{x}_t = (\vec{p}_r - \vec{p}_l)/\|\vec{p}_r - \vec{p}_l\|$), and the other two axes are derived from the left end ellipse as follows: The ellipse's first eigenvector $\vec{e}_1$ is to point in the same direction as the tubelet axis. This is done by rotating the eigensystem around $\vec{e}_1 \times \vec{x}_t$ by an angle of $\arccos(\vec{e}_1 \cdot \vec{x}_t)$. Then the rotated eigenvectors $\vec{e}_2$ and $\vec{e}_3$ define the tubelet's local y-axis $\vec{y}_t$ and z-axis $\vec{z}_t$ respectively.

4.2. Geometrical Definition of the Tubelets' Shape

Geometrically defining the shape of a tubelet first requires some considerations about ellipses in the yz-plane: Assuming r_1 and r_2 to be the two semi axes, ellipses are usually described by the Cartesian equation

$$\frac{y^2}{{r_1}^2} + \frac{z^2}{{r_2}^2} = 1\,. \tag{1}$$

Our ellipses are defined within the yz plane and may be rotated around the x axis. Therefore a representation of the ellipse corresponding to the polar coordinates of a circle is easier to handle. Additionally, we need the surface normal later for correct lighting, which is also easier to calculate in polar coordinates.

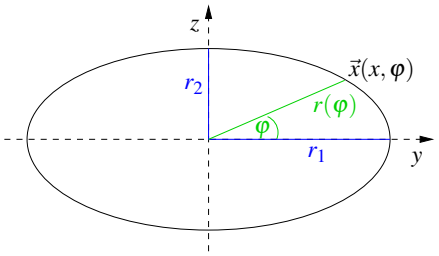

Figure 2: *Ellipse in polar coordinates.*

Let φ be the angle to the y axis, then the ellipse can be given in polar coordinates by

$$\vec{x}(x,\varphi) = \begin{pmatrix} x \\ r(\varphi)\cos\varphi \\ r(\varphi)\sin\varphi \end{pmatrix} \tag{2}$$

with $r(\varphi)$ determined by plugging these coordinates into the Cartesian equation (1)

$$r(\varphi) = \frac{r_1 r_2}{\sqrt{r_1 \sin^2\varphi + r_2 \cos^2\varphi}}\,. \tag{3}$$

Within a plane of constant x, rotating the whole ellipse around the x axis by an angle ρ changes (2) to

$$\vec{x}(x,\varphi) = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x \\ r(\varphi)\cos(\rho+\varphi) \\ r(\varphi)\sin(\rho+\varphi) \end{pmatrix}\,. \tag{4}$$

The tubelets are defined by an ellipse at each end and are centered along the x axis of their own local coordinate system. These ellipses may vary in the length of their semi axes and in the rotation along x, and they are located at $(-l/2,0,0)$ and $(l/2,0,0)$ respectively, where l is the length of the tubelet as depicted in Figure 3.

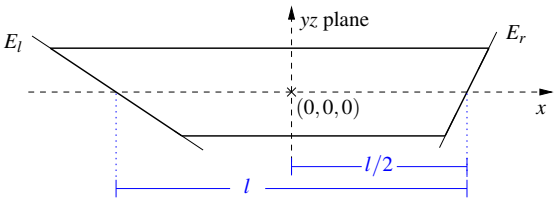

Figure 3: *Definition of a tubelet along x.*

For each x value along the tubelet the semi axes and the rotation angle ρ of the ellipse are calculated by linear interpolation from the properties of the ones at the tubelet's ends.

The tubelets are connected to each other to get longer tubes, so the cutting planes E_l and E_r at the ends of each tubelet have to be specified. In order to approximate curved tubes, these planes may be rotated arbitrarily as illustrated

in Figure 3. We will go into more detail about this in the following section, taking into account special needs arising with these properties.

4.3. Geometrical Background for Ray Casting

In order to ray cast each tubelet's surface we need the intersection point of the eye ray with the tubelet. As this intersection calculation leads to an equation of degree 4 which are very hard to calculate analytically and would require numerical solving methods, too expensive and time-consuming for graphics hardware. Therefore we decided to adopt the sphere tracing algorithm presented in [Har96]. Starting at the eye's position we step along the eye ray, determine the current distance to the tubelet's surface, and take this distance as the next step size to close in onto the tubelet. This procedure is repeated until the distance falls below a specified threshold ω. In most cases this works fine but some rays require additional adjustments which will be explained in Section 5.

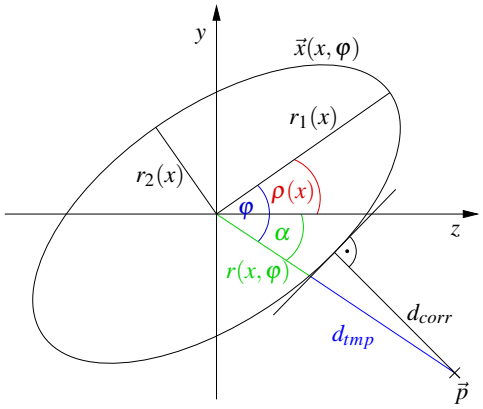

Figure 4: *Distance measurement from the current position $\vec{p}$ to the rotated ellipse $\vec{x}(x,\varphi)$ within a plane of constant x.*

From the current position, the shortest distance to the tubelet has to be calculated to get the new step size. Due to the rotation of the tubelet along x, the correct shortest distance d_{corr} to the tubelet's surface—which is measured along the surface normal $\vec{N}$—cannot be easily calculated as it would also lead to an iterative and time consuming way to find the solution. To avoid that, we decided to substitute the shortest distance by a close solution: we assume the x value to be the same as the one of our current position and calculate the distance d_{tmp} to the ellipse in this common plane normal to the x axis and through the current position as depicted in Figure 4:

$$d_{tmp} = \sqrt{\vec{p}_y{}^2 + \vec{p}_z{}^2} - r(x,\varphi) \tag{5}$$

where $\vec{p}_y$ is the y-component of $\vec{p}$ and $\vec{p}_z$ the z-component respectively, and with r satisfying (3) within the current plane of constant x value. In the next step we take the point d_{tmp} units further along the ray as our new current position. In case $d_{tmp} < 0$ we already intersected the surface and have to step backwards, which is equivalent to walking along the ray in negative direction and needs no special considerations.

In order to get a longer tube which is more flexible than a single tubelet we connect neighboring tubelets at their cutting planes E_l and E_r. To approximate a curved tube the planes are not necessarily normal to the local x axis, as mentioned before. These sloped tubelet ends cause further considerations about the tubelet's properties: merging the tubelets to form larger tubes also raises the problem of rotation consistency at the interconnections. Consistent properties of the ellipses are only guaranteed in slices perpendicular to x_t that do not contain parts of a neighboring tubelet, because otherwise there are two inconsistent interpolation parameters belonging to the two tubelets respectively. So we have to restrict the interpolation up to the point $\vec{q}$ as exemplified in Figure 5 for the tubelet's left end.

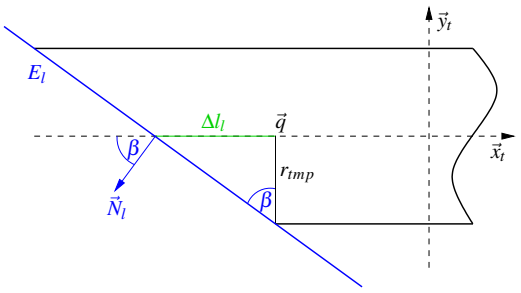

Figure 5: *Correction of distance and interpolation range. E_l is the left cutting plane, $\vec{N}_l$ the corresponding normal vector.*

To restrict the interpolation to the correct range of x values we need to calculate the distance $\Delta l = r_{tmp} \cdot \tan\beta$ with β being the angle between the tubelet's local z-axis z_t and the intersection line of the cutting plane and the plane normal to the local x-axis x_t, which is $\beta = 0$ for the left end and $\beta = \rho_r$ for the right end—due to the definition of the local coordinate system as described in 4.1—and r_{tmp} satisfying (3) with $\varphi = \beta - \rho(x)$ leading to

$$r_{tmp} = \begin{cases} r_{1l} r_{2l} / \sqrt{r_{2l}} & \text{(left end)} \\ r_{1r} r_{2r} / \sqrt{r_{1r} \sin^2 \rho_r + r_{2r} (\cos^2 \rho_r)} & \text{(right end).} \end{cases}$$

Taking this fact into account for the computation of the tubelet's total length we get

$$l_{total} = l + \Delta l_l + \Delta l_r \tag{6}$$

with $\Delta l_l, \Delta l_r \geq 0$.

To enhance the impression of the tubelet's surface shape the correct surface normals are needed for correct lighting. Using the derivatives of (4) we get the surface normal $\vec{N}'$ by evaluating

$$\vec{N}' = \frac{\partial \vec{x}}{\partial \varphi} \times \frac{\partial \vec{x}}{\partial x}. \tag{7}$$

We will go into more detail about lighting in the following section.

5. Sphere Tracing on the GPU

To generate tubelets on the GPU, we reduce the needed surface to its parameters as described in Section 4. The vertex program calculates a bounding cuboid to scale the rendering primitive (a point) accordingly. Most of the values that are constant for a whole tubelet are computed from the parameters as well. This additional data is then passed to a fragment program along with the parameters so it can find the proper surface intersection with one ray cast per pixel, add phong shading and correct the depth value to fit the geometry.

For the GPU-based part of the algorithm we need the position, two colors, orientation (as a quaternion), the four radii, the right end rotation angle, the total length and the normals of the two bounding planes. We can fit this data into five float quadruples (O_{local} and l, q, $r_{1l}, r_{2l}, r_{1r}, r_{2r}$, $\vec{N}_l$, $\vec{N}_r$ and ρ_r, as ρ_l is always 0 due to the definition of the local coordinate system) plus two byte quadruples (color), so we only need to upload 644 bytes per tubelet, which is less than what is needed for 10 triangles with normals and constant color. For each tubelet we upload a single point (with the attributes as texture coordinates) to the graphics card, since a point is the smallest type of billboard in terms of data size, and as bonus we do not even need to adjust the orientation to face the eye position $\vec{p}_e$.

The vertex program computes two orientation matrices. The first one (M_c) is obtained after combining the tubelet orientation quaternion with the camera orientation quaternion. It is used for orbiting the eye point around the tubelet, to obtain the local coordinate system described in Section 4.1. The second matrix (M_o) is obtained from only the tubelet orientation and used to calculate a bounding cuboid from the worst-case dimensions of the tube, i.e. a cylinder with length l_{total} and radius $max(r_{1l}, r_{2l}, r_{1r}, r_{2r})$. This cuboid is projected onto the view plane to obtain the point's extents and center. Since the light position is also constant for all pixels of one tubelet, we rotate the light vector of our single light source with M_c in order to always have a headlight-like illumination.

The parameters we have to pass to the fragment program are the following: the eye position $\vec{p}_e$ relative to the tubelet centered at $(0,0,0)$, the rotation matrix from the combined quaternions M_c, the transformed light vector, the two bounding planes and the radii and length. Furthermore we only need to calculate $\Delta l_l, \Delta l_r$ once for each tubelet so they are computed in the vertex program and passed to the fragment shader. Together with the re-oriented z vector $\vec{o}'$ we use up all available varying parameters shared between vertex and fragment program.

The fragment program first has to find the vector $\vec{s}$ which connects the eye to the current pixel starting from the x and y component of the fragment's *window position WPOS*. To account for the fact that the origin is at the tubelet and the eye point orbits around it, M_c has to be applied to the resulting normalized ray direction $\vec{s}$ as well. To speed up the iteration process, we use an approximation of the intersection point as our starting point: The tubelet's surface is enclosed between two conical frustums, interpolating between the major axes of both ellipses in the bigger frustum and interpolating between the two minor axes in the other one. We calculate the intersection of our ray with the two cones and use the middle point between both intersection points as the start point for ray casting.

We then walk along the ray with a step size computed from the approximated distance as in (5) until either the distance is below a threshold ω or a maximum number of steps has been walked (see below). If the distance left after the last step is beyond this threshold or we are outside the two clipping planes E_l, E_r, the fragment is discarded. This leads to holes in the surface if the ray is approximately parallel to the tubelet's axis as the step size is almost constant and often very small, but it might still be far from the intersection point with the surface. To enhance the results in that case, we use an adaptive step size: if the angle between $\vec{x}_t$ and $\vec{s}$ is small, we scale the step size according to the angle between the surface normal and the ray direction by $(1-\cos(\vec{N}\cdot\vec{s}))+1$ which avoids these holes and thus leads to much better results with less iteration steps.

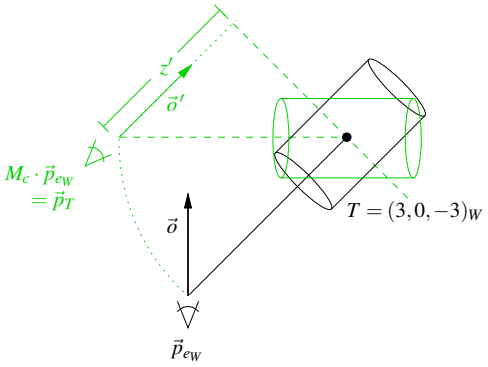

Figure 6: *Local tubelet coordinates (green) in relation to world coordinates (black).*

In case the fragment is not discarded, the correct depth is calculated to ensure that tubelets intersect correctly with each other and the volume as well. Since the eye point is displaced from $(0,0,0)^T$ and the view direction is no longer $\vec{o} = (0,0,-1)^T$, the depth z' is the distance to a plane normal to the orientation transformed by $\vec{o}' = M_c^T \vec{o}$ (see Figure 6), so we can use the Hessian normal form to get the distance and then fit the result to the exponential OpenGL depth range:

$$z' = \vec{o}' \cdot (\vec{p} - \vec{p}_e)$$
$$z_{ogl} = -\frac{z_F + z_N}{2(z_N - z_F)} + \frac{1}{2} + \frac{z_F z_N}{(z_N - z_F)z} \tag{8}$$

The normal is calculated as defined in (7) and used for Phong shading of the surface.

A drawback of this technique is that one cannot look inside the tubelets, since for rendering the back face we would have to start iterating on the other side of the local x axis and thus require two times the performance we need now. However there is no need to render the back face since users are usually not looking down the length of a tubelet because the relevant information (how the rotation and radii change over a certain distance) is visualized along the length of the tubelet and has to be interpreted in the context of the local x coordinate.

6. Results

The advantage of DTI is its capability to provide the intrinsic diffusion properties of water within tissue. Due to the anatomical structure of neuronal pathways this diffusion is anisotropic in areas of major white matter tracts. Thus, DTI can reveal coherences in-vivo which are not visible in MRI_{T1} or MRI_{T2} datasets. An accepted method to access this information is to apply fiber tracking. However, since streamlines cannot convey tensor information their extension to hyperstreamlines is of certain value for detailed data analysis.

Figure 7 (left) shows a line-rendering in comparison to hyperstreamlines (right) of a pyramidal tract combined with direct volume rendering of a MRI_{T1} dataset.

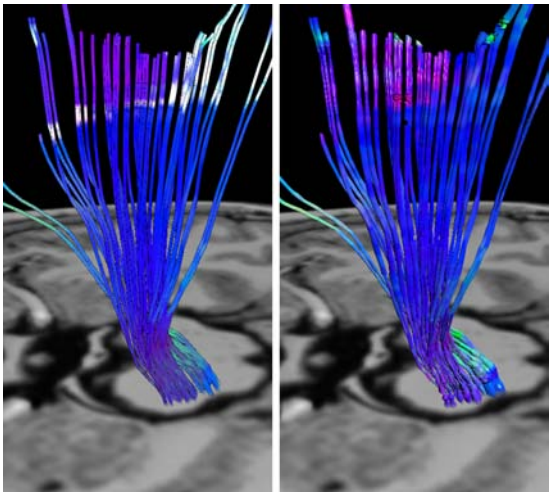

Figure 7: *These two figures show the same pyramidal tract rendered with standard streamline (left) and with our method (right) in combination with direct volume rendering of a T1-weighted MRI dataset. For coloring the principal eigenvectors are mapped into RGB-color space.*

Using hyperstreamlines instead of simple lines enables the analysis of the whole tensor data. Areas with large eigenvalues will result in larger diameters of the hyperstreamline. This allows users to make conclusions about the underlying tissue. For a hyperstreamline showing a higher diameter it is very likely that it is not aligned with a neuronal pathway since white matter restricts the diffusion perpendicular to the cell orientation. Therefore, such expansions are an indication for an uncertainty in the fiber tracking.

The analysis regarding uncertainties can be further improved by the application of special color schemes. Instead of utilizing the standard RGB-scheme, where the principal eigenvector is used as color vector, the color can be selected by a scheme based on an approach presented by Westin *et al.* [WMK*99]. Thereby, areas with high anisotropy are depicted green while areas with planar diffusion are colored yellow and isotropic areas appear red. Accordingly, red tube segments do have a higher uncertainty. Figure 1 shows a whole brain tracking. It can be seen that especially in the end segments the color changes from green to red. This is plausible since the fiber tracking algorithm stops when reaching a voxel with a sufficiently low anisotropy which is depicted red.

Segments which appear yellow correspond to regions of planar diffusion. This occurrence is generally considered to be a potential fiber crossing which cannot be treated adequately with current fiber tracking algorithms. Therefore, such segments are of special interest and supportive rendering is desired. For the actual analysis of such regions hyperstreamlines are superior to common streamlines and -tubes. They provide information about the spatial orientation of the diffusion (Figure 8 *left*). However, hyperstreamlines suffer from the same problem of restricted data accuracy as standard streamlines. Since DTI data does not provide better resolution than the current 2mm voxel spacing, all tracked lines can only be considered averaged models for the underlying tissue structure.

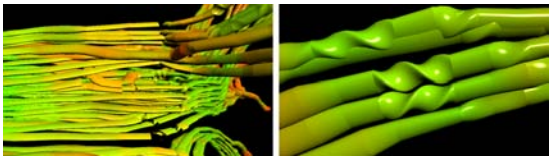

Figure 8: *The left figure shows a bundle of hyperstreamlines traversing a region of planar diffusion which leads to yellow coloring and a flattening of the tube. On the right side, some segments showing extreme torsion are depicted. The displayed extreme torsions are added manually as a proof of concept by switching off angle correction, thus allowing rotations larger than 90 degrees between consecutive ellipses.*

Another feature which is depicted by hyperstreamlines is torsion (Figure 8 *right*). While strong torsion is not necessarily required for DTI visualization it is extremely useful for other data, namely technical simulation data.

7. Performance

Our approach has been integrated into a framework for DTI visualization used at the Neurocenter for easy access to MRI data preprocessing and rendering of correct context information as well as the possibility to compare our method with existing streamline/streamtube implementations. Our method uses GLSL high-level shaders in two variants: the first avoids Shader Model 3 functionality and walks a fixed number of steps before testing for a hit, while the second uses a dynamic loop that stops if either ω can be satisfied or a certain number of steps is exceeded. For current hardware the second approach cannot improve performance, since fragments in a certain neighborhood are required to have the same execution time, so a 'slow' fragment defeats any time-saving neighbors. This variant can however be used for investigating the relation between surface curvature and number of steps needed to intersect the surface reliably (see Figure 9). For practical purposes, a fixed iteration bound of 8 works sufficiently well and yields a performance of about 33 MPixels/s according to *NVShaderPerf*.

Figure 9: *Iteration count to satisfy threshold shown as hue where red means higher iteration count.*

To sensibly compare performance between our method and an existing polygon-based approach, some peculiarities have to be considered. The polygon tubes are subdivided into 16 segments per profile in order for the surface to have comparable shading quality. We use the same number of tube segments in both approaches, however the existing algorithms linearly interpolate between the profiles, which yields connections that are not really correct (see Figure 10). To obtain a high-quality surface like our method, the segments would have to be subdivided in relation to the spanned rotation angle, which would yield at least three times the number of primitives that are used currently and thus reduce rendering performance drastically.

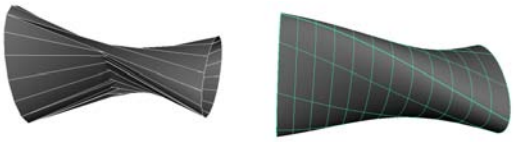

Figure 10: *Interpolated vertex positions (left) with erroneous self-intersection vs. interpolated ellipse orientation (right).*

As can be seen in Table 1, our approach cannot offer the performance of the much simpler polygon-based approach for small and medium-sized datasets. With the whole-brain tracking, however, we are coming close to the break-even point, because the high load of the geometric primitives on the CPU and the graphics bus has about nullified the advantage of cheap fragment processing, so our approach is about twice as fast with a medium-sized viewport on a GeForce 7800 GTX. However, our approach is limited by fragment processing power, which means that using a nearly full-screen-sized viewport reduces our frame rate to half the frame rate achieved by the polygon-based approach. This can be solved by several means though, since fragment processing power can be much more easily increased than throughput or geometry processing: we could either use two SLI-coupled graphics cards for about twice the performance or resort to distributed parallel rendering as often employed in DVR. Taking into account the recent development of graphics cards, we can also rely on the fact that the fragment units of the next generation of graphics cards will at least about double our performance through higher parallelism and other optimizations.

It is also evident in Table 1 that standard direct volume rendering does not significantly impact the performance of our approach. Nevertheless, using the instrumented driver we also found that there still must be a bottleneck in our implementation since the fragment shader load is maxed out at 74% while other raycasting algorithms [RE05] reach up to 91%. We are currently investigating this effect.

	Optic Tract	Pyramidal	Brain
#segments	18,681	34,778	226,032
fps 434×673	10	10	6
fps 434×673 with volume	10	10	5.5
fps 1560×1051	2.5	2.3	1.5
fps 1560×1051 with volume	2.4	2.3	1.5
fps polygons	> 100	> 100	4
fps polygons with volume	15	15	3.3

Table 1: *Performance measured on a GeForce 7800 GTX with instrumented developer driver v79.70. Viewport sizes are as indicated, with the tubes zoomed to fit. A contextual volume rendering is included where mentioned. For polygon-based visualization the viewport size is always 1560×1051, with the volume approximately filling it. Each polygonal segment consists of a triangle strip of 21 elements.*

8. Conclusion

We have demonstrated an alternative method to visualize hyperstreamlines relying on GPU-based iterative ray casting. This method allows us to calculate intersections with surfaces that cannot be implicitly ray cast. The main advantage of this approach is its suitability for large datasets and its inherent scalability with the evolution of graphics processors

and its suitability for parallel rendering methods as SLI or graphics clusters.

Our method can be applied to the visualization of other kinds of unpleasant surfaces and therefore it is of more general interest than the specific application demonstrated here. In the context of medical visualization the enhanced geometry allows the user to benefit from both the connectivity information (usually available from streamlines) and the orientation information (usually available at discrete points by using glyphs) which is visualized as the semi axes of our ellipse tube section. As future work we plan to investigate the possibility to upload the fiber tracking results directly to the GPU and extract our parameters on the fly.

Acknowledgments

This work was partially supported by the Deutsche Forschungsgemeinschaft in the context of SFB 603, Project C9 and partially supported by the DLR in the context of Project 688. We would like to thank Dorit Merhof and Markus Sonntag for various contributions to the visualization framework. Brain dataset courtesy of Christopher Nimsky, MD at the Dept. of Neurosurgery, University of Erlangen-Nuremberg, Germany.

References

[BMP*01] BIHAN D. L., MANGIN J.-F., POUPON C., CLARK C. A., PAPPATA S., MOLKO N., CHABRIAT H.: Diffusion tensor imaging: Concepts and applications. *Journal of Magnetic Resonance Imaging 13* (2001), 534–546.

[BPP*00] BASSER P. J., PAJEVIC S., PIERPAOLI C., DUDA J., ALDROUBI A.: In vivo fiber tractography using dt-mri data. *Magnetic Resonance in Medicine 44* (2000), 625–632.

[DH93] DELMARCELLE T., HESSELINK L.: Visualizing second-order tensor fields with hyperstreamlines. *IEEE Comput. Graph. Appl. 13*, 4 (1993), 25–33.

[Don05] DONNELLY W.: *GPU Gems 2*. Addison-Wesley, 2005, ch. Per-Pixel Displacement Mapping with Distance Functions.

[ERMB00] ELIAS R. MELHEMA RYUTA ITOHA L. J., BARKERA P. B.: Diffusion tensor mr imaging of the brain: Effect of diffusion weighting on trace and anisotropy measurements. *American Journal of Neuroradiology 21* (2000), 1813–1820.

[ESM*05] ENDERS F., SAUBER N., MERHOF D., HASTREITER P., NIMSKY C., STAMMINGER M.: Visualization of white matter tracts with wrapped streamlines. In *Proceedings of IEEE Visualization 2005* (2005), IEEE, pp. 51–58.

[Gum03] GUMHOLD S.: Splatting illuminated ellipsoids with depth correction. In *VMV* (2003), pp. 245–252.

[Har96] HART J. C.: Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer 12*, 10 (December 1996), 527–545.

[KE04] KLEIN T., ERTL T.: Illustrating Magnetic Field Lines using a Discrete Particle Model. In *Workshop on Vision, Modelling, and Visualization VMV '04* (2004).

[Kin04] KINDLMANN G.: Superquadric tensor glyphs. In *Proc. Eurographics - IEEE TCVG Symposium on Visualization* (2004), pp. 147–154.

[KW99] KINDLMANN G., WEINSTEIN D.: Hue-balls and lit-tensors for direct volume rendering of diffusion tensor fields. In *VIS '99: Proceedings of the conference on Visualization '99* (Los Alamitos, CA, USA, 1999), IEEE Computer Society Press, pp. 183–189.

[KWH00] KINDLMANN G., WEINSTEIN D., HART D.: Strategies for direct volume rendering of diffusion tensor fields. *IEEE Transactions on Visualization and Computer Graphics 6*, 2 (2000), 124–138.

[LAS*02] LORI N., AKBUDAK E., SHIMONY J., CULL T., SNYDER A., GUILLORY R., CONTURO T.: Diffusion tensor fiber tracking of human brain connectivity: aquisition methods, reliability analysis and biological results. *NMR in Biomedicine 15*, 7-8 (2002), 494–515.

[LH04] LACZ P., HART J. C.: Procedural Geometric Synthesis on the GPU. In *Proceedings of the GP^2 Workshop* (2004).

[MP03] MĚCH R., PRUSINKIEWICZ P.: Generating subdivision curves with L-systems on a GPU. In *GRAPH '03: Proceedings of the SIGGRAPH 2003 conference on Sketches & applications* (2003), ACM Press, pp. 1–1.

[PAB02] PAJEVIC S., ALDROUBI A., BASSER P.: A continuous tensor field approximation of discrete dt-mri data for extracting microstructural and architectural features of tissue. *Journal of Magnetic Resonance 154*, 1 (2002), 85–100.

[RE05] REINA G., ERTL T.: Hardware-Accelerated Glyphs for Mono- and Dipoles in Molecular Dynamics Visualization. In *Procceedings of EUROGRAPHICS - IEEE VGTC Symposium on Visualization 2005* (2005).

[SGS05] STOLL C., GUMHOLD S., SEIDEL H.: Visualization with stylized line primitives. In *Proceedings of IEEE Visualization '05* (2005), IEEE.

[WL01] WUENSCHE B., LOBB R.: The visualization of diffusion tensor fields in the brain. In *Proc. of the International Conference on Mathematics and Engineering Techniques in Medicine and Biological Science, METMBS* (2001), pp. 498–504.

[WMK*99] WESTIN C.-F., MAIER S. E., KHIDHIR B., EVERETT P., JOLESZ F. A., KIKINIS R.: Image processing for diffusion tensor magnetic resonance imaging. In *Medical Image Computing and Computer-Assisted Intervention* (September 19–22 1999), Lecture Notes in Computer Science, pp. 441–452.

[WMM*02] WESTIN C.-F., MAIER S. E., MAMATA H., NABAVI A., JOLESZ F. A., KIKINIS R.: Processing and visualization of diffusion tensor MRI. *Medical Image Analysis 6*, 2 (2002), 93–108.

[ZDL03] ZHANG S., DEMIRALP C., LAIDLAW D. H.: Visualizing diffusion tensor mr images using streamtubes and streamsurfaces. *IEEE Transactions on Visualization and Computer Graphics 9*, 4 (2003), 454–462.