Interactive fiber structure visualization of the heart

Peeters, T.H.J.M.; Vilanova Bartroli, A.; ter Haar Romenij, B.M.

Published in:
Computer Graphics Forum

DOI:
10.1111/j.1467-8659.2009.01421.x

Published: 01/01/2009

Document Version
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

COMPUTER GRAPHICS forum

DOI: 10.1111/j.1467-8659.2009.01421.x

COMPUTER GRAPHICS forum
Volume 28 (2009), number 8 pp. 2140–2150

Interactive Fibre Structure Visualization of the Heart

T. H. J. M. Peeters, A. Vilanova and B. M. ter Haar Romeny

Department of Biomedical Engineering, Technische Universiteit Eindhoven, Eindhoven, The Netherlands
{T.Peeters, A.Vilanova, B.M.terhaarRomeny}@tue.nl

Abstract

The heart consists of densely packed muscle fibres. The orientation of these fibres can be acquired by using Diffusion Tensor Imaging (DTI) ex vivo. A good way to visualize the fibre structure in a cross section of the heart is by showing short line segments originating from the cross section and aligned with the local direction of the fibres. If the line segments are placed dense enough, one can see how the fibre orientations change. However, generation of the line segments takes time and thus the user has to wait for new geometry to be generated when the plane defining the cross section is changed. We present a new direct rendering method for the visualization of the 3D vector field in a 2D user-definable cross section of a heart. On the intersection of the plane with the vector field, the full 3D vectors are rendered as 3D line segments with a local ray casting approach. No preprocessing of the data is needed and no geometry is generated. This technique allows a fast inspection of the data to identify interesting areas where further analysis is necessary (e.g. quantification or generation of streamlines). We also show how the technique is generalized to other glyph shapes than line segments by implementing ellipsoids.

Keywords: diffusion tensor imaging, heart visualization, glyphs, GPU based

ACM CCS: Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Viewing Algorithms I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism; J.3 [Computer Applications]: Life and Medical Sciences

1. Introduction

The heart is a hollow muscle that pumps blood through the body by repeated, rhythmic contractions. As opposed to skeletal muscle, the heart contracts without being triggered by nerve impulses, and it can work continuously without fatigue. The efficiency of the heart as a pump is the result of the arrangement of the muscle fibres in the heart wall. This cardiac structure is not fully understood and has been a topic of research and discussion for at least a few hundred years. Even today it is a disputed topic [AHR*05]. Heart disorders can cause a change in the fibrous structure of the heart. For example, if a person survives acute cardiac ischemia, commonly known as a heart attack, a wound healing process takes place that changes the structure of the fibres in the infarcted area. Also, in non-ischemic regions, the heart wall can remodel and thicken in order to compensate for the loss of functional muscle fibres in the ischemic areas. Because the heart structure, and changes caused by heart disorders, are not fully understood, research is being done with the purpose of improving our insight in the fibrous structure of both healthy and ischemic hearts. The long-term goal of this research is to improve treatment of cardiac infarction, and to avoid heart failure that occurs when the remodelling of the heart after an infarct is not sufficient to compensate for the physiological needs of the body.

One of the tools that are used to analyze the heart is Diffusion Tensor Imaging (DTI). DTI is an MRI technique that measures the local diffusion of water in tissue. The diffusion in each voxel is represented by a 3×3 symmetric positive-definite tensor. Eigenanalysis can be applied to these tensors. The computed eigenvectors $\vec{e}_1, \vec{e}_2, \vec{e}_3$ and corresponding eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3 > 0$ represent, respectively, the principal diffusion directions and the corresponding diffusion coefficients. These eigenvectors are bi-directional, but for convenience in the rest of this paper, we will refer to them simply as *vectors*. For convenience the eigenvectors are referred to in the rest of the paper as *vectors*. The main diffusion direction $\vec{e}_1$ relates to the local muscle fibre orientation in each voxel of the volume. We use this vector field of $\vec{e}_1$ to visualize the fibre structure of the heart.

DTI is an improvement over conventional histological techniques, because it is not destructive and less labour-intensitive [JPSH04]. However, like histology, heart DTI

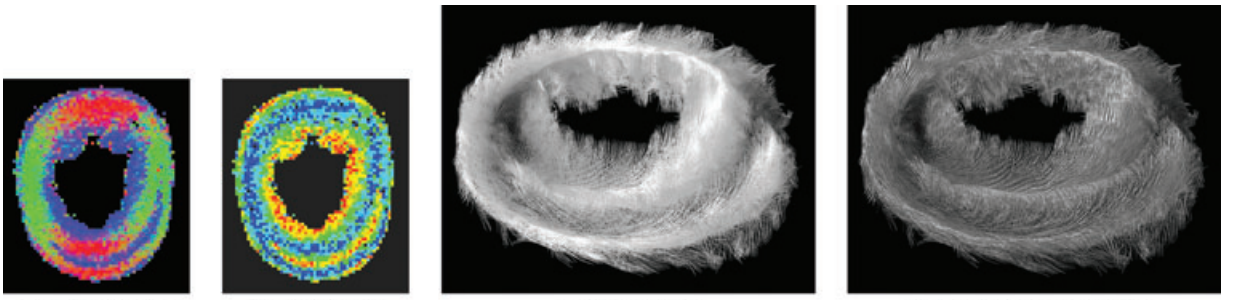

Figure 1: *(a) RGB colouring of fibre orientation. XYZ components of the vectors are mapped directly to the RGB components of the colour. (b) Colour coding of α_h. Blue indicates in-plane fibres, red is out-of-plane. (c), (d) Rendering of tracked fibres in an axial slice of a healthy mouse heart which was scanned ex vivo. In (c) line lighting is used to show the fibres shapes. In (d) the line lighting is supplemented with rendering of shadows to enhance the perception of coherent structures among fibres [PVStHR06].*

cannot be applied *in vivo*. Because the movement of a living heart complicates a DTI scan too much, for heart research we make use of healthy and infarcted mouse and rat hearts which were scanned *ex vivo*. These scans were made for research done to improve our understanding of the structure of the heart and to improve treatment of people recovering from cardiac ischemia.

The application of DTI to the heart muscle is relatively new [JPSH04] and most visualization methods for DTI focus onextracting important structures in the brain by using either tractography [VZKL05] or segmentation [WV05, ZTW06]. These methods cannot be applied to the heart directly because the data is of a different nature. The heart wall consists of a densely packed set of muscle fibres of which the orientation changes gradually throughout the heart wall. Tractography can be used to give a global intuition of the structure of fibres in a healthy heart, or show erratic behaviour in diseased hearts. However, tractography can easily result in a visualization that is too dense and thus suffers from occlusion. Therefore, it should be complemented by an interactive method that shows local and more detailed information. Segmentation is not possible because the fibre orientations in a healthy heart change gradually and no clear borders can be given between different parts of the heart, as is the case in the brain.

A good way to visualize the fibre structure in a cross section of the heart is by showing short lines originating from the cross section and aligned with the local direction of the fibres [PVStHR06]. If the lines are placed dense enough, one can see how the fibre orientations change in a continous way (see Figure 1(d)).

However, in order to make up for the lack of context the user has when showing detailed local information, we propose a new technique that makes it possible for the user to interactively place a plane-of-interest (POI) that defines the cross section from which the line segments originate. With this method the user can quickly browse the data by translating and rotating the POI, and if needed, identify interesting areas where further analysis is done using e.g. quantification or tractography.

In our proposed method, no time is needed for the generation of geometry that will be rendered. On the intersection of the POI with the vector field of $\vec{e}_1$, the full 3D vectors are rendered as 3D line segments with a local ray casting approach. Existing methods that generate densely placed glyphs or short streamlines need to recompute the geometry each time the POI is changed. This causes that action to not be interactive. An additional advantage of our method is that the seeding distance and line length can be changed interactively, without the need to update geometry. Also, most glyphing techniques use less dense placement of the glyphs than what we propose. These techniques are less suitable for visualization of the heart muscle because the gradual changes in fibre orientation cannot be observed easily. We apply line lighting and shadow computations in order to convey the 3D orientation and structure of the vectors to the user. This lighting and shadowing is essential to convey the structure to the user when using very dense seeding. In order to have interactive performance, we implemented both the ray casting and the lighting and shadowing on the GPU.

Our contribution is a new method for interactive visualization of the fibre structure of the heart. We apply local ray casting on the GPU to render line-segment glyphs that represent fibre orientations without generating geometry. Using this method, the user can interactively place the POI, which defines the cross section that will be visualized. We show that our method clearly outperforms geometry-based methods, if the placement of the POI has to be interactive. We also show that the method is general enough to show other glyphs than line segments by implementing ellipsoid glyphs.

In Section 2, we list existing methods for visualizing DTI data and other related techniques. In Section 3, we describe

the method we propose in a general way. In Section 4, we show how to implement the method on the GPU and give implementation details. Our results are given in Section 5. Finally, in Section 6, we summarize our contributions and identify directions for future research.

2. Related Work

Various methods exist for visualizing DTI data. The simplest is to reduce the tensor volume to a volume of scalars by computing an anisotropy index such as, for example, fractional anisotropy (*FA*) [BP96, VZKL05] in each voxel. This scalar volume can then easily be visualized by mapping the scalar values to colours using a colour lookup table. The resulting colours are then shown in a 2D cross section of the volume. The advantage is that this method is easy to implement and fast to render. The disadvantage is that a lot of information is lost. Less information is lost if the tensor field is reduced to the vector field of $\vec{e}_1$ which defines the main diffusion direction in each voxel. A popular way to visualize this field is by slicing the data and mapping the components of $\vec{e}_1$ to RGB colour space (see Figure 1(a)). This colouring can be combined with a weighting, e.g. by *FA*, in order to show more information. However, this visualization is ambiguous (different vectors can have the same colour) and not intuitive.

An often-used scalar index for visualization and quantification of fibre orientations in the heart is the *helix angle* α_h [DS79]. It represents the angle between the fibre direction and the plane perpendicular to the long axis of the heart. The helix angle is visualized in one slice of a heart in Figure 1(b). The major disadvantage of using α_h for visualization is, again, the fact that it cannot show the full 3D vector information. Furthermore, as with the RGB colour coding, colour is not always an intuitive way of representing orientations.

For showing a more global overview of a DTI dataset, several methods for tracking fibres can be used [VZKL05, MCG94]. In brain DTI data, the reconstructed fibres are used as approximations for bundles of axons that connect different parts of the brain. When DTI and fibre tracking are applied to heart data, the reconstructed fibres approximate bundles of muscle fibres and can be used to show the structure of the heart [ZB03]. However, in our application, where we have densely packed fibres in the heart wall, visual clutter will be a problem. Also, preprocessing of the data is needed in order to acquire the streamlines that will be rendered. We want to avoid this step and make the definition of the region-of-interest (ROI) interactive. There are methods where special data structures are used to select precalculated fibres that go through interactively defined ROIs [BBP*05, ASM*04] in the brain. However, the goal and approach in those methods is different from ours. They visualize connectivity in the brain by showing long fibre bundles going through relatively small ROIs, and we show the changes in fibre orientation in the heart by rendering short line segments in larger ROIs.

Dense and texture-based methods such as LIC [CL93] and IBFV [vW02] have been very succesful in visualizing 2D vector fields. Extensions of these 2D techniques have been made to apply them to cross sections of 3D vector fields [LHD*04, SBH99]. However, these methods often project the 3D vectors on a 2D surface [SBH99]. Thus they lose the third dimension of the vectors or suffer from clutter. In our application projecting the 3D vector would easily lead to misinterpretation of the data. True 3D texture-based approaches exist [TvW03, IG97, SFCN02] where 3D textures are generated. However, often this texture generation is not interactive for large datasets (i.e. 512^3) [HA04]. Rendering of the output volume using standard volume rendering techniques can give problems with aliasing, occlusion and the perception of line structures in the volume. There are solutions for these problems such as oversampling the input texture or blurring the output texture [HA04], injecting 'opacity noise' [TvW03] and several shading techniques [HA04, WSE07]. However, in the end the visual result heavily depends on the resolutions of the textures that are used so a balance must be found between texture size and performance. Also, in flow visualization methods that rely on the rendering of a scalar volume, it is difficult to distinguish the individual line structures because only volumetric data is available.

Peeters et al. [PVStHR06] visualize the fibrous structure of the heart by rendering short line segments in the main diffusion directions in one slice of the heart (see Figures 1(c) and (d)). In order to effectively convey the structure of the fibres, very dense seeding is used. Line lighting and shadowing is applied to the line segments, which is essential to show the coherent structure of groups of fibres. The main disadvantage of this method is that many line segments (in the order of 10 K and more) must be generated. Because of this, the user cannot interactively change the POI that defines the positions of the seed points, or other parameters such as seed-point density and line length.

There are methods for interactive and high-quality rendering of glyphs using GPU ray casting [Gum03, SWBG06]. However, those methods focus on geometrically more complex glyphs such as ellipsoids as opposed to our 'simple' line segments. They render a 2D sprite for each glyph that is shown The use of a sprite per-glyph would not be beneficial for the complexity and performance of our rendering. Therefore, we render only one bounding box that contains all our glyphs. Also, we exploit the fact that our densely packed glyphs originate from points on a square grid and determine inside the fragment shader which glyph is visible, while the other approaches use the Z-buffer for this purpose.

The method that we propose was initially inspired by relief mapping [OBM00, POC05]. Relief mapping maps a relief texture to a surface. The relief texture contains a displacement in the direction orthogonal to the surface in each texel. The surface with the relief texture can be rendered in real-time by applying ray casting in the fragment shader on the GPU [POC05].

Although relief mapping inspired us, we cannot use this approach directly. We cannot convert the dense field of lines that we render into a relief map. Relief maps only support displacements orthogonal to the surface, while our line segments can have any orientation. Furthermore, a step-based ray-casting approach would not work for us because we would miss the thin lines that we want to render when taking discrete steps along the view ray. Thus we propose a new method for interactive visualizations of fibre orientations of cross sections of DTI data in the following sections.

3. Ray Casting Vectors in a Plane

Our goal is to render a dense set of simple glyphs (i.e. line segments) that originate from a user-specified cross section defined by a POI. The seed points that are used as the origins for the line segments are implicitly defined by the POI and a user-specified seed-distance d_s.

We do not do any rendering calculations on the CPU. First, the whole vector volume is loaded into the GPU memory as a 3D texture. Then, we render a bounding box around all the line segments originating from the user-defined POI. As a result of this, a fragment shader is called for each of the pixels that potentially has to show a part of the data to be visualized.

For each pixel, we have the following rendering steps.

1. Determine the view-ray $\mathcal{V} = V + \mu\vec{v}$. V is the camera position and $\vec{v}$ is the normalized view direction.
2. Select the seed points **S** on the POI that are in range for $\mathcal{V}$.
3. Compute intersections **I** of the glyphs originating from **S** with view-ray $\mathcal{V}$.
4. Render the glyph with intersection $I \in \mathbf{I}$ closest to the camera position V. This includes lighting and shadow computations.

We know the camera position V and the intersection of the POI and $\mathcal{V}$, so step 1 is straightforward. Steps 2–4 are further explained in Sections 3.1 to 3.3. Algorithm details that depend on the type of glyph (line segments or ellipsoids) are given in Sections 4.2 and 4.3.

3.1. Select seed points in range

The line segments that will render originate from seed points in the POI. The POI is defined by three vertices O, P_1 and P_2 where $\overrightarrow{OP_1}$, is orthogonal to $\overrightarrow{OP_2}$, (see Figure 2). We do not explicitly generate vertices to be used as seed points, but we use the seed distance d_s and place the seed points on a square grid on the POI. In order to select the seed points **S** that have a line segment that potentially intersects the view-ray $\mathcal{V}$ we need to know the intersection point P of $\mathcal{V}$ and the POI, and the angle between view direction $\vec{v}$ and the normal $\vec{n}$ of the POI. This is illustrated in Figure 2.

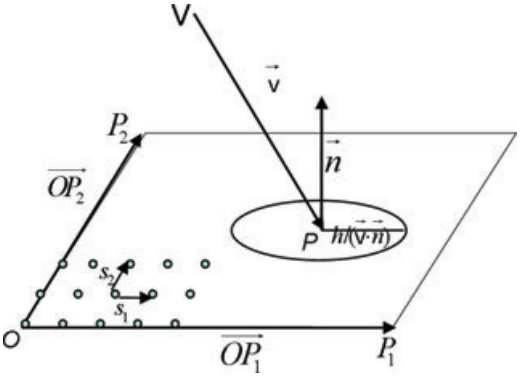

Figure 2: *Illustration of the POI. The figure shows some seed points in the lower-left of the POI. The seed points cover the whole plane. Vectors $\vec{s}_1$ and $\vec{s}_2$ are shown that are used to iterate over all seed points. They are parallel to vectors $\overrightarrow{OP_1}$, $\overrightarrow{OP_2}$ which are also shown. Furthermore, we show camera position V, view direction $\vec{v}$, POI normal $\vec{n}$ and the radius $h/(\vec{v}\cdot\vec{n})$ of the circle to select the seed points in the range of the view-ray.*

For the computation of P, we use the following equation for the POI:

$$\mathcal{P} : n_x\vec{x} + n_y\vec{y} + n_z\vec{z} + d = 0 \tag{1}$$

where $(n_x, n_y, n_y)^T = \vec{n}$ and d is the distance from the POI to the point (0,0,0) in world coordinates. In the intersection point P, we have $(V + \mu\vec{v})\cdot\vec{n} + d = 0$. From this, we can compute μ as follows:

$$\mu = -\frac{\vec{n}\cdot V + d}{\vec{n}\cdot\vec{v}} \tag{2}$$

If $\mathcal{V}$ is parallel to the POI we have $\vec{n}\cdot\vec{v} = 0$ and $\mu = \infty$. However, as we will show further, the range of seed points to be taken into account is also ∞ in this case, so no seed points will be missed.

The length of the glyphs is given by h. So, we only need to take those seed points S into account where the distance $d(S, \mathcal{V})$ between the seed point S and the viewray $\mathcal{V}$ is at most h. We know that for $|\vec{v}\cdot\vec{n}| = 1$, we only need to look at those seed points that are inside the circle on the POI with origin P and radius h. However, if the viewing direction is not orthogonal to the plane, we need to look in a larger area. An initial upper bound for the area of the seed points is a circle with radius $h/(\vec{v}\cdot\vec{n})$. When $\vec{v}\cdot\vec{n} = 0$ the result is ∞. In the implementation this is not a problem because we only look at seed points that are inside the boundaries of the POI and the input tensor field.

We now have initial bounds for which seed points are possibly in the range for $\mathcal{V}$. For each row of seed points, we compute which are the first and last seed point that is range for $\mathcal{V}$. Then we only take those seed points and the ones

in between them into account. Details of this algorithm are given in Section 4.1.

3.2. Compute glyph—view-ray intersections

Next, we need to know which glyph is visible in the current pixel. $\mathbf{S}$ is the collection of seed points with $d(S, \mathcal{V}) \leq h$ for all $S \in \mathbf{S}$. For each seed point $S \in \mathbf{S}$ we run the following algorithm:

- If $\mathcal{V}$ does not intersect glyph $\mathcal{G}(S)$, discard the seed point.
- Otherwise, calculate the intersection point I of $\mathcal{V}$ and $\mathcal{G}(S)$.

We do this for each $S \in \mathbf{S}$ and keep track of the seed point for which $||S - V||$ is the smallest. That is, for which seed point $I = V + \mu\vec{v}$ has the smallest $\mu \geq 0$. Thus, we select that glyph and render it with the proper lighting and shadowing.

3.3. Shadowing

If we render the line segments using a colour that does not depend on its neighbourhood then it becomes impossible to distinguish different line segments that are very close to each other (see Figure 1(c)). We use shadowing to avoid this (see Figure 1(d)). In order to determine for each fragment whether it is in direct light or in shadow, we repeat the algorithms given in Sections 3.1 and 3.2. However, now we use the light-ray $\mathcal{L}$ instead of the view-ray

$$\mathcal{L}(\mu) = L + \mu\vec{l}, \tag{3}$$

where L is the light position, and $\vec{l}$ the light-ray direction. Thus, we replace the view direction $\vec{v}$ by the light direction $\vec{l} = \frac{I-L}{||I-L||}$ and E by L. We again compute the glyph with the smallest distance $||I - L||$ from the intersection to the light source. If we acquire the same line segment as the one that is the closest to V in the current fragment, then it is directly lighted. Otherwise, there is another line segment in between the light source and the one that we are currently rendering. In that case, it is in shadow.

In the lighting equation, we use lower values for the ambient and diffuse light coefficients for fragments that are in shadow to make the shadowed fragments darker. Also, we set the specular component to 0 to avoid specular highlights in shadow.

4. Algorithm Details

We implemented our method as a *mapper* in the Visualization ToolKit (VTK) [SML04] and integrated it in with our DTI-visualization tool called DTITool that is used by our collaborators to do heart and brain research. Because of this integration, we can combine our new method with other visualizations such as colour-coded planes and fibre tracking, which is shown in Figures 3–5. The mapper has as input the vector volume, and a vtkPlaneWidget. The vtkPlaneWidget defines the POI and can be interactively rotated, translated and scaled by the user. Furthermore, the user can set the seed distance d_s. The shader programs that run on the GPU are written in OpenGL Shading Language (GLSL).

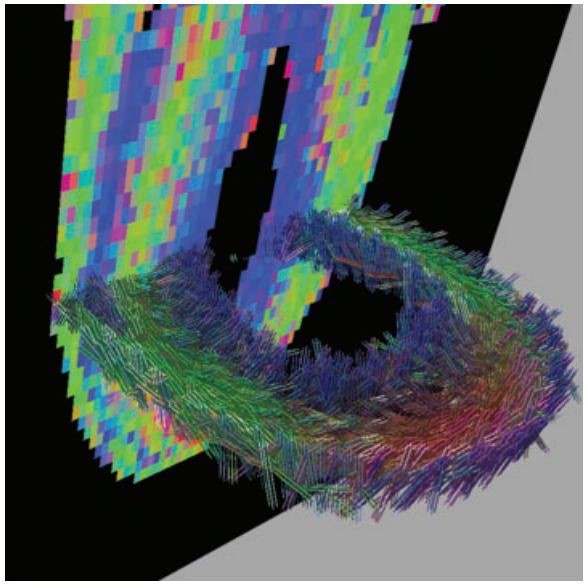

Figure 3: *Our new rendering method showing an axial cross section of a healthy mouse heart. RGB colouring of fibre orientation was applied to the fibres and to the textured plane showing a coronal cross section in the background.*

First, we do eigenalysis to compute the eigenvectors. We then load the main vectors in GPU memory as a 3D RGB float texture. The XYZ-components of the vectors are stored in the RGB-components of the texture. To avoid interpolation problems, on the GPU, we make use of nearest neighbour interpolation of the vectors.

In Section 4.1, we give implementation details about how relevant seed points are selected in the fragment shader. In Section 4.2, we explain how to do intersection and lighting for line segments. In order to show that our method is easily adapted to other glyphs than line segments, in Section 4.3, we show how to do the intersection and lighting for ellipsoids.

4.1. Seed-point selection

For the representation of the POI as described in Section 3.1 we pass the origin O and two points P_1 and P_2 with orthogonal vectors $\overrightarrow{OP}_1$, and $\overrightarrow{OP}_2$ (see Figure 2) as uniform variables to the shaders. We also compute seeding step vectors $\vec{s}_1, \vec{s}_2$ parallel to respectively $\overrightarrow{OP}_1$, and $\overrightarrow{OP}_2$ with length d_s that will be used to go from one seeding position to the next.

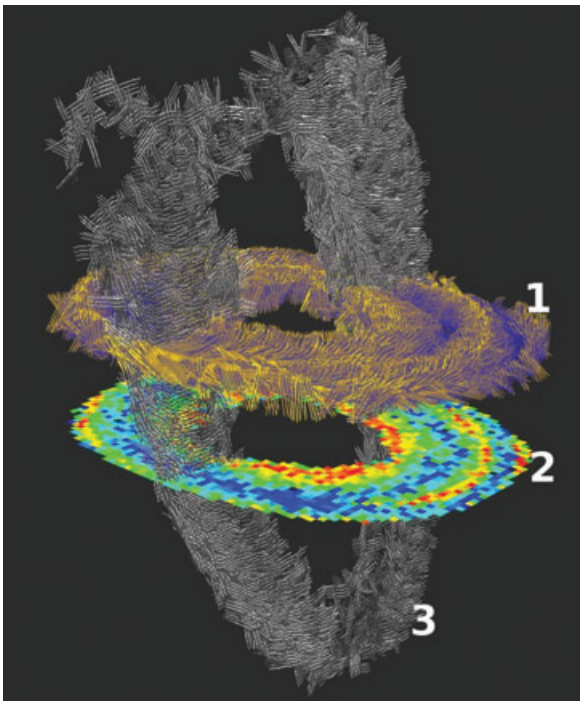

Figure 4: *Three cross sections of a healthy mouse heart visualized with three different methods. (1) Short line segments rendered as geometry. (2) Colour-coding of α_h shown as a texture on a plane. (3) Short line segments rendered with our new method.*

We showed in Section 3.1 that the seed points with a distance larger than $h/(\vec{v}\cdot\vec{n})$ to the intersection P of the view ray and the POI do not need be taken into account. In the fragment shader, we use this to compute the number of steps in directions $\vec{s_1}$, $\vec{s_2}$, $-\vec{s_1}$ and $-\vec{s_2}$ that need to be taken from the seed point closest to P in order to iterate over all the seed points by:

$$NumSteps = \left\lfloor \frac{h/d_s}{\vec{n}\cdot\vec{v}} \right\rfloor \quad \text{if } \vec{n}\cdot\vec{v} \neq 0$$
$$NumSteps = \infty \quad \text{if } \vec{n}\cdot\vec{v} = 0.$$

Each seed point S can be written as $S = O + i\vec{s_1} + j\vec{s_2}$ with integer values i and j. The initial ranges for i and j are given by $i \in [min_i, max_i]$, $j \in [min_j, max_j]$ as determined by the calculation of *NumSteps* described earlier, and bounded by the size of the POI and of the input volume. We iterate over the selected seed points in a nested loop. To avoid handling points that are out-of-range for the viewray $\mathcal{V}$, in the inner loop, we compute the range for i to have only seed points with a distance of at most h to $\mathcal{V}$. We define $\mathcal{C}$ as the cylinder with axis $\mathcal{V}$ and radius h (see Figure 6). Then, we iterate over the seed points that are in range as follows:

```
float μrange[2]; float irange[2];
for (j = min_j; j ≤ max_j; j = j + 1)
  {
  point R = O + j * s2;
  line L(μ) = R + μ s1/|s1|;
  float d = minimal distance between V and L;
  float μ_d = argument of L(μ) in the point
    where L is closest to V.
  If (d ≤ h)
    {
    \\Compute the values of μ where L intersects C:
    vec n = (s1 × v)/|s1 × v|;
    vec o = (n × s1)/|n × s1|;
    float t = (h^2 − d^2)/(v · o);
    float μrange[2] = {μ_d − t, μ_d + t};
    If there is only one intersection,
      then t = 0 and μrange[0] = μrange[1].
    irange[0] = max(min_i, ceil(μrange[0]));
    irange[1] = min(max_i, floor(μrange[1]));
    for (i = irange[0]; i ≤ irange[1]; i = i + 1)
      {
      Handle seed point S = R + i * s1 + j * s2;
}}}
```

Using this algorithm, texture lookups and distance calculations are only done for the seed points with a distance of at most h to the view-ray $\mathcal{V}$. Thus, we use the optimal search area for seed points that can be the origin of glyphs that intersect $\mathcal{V}$.

4.2. Intersection and lighting of line segments

Because lines are infinitesimally thin, the chance that a line intersects with the view ray is infinitesimally small. Therefore, we assign a thickness r to the lines, where r depends on the distance to V. Because the lines that we render have a length h and are not infinitely long, we also incorporate h in our algorithm. In order to find the line segment that is visible in the current fragment, for each seed point S in the set of preselected seed points $\mathbf{S}$, we run the following algorithm:

- $\vec{w}$ is the eigenvector at position S of the input volume
- Compute the closest distance d between line $\mathcal{W}(\mu) = S + \mu\vec{w}$ and $\mathcal{V}$. The algorithm is given in Appendix A.
- If $d > r$ then the line can be discarded.
- If $d \leq r$, but not in a point Q where $||Q - S|| \leq h$, then the line can also be discarded because it would need a length larger than h to be visible.
- Otherwise, the current line is considered to intersect the view ray.

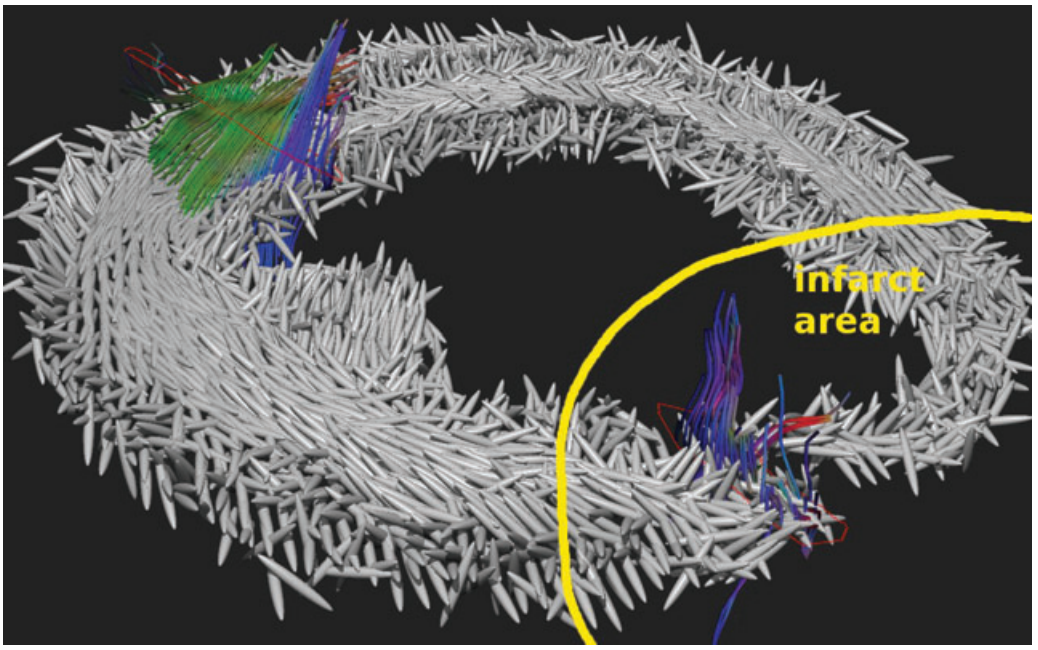

Figure 5: *Our new rendering method using ellipsoids with fixed shape to show the fibre orientations in the cross section of an infarcted heart, which was scanned 28 days after the infarct. We also tracked fibres from two different seeding areas. The resulting fibres were rendered as thin tubes and RGB colouring of the fibre orientation was used.*

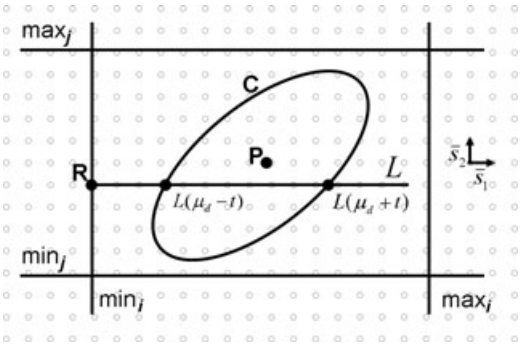

Figure 6: *Illustration of the seed-point selection. Point P is the intersection of the view-ray with the POI and ellipsoid C is the intersection of the POI with cylinder C. Line L is defined by $R + \mu\vec{s}_1/|\vec{s}_1|$. The small grey circles denote seed-point locations.*

The line that will be visible in the current pixel is the line that has the intersection I with the view ray that is the closest to the camera position V. Thus, we select that line and render it with the proper lighting and shadowing.

In the Phong lighting model [Pho75], the light intensity g at a point on a surface, follows the equation:

$$g = k_a + k_d(\vec{l} \cdot \vec{n}) + k_s(\vec{v} \cdot \vec{r})^p. \quad (4)$$

The material-specific values of k_a, k_d, k_s and p are the ambient, diffuse and specular coefficients, and the specular component or shininess. Vector $\vec{n}$ is the normal at the surface point. $\vec{l}$ is the direction of the light, $\vec{v}$ is the view direction, and $\vec{r}$ is the reflection of $\vec{l}$ at $\vec{n}$. Vectors $\vec{n}$, $\vec{l}$, $\vec{v}$, and $\vec{r}$ have unit length.

This model cannot be applied to illuminate lines directly, because lines do not have a single normal $\vec{n}$, but a plane of normals perpendicular to the tangent direction $\vec{w}$. This problem can be resolved by choosing for $\vec{n}$ the vector in the normal plane that maximizes $(\vec{l} \cdot \vec{n})$ and $(\vec{v} \cdot \vec{r})$ in Equation (4). To avoid explicit calculation of the optimal $\vec{n}$, the following equations can be used [SZH97]:

$$\vec{l} \cdot \vec{n} = \sqrt{1 - (\vec{l} \cdot \vec{w})^2} \quad (5)$$

$$\vec{v} \cdot \vec{r} = (\vec{l} \cdot \vec{n})\sqrt{1 - (\vec{v} \cdot \vec{w})^2} - (\vec{l} \cdot \vec{w})(\vec{v} \cdot \vec{w}) \quad (6)$$

Using Equations (5) and (6), the calculation of g in Equation (4) is implemented directly in the fragment shader. We also update the current fragment depth, which is stored in the Z-buffer, such that our rendered line segments integrate correctly with other objects in the scene.

4.3. Intersection and lighting of ellipsoids

In order to show that our method is general enough to deal with other glyph-shapes than line-segments, we implemented

ellipsoid glyphs. Later, we explain how to handle view-ray–glyph intersection and lighting if we have ellipsoid glyphs. Other parts of the algorithm do not depend on the type of glyphs that we choose to render.

The equation of an axes-aligned ellipsoid is

$$\frac{x^2}{r_x^2} + \frac{y^2}{r_y^2} + \frac{z^2}{r_z^2} = 1, \tag{7}$$

where $r_{x,y,z}$ are the radii of the ellipsoid in the x, y and z-directions. In order to compute the intersection of the view-ray $\mathcal{V}(\mu) = V + \mu\vec{v}$ with the ellipsoid, we fill in $V + \mu\vec{v}$ for $(x,\ y,\ z)^{\mathrm{T}}$. The resulting equation can be solved using the quadratic formula

$$\mu = \frac{-b \pm \sqrt{D}}{2a}, \text{ where } D = b^2 - 4ac \tag{8}$$

where

$$a = \sum_{i \in \{x,y,z\}} r_i \vec{v}_i^2 \tag{9}$$

$$b = \sum_{i \in \{x,y,z\}} 2r_i(V_i - S_i)\vec{v}_i \tag{10}$$

$$c = \sum_{i \in \{x,y,z\}} r(V_i - S_i)^2, \tag{11}$$

where S is the centre of the ellipsoid. The number of solutions depends on the value of D. For $D < 0$ there is no intersection. For $D \geq 0$, the closest intersection has $\mu = (b^2 - \sqrt{D})/2a$.

Here, we render ellipsoids with fixed radii $(r_x,\ r_y,\ r_z) = (0.8,\ 0.1,\ 0.1)$, and the long axis aligned with the vector given at seed-point positions S. In order to solve $\mathcal{V} = \mathcal{G}(S)$, where $\mathcal{G}(S)$ defines the ellipsoid, we transform view ray $\mathcal{V} = V + \mu\vec{v}$ into the local coordinate system by multiplying $\vec{v}$ and V (relative to S) with rotation matrix R_S, which aligns the x-axis with the vector in S. Then, we solve equation (8). We can fill in the computed μ directly in the original equation for $\mathcal{V}$ to compute the intersection point in world coordinates.

In order to compute the normal $\vec{n}_e$ of the ellipsoid in the intersection point I, we first compute the vector pointing from S to I. This would be the proper normal $\vec{n}_s$, if we were dealing with a simple sphere. However, because we have an ellipsoid, we compute the derivative of $(x^2/r_x + y^2/r_y + z^2/r_z - 1)$ in the local coordinate system

$$\vec{n}_e = R_S^{\mathrm{T}}(2M(R_S\vec{n}_s)), \tag{12}$$

where M is the diagonal matrix of the inverse ellipsoid radii, thus $M_{ii} = 1/r_i$ for $i \in \{x,\ y,\ z\}$ and $M_{ij} = 0$ if $i \neq j$. R_S^{T} is the transpose of rotation matrix R_S. Vector $\vec{n}_e$ can be used in standard lighting calculations.

5. Results

We applied the proposed visualization to a series of healthy and infarcted mouse hearts. Four datasets were available of healthy hearts. For infarcted hearts we had 5, 4 and 5 datasets measured, respectively, at 7, 14 and 28 days after infarction. Each heart is only a few millimeters long and the scanning resolution was $117\,\mu m \times 117\,\mu m \times 234\,\mu m$. Each dataset has $128 \times 128 \times 64$ voxels. We also used 7 healthy rat heart datasets with dimensions varying from $64 \times 64 \times 128$ to $96 \times 96 \times 128$ voxels.

We visualized the data with our proposed technique and compared it with a previous method that generates geometry [PVStHR06]. First, we analyze the visual results in Section 5.1 and then we give performance measurements and a comparison in Section 5.2.

5.1. Visual aspects

Figure 3 shows a short-axis cross section of a healthy mouse heart. The visualization shows how the fibre orientation changes in the heart wall. We applied RGB colouring of the fibre orientation to the fibres. We also applied it to the textured plane showing a coronal cross section in the background. The user can enable the plane widget that can be used to modify the POI by translating and rotating it. The line segments originating from the POI are immediately visible while interacting with the POI. For 2D images, coloured cross sections can be more clear, but when the user has the possibility to interact with the scene, our visualization conveys the fibre structure in a more intuitive way.

The results of the proposed method look the same as the method that generates geometry [PVStHR06]. To illustrate this, Figure 4 shows a cross section using geometry, as well as a cross section rendered using our new method. The topmost short-axis cross section (1) uses short line segments that were rendered as geometry. The bottom short-axis slice (2) shows fibre orientations as colours using α_h colouring (as described in Section 2). Note that for neither method shadows cast by other geometry is supported. The third slice (3) was placed freely using our interactive POI and is close to a long-axis cross section. It was rendered using our proposed method. In order to distinguish the two methods for rendering line segments, we applied tone shading to the geometry-based rendering to give it a different colour. The seeding distance and fibre length was the same for both methods.

Besides a comparison of the two methods, Figure 4 also shows how our new method integrates well with geometry. Because for each fragment that we render, we send the correct depth to the depth buffer, depth-based visibility is automatically dealt with by the GPU. This Figure shows that it combines well with line segments and with a textured plane that is transparent where no fibres are present. But it works for any geometry that is rendered in the same scene.

Table 1: *Performance measurements for rendering line segments and ellipsoids in several datasets with different methods.*

Method	Dataset	Number of seeds	h	Generate geometry (s)	Render performance (FPS)	Figure
Ray cast lines	Mouseheart	10 K	1.0	–	35–50	3
Geometry lines	Mouseheart	10 K	1.0	5	28–30	1(d)
Ray cast lines	Mouseheart	27 K	1.0	–	30–40	4
Geometry lines	Mouseheart	27 K	1.0	14	25–28	–
Ray cast ellipse	Infarcted	8 K	3.0	–	20–35	5
Geometry ellipse	Infarcted	8 K	3.0	10	10–12	–

Note: Rendering was done on a PC with an Intel Pentium 4 3.20 GHz CPU, 3 GB RAM and a GeForce 8800 GTX graphics card with 768 MB of memory. The rendering viewport was 1024 × 768 pixels. The mouseheart and infarcted datasets both have dimensions of 128 × 128 × 64 voxels. The generated ellipses for method 'geometry ellipse' have 64 vertices per glyph. The value of h is the length of the line segments or the largest radius of the ellipsoids.

Figure 5 shows a short-axis slice of an infarcted mouse heart. The fibre orientations were rendered using ellipsoids as described in Section 4.3. An exact border for the infarct cannot be given, but in the infarcted area the heart wall is thinner than normally. Also, the fibre orientations are less structured. We show this by visualizing fibres tracked using streamline tracing in a healthy and an infarcted area. The difference in how well aligned fibre orientations are locally is also visible from the ellipsoids that we render. In the infarcted area (lower-right) there is no apparent structure, while in, e.g. the left of the image the ellipsoids are nicely aligned and their orientation changes smoothly when going through the heart wall.

5.2. Performance

We compared the performance of the proposed method to the performance of a method where geometry is generated. The results are shown in Table 1. The measurements were taken such that the whole rendered scene is visible in the current viewport, and covering more than half the screen area (similar to what is shown in the figures). If we zoom very far such that only a few dozen glyphs fill the viewport, the performance drops because coverage of screen space by the glyphs increases, but not lower than 10–15 FPS. When zooming out the performance increases, with an upper limit of about 100 FPS. Performance doubles if shadows are disabled.

Although the methods we used for generating and rendering geometry were not optimized, it can be seen that the two different rendering approaches are competitive when it comes to rendering performance. However, if the user wants to change the POI or properties of the lines or ellipsoids that are being rendered, then new geometry needs to be generated which currently takes a waiting time in the order of seconds. With our proposed ray casting method, this step is not needed so we clearly outperform geometry-based methods there.

The performance of the geometry-based method can be improved, for example by making use of geometry shaders. However, the seed points will still need to be generated on CPU. Also, for rendering ellipsoids with a high visual quality, a lot of vertices are needed and thus a geometry-based approach will not outperform our ray-casting approach.

6. Conclusions and Future Work

Our main contribution is a new GPU-based ray casting technique for interactively visualizing cross sections of the heart, which consists of densely-packed muscle fibres. This cross section can be chosen interactively by the user by moving and rotating a POI. In the POI, the full 3D fibre orientations are visualized as short lines or ellipsoids, using proper lighting and shadowing. This enables the user to quickly inspect a volume of vectors derived from a DTI scan of mouse hearts. For this application, it is very important that the seeding is very dense in order to show the gradual change in fibre orientation througout the heart wall. It is also important that the user can interactively place the POI. The proposed method outperforms the approach where geometry is generated, and can be used for fast inspection of the data to identify interesting areas where further analysis is necessary. It can be used, for example, to select areas where quantification (e.g. statistics of fibre orientation or *FA* in healthy vs. infarcted areas) is done or where to place seed points to initiate fibre tracking. An additional advantage of our technique is that it allows interactive changes in parameters, such as seeding density and line-segment length, without the need to generate geometry.

It would be a useful extension to support triangles to define the cross section instead of the rectangular plane that we have now. Because our visualization integrates well with other rendered objects in the scene, this would allow for visualization of any triangulated surface with our method. The main challenge with this extension is, however, to define a meaningful surface in the heart-wall that we could visualize.

Another possible extension is to show more information in the glyphs. In DTI data, the diffusion tensors contain more

information than fibre orientations. It is possible to show that information using, for example, colour, line-segment length, or ellipsoid shape. However, the main challenge in showing additional information, especially if the seeding is very dense, is how to convey such large amounts of information to the user in an effective way such that important patterns in the data become visible.

Acknowledgements

We thank the Biomedical NMR group, and especially Gustav Strijkers and Annemiek Bouts, of the Department of Biomedical Engineering of the Technische Universiteit Eindhoven for the datasets. This study was financially supported by the Dutch BSIK program entitled Molecular Imaging of Ischemic heart disease (project number BSIK 03033). Part of this work was supported by the VENI program of the Netherlands Organisation for Scientific Research (NWO).

References

[AHR*05] Anderson R., Ho S., Redmann K., Sanchez-quintana D., Lunkenheimer P.: The anatomical arrangement of the myocardial cells making up the ventricular mass. *European Journal of Cardio-Thoracic Surgery 28* (2005), 517–525.

[ASM*04] Akers D., Sherbondy A., Mackenzie R., Dougherty R., Wandell B.: Exploration of the brain's white matter pathways with dynamic queries. In *Proceedings of IEEE Visualization 2004* (2004), IEEE Computer Society, pp. 377–384.

[BBP*05] Blaas J., Botha C., Peters B., Vos F., Post F.: Fast and reproducible fiber bundle selection in DTI visualization. In *Proceedings of IEEE Visualization 2005* (2005), pp. 59–64.

[BP96] Basser P. J., Pierpaoli C.: Microstructural and physiological features of tissues elucidated by quantitative-diffusion-tensor MRI. *Journal of Magnetic Resonance B 111*, 3 (1996), 209–219.

[CL93] Cabral B., Leedom L.: Imaging vector fields using line integral convolution. In *Proceedings of ACM SIGGRAPH '93* (1993), ACM, pp. 263–272.

[DS79] Streeter J. D. D.: Gross morphology and fiber geometry of the heart. In *Handbook of physiology—The Cardiovascular System vol. 1* (1979), Berne R. (Ed.), Williams and Wilkins, pp. 61–112.

[Gum03] Gumhold S.: Splatting illuminated ellipsoids with depth correction. In *VMV* (2003), Ertl T. (Ed.), Aka GmbH, pp. 245–252.

[HA04] Helgeland A., Andreassen O.: Visualization of vector fields using seed lic and volume rendering. *IEEE Transactions on Visualization and Computer Graphics 10*, 6 (2004), 673–682.

[IG97] Interrante V., Grosch C.: Strategies for effectively visualizing 3d flow with volume lic. In *VIS '97: Proceedings of the 8th conference on Visualization '97* (Los Alamitos, CA, USA, 1997), IEEE Computer Society Press, pp. 421–424.

[JPSH04] Jiang Y., Pandya K., Smithies O., Hsu E.: Three-dimensional diffusion tensor microscopy of fixed mouse hearts. *Magnetic Resonance in Medicine 52* (2004), 453–460.

[LHD*04] Laramee R. S., Hauser H., Doleisch H., Vrolijk B., Post F. H., Weiskopf D.: The state of the art in flow visualisation: Dense and texture-based techniques. *Computer Graphics Forum* (2004), 203–221.

[MCG94] Max N., Crawfis R., Grant C.: Visualizing 3d velocity fields near contour surfaces. In *VIS '94: Proceedings of the conference on Visualization '94* (Los Alamitos, CA, USA, 1994), IEEE Computer Society Press, pp. 248–255.

[OBM00] Oliveira M. M., Bishop G., Mcallister D.: Relief texture mapping. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2000), ACM Press/Addison-Wesley Publishing Co., pp. 359–368.

[Pho75] Phong B. T.: Illumination for computer generated pictures. *Commun. ACM 18*, 6 (1975), 311–317.

[POC05] Policarpo F., Oliveira M. M., Comba J. L. D.: Real-time relief mapping on arbitrary polygonal surfaces. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2005), ACM, pp. 155–162.

[PVStHR06] Peeters T., Vilanova A., Strijkers G., ter haar romeny B.: Visualization of the fibrous structure of the heart. In *Proceedings of Vision Modeling and Visualization 2006* (2006), Kobbelt L., Kuhlen T., Aach T., Westermann R. (Eds.), pp. 309–316.

[SBH99] Scheuermann G., Burbach H., Hagen H.: Visualizing planar vector fields with normal component using line integral convolution. In *VIS '99: Proceedings of the conference on Visualization '99* (Los Alamitos, CA, USA, 1999), IEEE Computer Society Press, pp. 255–261.

[SE02] Schneider P. J., Eberly D.: *Geometric Tools for Computer Graphics*. Elsevier Science Inc., New York, NY, USA, 2002.

[SFCN02] Suzuki Y., Fujishiro I., Chen L., Nakamura H.: Case study: hardware-accelerated selective lic volume rendering. In *VIS '02: Proceedings of the conference on Visualization '02*. Washington, DC, USA, 2002, IEEE Computer Society, pp. 485–488.

[SML04] Schroeder W., Martin K., Lorensen B.: *The Visualization Toolkit, Third Edition*. Kitware Inc., 2004.

[SWBG06] Sigg C., Weyrich T., Botsch M., Gross M.: GPU-based ray-casting of quadratic surfaces. In *Eurographics Symposium on Point-Based Graphics*. Botsch M., Chen B. (Eds.) (2006), pp. 59–65.

[SZH97] Stalling D., Zockler M., Hege H.-C.: Fast display of illuminated field lines. *IEEE Transactions on Visualization and Computer Graphics 3*, 2 (1997), 118–128.

[TvW03] Telea A., van wijk J. J.: 3D IBFV: Hardware-accelerated 3d flow visualization. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (Washington, DC, USA, 2003), IEEE Computer Society, pp. 233–240.

[vW02] van Wijk J. J.: Image based flow visualization. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2002), ACM, pp. 745–754.

[VZKL05] Vilanova A., Zhang S., Kindlmann G., Laidlaw D.: An introduction to visualization of diffusion tensor imaging and its applications. In *Visualization and Processing of Tensor Fields*. Weickert J., Hagen H. (Eds.), Mathematics and Visualization, Springer-Verlag, Berlin, 2005, ch. 7, pp. 121–153.

[WSE07] Weiskopf D., Schafhitzel T., Ertl T.: Texture-based visualization of unsteady 3d flow by real-time advection and volumetric illumination. *IEEE Transactions on Visualization and Computer Graphics 13*, 3 (2007), 569–582.

[WV05] Wang Z., Vemuri B.: DTI segmentation using an information theoretic tensor dissimilarity measure. *IEEE Transactions on Medical Imaging 24*, 10 (2005), 1267–1277.

[ZB03] Zhukov L., Barr A. H.: Heart-muscle fiber reconstruction from diffusion tensor MRI. In *Proceedings of IEEE Visualization 2003* (2003), IEEE Computer Society, pp. 597–602.

[ZTW06] Ziyan U., Tuch D., Westin C.: Segmentation of thalamic nuclei from DTI using spectral clustering. In *Ninth International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI'06)* (Copenhagen, Denmark, October 2006), Lecture Notes in Computer Science 4191, pp. 807–814.

Appendix A: Closest Points Between Two Lines

The closest points of two lines $\mathcal{L}_0 = P_0 + s\vec{d}_0$ and $\mathcal{L}_1 = P_1 + t\vec{d}_1$ can be calculated as follows [SE02]:

$\vec{u} = P_0 - P_1$;
$a = \vec{d}_0 \cdot \vec{d}_0$; $b = \vec{d}_0 \cdot \vec{d}_1$; $c = \vec{d}_1 \cdot \vec{d}_1$;
$d = \vec{d}_0 \cdot \vec{u}$; $e = \vec{d}_1 \cdot \vec{u}$; $f = \vec{u} \cdot \vec{u}$;
$g = ac - bb$;
\\Check for (near) paralellism
if $(g < \varepsilon)$ { \\ Small ε
  $s = 0$;
  \\ Choose largest denominator
  \\ to minimize numerical errors
  if $(b > c)$ $t = d/b$;
  else $t = e/c$;
} else {
  $s = (be - cd)/g$;
  $t = (ae - bd)/g$;
}

By filling out s and t in the equations of $\mathcal{L}_0, \mathcal{L}_1$ we can compute the closest points Q_0, Q_1 of the two lines, and their distance $d_q = ||Q_1 - Q_0||$. In our case, if $\mathcal{L}_1$ is the line that we want to render, we only take the line into account if $t(\vec{d}_1 \cdot \vec{d}_1) \leq h$ and if $d_q \leq r$.

Supporting Information

Additional Supporting Information may be found in the online version of this article:

Video Clip S1. The video clip is an .avi file.

Please note: Blackwell Publishing are not responsible for the content or functionality of any supporting materials supplied by the authors. Any queries (other than missing material) should be directed to the corresponding author for the article.