Stochastic DT-MRI Connectivity Mapping on the GPU

Tim McGraw, *Member, IEEE* and Mariappan Nadar, *Member, IEEE*

Abstract—We present a method for stochastic fiber tract mapping from diffusion tensor MRI (DT-MRI) implemented on graphics hardware. From the simulated fibers we compute a connectivity map that gives an indication of the probability that two points in the dataset are connected by a neuronal fiber path. A Bayesian formulation of the fiber model is given and it is shown that the inversion method can be used to construct plausible connectivity. An implementation of this fiber model on the graphics processing unit (GPU) is presented. Since the fiber paths can be stochastically generated independently of one another, the algorithm is highly parallelizable. This allows us to exploit the data-parallel nature of the GPU fragment processors. We also present a framework for the connectivity computation on the GPU. Our implementation allows the user to interactively select regions of interest and observe the evolving connectivity results during computation. Results are presented from the stochastic generation of over 250,000 fiber steps per iteration at interactive frame rates on consumer-grade graphics hardware.

Index Terms—diffusion tensor, magnetic resonance imaging, stochastic tractography.

1 Introduction

Many neurological disorders are characterized by changes in brain white-matter connectivity, for example stroke [56], trauma [54], and multiple sclerosis [55]. Additionally, presurgical planning for epilepsy and tumor resection can incorporate connectivity information [35]. DT-MRI makes it possible to compute, in vivo, many useful quantities, including estimates of structural connectivity within neural tissue [1, 3, 40]. Much clinical research is based on the use of pointwise indices such as diffusion anisotropy, mean diffusivity, or sparse connectivity matrices computed from diffusion weighted images. The next vertical step in the processing of DT-MRI is to analyze the full connectivity map: the connectivity between all pairs of points in the image. Fast computation and display of connectivity information will significantly advance the clinical usefulness of DT-MRI. Nonfocal effects, such as those secondary to diffuse axonal injury, could also be studied using these methods. Since connectivity may be affected after injury and far from the site of injury, studying global connectivity measures is well justified. Reduced computation time will make these methods more attractive for use in a time critical settings, such as assessment of brain injury following stroke or trauma.

1.1 DT-MRI Visualization

The challenge of DT-MRI visualization is to simultaneously convey as much relevant information as possible: mean diffusivity, principal diffusion direction (PDD), anisotropy, and oblateness/prolateness of the diffusion ellipsoid. Many of these quantities can be computed from the elements of the tensor, D, at each voxel or from the eigenvalue decomposition of D: the PDD is the dominant eigenvector of D, fractional anisotropy (FA) is the normalized variance of the eigenvalues, mean diffusivity is the trace of D. We will categorize and describe the most common tensor field visualization techniques here. Glyph-based visualization relies on a small graphical icon at each voxel to represent a tensor. For example, ellipsoids can be computed by transforming the vertices of a triangulated sphere by the diffusion tensor (Figure 1). However, the appearance of these glyphs can be uninformative at some viewing angles. Kindlmann et al. [16] used superquadric glyphs to overcome this visual ambiguity. A composite glyph was introduced by Westin et al. [57] to address the same problem. Kindlmann and Westin [17] presented a method of optimally placing glyphs to emphasize the structure of the data. In order to emphasize spatial continuity of the PDD, and the supposed direction of underlying neuronal fibers, streamlines and streamtubes are used to visualize diffusion information. Streamlines are curves which are tangent to the PDD vector field at each point along the curve (also called integral curves). Streamtubes are cylindrical surfaces whose axis is a streamline. Although these are vector field visualization techniques, they can be adapted to reflect additional information about the tensor field. Song et al. [59] displayed streamtubes and streamsurfaces of the PDD field. Moberts et al. [33] evaluate several schemes for clustering of fiber paths so that similar tracts can be colored similarly. Enders et al. [11] render bundles of fibers by computing the surface enveloping the fibers. Particles are another discrete technique for representing a tensor field. In this case, the icon or glyph representing the tensor is not stationary, but advects through the PDD field, changing its appearance to reflect the tensor at its current position. Kondratieva et al.[21] used the GPU to accelerate the rendering and animation of particles for tensor field visualization.

Glyph based techniques suffer from some problems. For large datasets the display can become too dense. In 3D datasets the discrete glyphs and streamtubes often obscure each other. The local structure that they convey can become lost. To overcome this problem, traditional volume visualization techniques [10, 28] such as raycasting and splatting can be applied to a field of scalar indices of the tensors, such as FA or tensor trace. These techniques reveal more large scale structure than the glyph displays can. Kindlmann et al. [18] applied these direct volume rendering concepts to scalar quantities computed from tensor-valued data. Texture-based visualization techniques produce an image in which texture orientation and frequency reflect the tensor data. Line integral convolution (LIC), introduced by Cabral and Leedom [7], is a process of blurring or convolving a noise image with a curvilinear kernel aligned with the local streamline through each voxel. The resulting image has highly correlated intensity values along each streamline, and uncorrelated intensity across streamlines. Hsu [14] first applied LIC to DT-MRI images of the myocardium. McGraw et al. [31] extended this technique by combining the LIC texture, a color computed from the PDD field, and the FA image to produce an image reflecting diffusion direction and anisotropy (Figure 2). Zheng and Pang [60] introduced a technique called 'HyperLIC' which allows all eigenvectors of D to influence the texture. Preusser et al. [53] have used an anisotropic reaction-diffusion equation to visualize vector fields. Laidlaw et al. [23] have proposed a hybrid approach: rendering textured glyphs. The resulting glyphs can portray all 6 unique diffusion tensor components. The glyphs can be overlaid on a scalar image (FA, for example) resulting in a layered image. Figures 1-2 show examples of tensor field visualization applied to a DT-MRI image of the human brain. Note that streamline visualization can give a misleading impression of certainty since only a single pathway be-

- *Tim McGraw is with West Virginia University, E-mail: tim.mcgraw@mail.wvu.edu.*
- *Mariappan Nadar is with Siemens Corporate Research, E-mail: mariappan.nadar@siemens.com.*

Manuscript received 31 March 2007; accepted 1 August 2007; posted online 27 October 2007.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

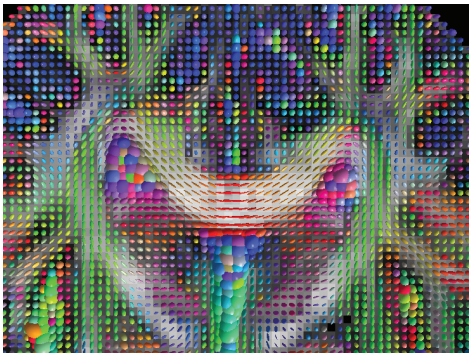

Fig. 1. Ellipsoid glyphs.

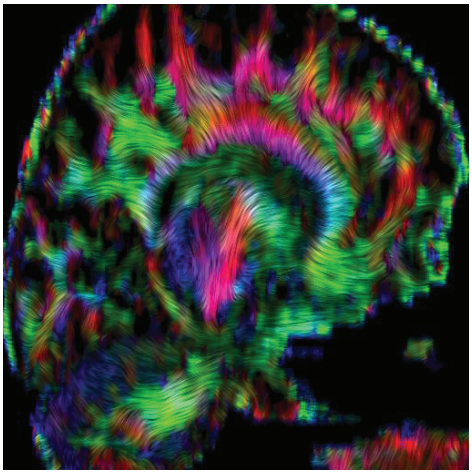

Fig. 2. Line integral convolution.

tween any two points is displayed. Although the PDD is usually a good *estimate* of the local fiber direction, many factors influence the quality of that estimate. In the following section we review some of the previously proposed methods for tracing fiber tracts from DT-MRI.

1.2 Tractography

Inferring the integrity and trajectory of white-matter pathways in the central nervous system has long been a goal of DT-MRI analysis. Detecting the presence of nerve fiber bundles has been solved using scalar measures of diffusion magnitude and anisotropy. In DT-MRI, indices of anisotropy include FA [2], relative anisotropy [2], volume ratio [39] and lattice anisotropy [39]. These local indices are computed from the diffusion tensor at each voxel. FA has found success in clinical applications, being used to detect tissue damage after stroke [48]. FA is attractive for its ease of computation and display, and robustness to noise in the data. Scalar indices make it possible to detect changes in neuronal fiber integrity, but do not indicate the degree to which connectivity has been compromised. They also do not describe the long-range implications of the change in anisotropy, such as which regions of the brain have been affected. Tractography is the process of estimating white-matter fiber pathways. Early tractography methods [34, 15] were similar to streamline computations in fluid mechanics. Fibers are traced by repeatedly stepping in the local direction of principal diffusion. Such tractography methods are confounded by the presence of fiber crossings or bifurcations, and the algorithms usually handle these cases by simply halting the tracking progress. Some algorithms [41, 42] use regularizing assumptions to make the tracking robust to the presence of noise and isotropy. Another approach [25] is to allow the tensor to merely deflect the current fiber direction.

Many recently proposed tractography algorithms are probabilistic in nature. While such techniques may generate families of possible fiber tracts, others eschew the fiber representation and instead produce probabilities of connectivity. Stochastic algorithms have an advantage over deterministic techniques in that it is more natural to take uncertainty into account. Uncertainty in diffusion tractography is due to a number of sources, including partial voluming and noise. By incorporating uncertainty or randomness into the tractography algorithm it is possible to compute and display distributions of fiber paths. Even though it may be possible within the stochastic framework to determine the most probable fiber path between two points, less probable paths may correspond to actual connectivity. The full distribution of possible fiber paths may have much clinical value. Bootstrap techniques have been used to generate distributions of diffusion weighted image intensities which lead to distributions of fiber tracts [24]. Bayesian models have enjoyed popularity in the context of fiber tracking due to the ability of these models to incorporate prior knowledge. Prior knowledge of fiber bending angle can be used to allow DT-MRI tractography to continue through voxels with low anisotropy. Monte Carlo techniques have been explored by [20, 37, 4, 38] to solve for connectivity probabilities which are formulated as a high-dimension integral. Importance sampling was used by [6] to generate large numbers of fiber tracts and visualize their dispersion. Mangin et al. [29] compute a sparse connectivity matrix in a stochastic framework.

The drawbacks of the current methods are that (a) they require time-consuming computation or stochastic simulation or (b) the resulting output only describes connectivity between a sparse set of image voxels. Algorithm run-time is a considerable hindrance to practical use. Sparse connectivity matrices may miss critical regions of impaired connectivity corresponding to minor fascicles. The ability to efficiently compute full, global connectivity will allow DT-MRI to be used in new applications, both clinically and in further research.

1.3 GPGPU for Medical Image Processing

Modern graphics hardware can be used to accelerate the display of medical image volume data [22]. This hardware is also capable of speeding-up the processing of the data prior to visualization. In the field of medical image processing, general purpose graphics processing unit (GPGPU) algorithms have been used for filtering [46], segmentation [45, 8, 26, 27], registration [49, 50] and image reconstruction [58, 47, 51]. In the context of DT-MRI, the GPU has been utilized for particle [21] and streamtube [43, 32] based visualization techniques, but to date, nobody has performed stochastic tractography or connectivity matrix computation on the GPU.

The fragment processors of the GPU excel at performing independent data-parallel operations. By implementing the fiber generation on the GPU we can generate many fibers in parallel. Memory limitations on the GPU are one constraint that limits the use of techniques for large datasets. Typically, the GPU may have between 256MB and 1GB of RAM. An overview of the capabilities of modern GPUs and common GPGPU programming techniques is given by Owens et al.[36].

2 Fiber Path Model

The fiber path is modeled as a sequence of displacement vectors $v_{1:n} = \{v_1, v_2, v_3 \ldots v_n\}$. A fiber path starting at a point x_0 consists of the points $\{x_0, x_1, x_2 \ldots x_n\}$ where $x_j = x_0 + \sum_{i=1}^{k} v_i$. We will assume that the path is arc-length parameterized so that all of the steps v_i are of equal length. Given the diffusion tensor field, $D(x)$, there is a probability associated with each step along the path. In order to impose a smoothness constraint on the path we will also condition the step probability on the previous step direction. So the path has a Markovian property which can also be seen as a mechanical constraint which prevents the fiber from bending too sharply. The probability of displacement v_i given the diffusion tensor, D, at the current location and the previous displacement, v_{i-1} can be denoted $p(v_i|D, v_{i-1})$. From this we may compute the probability associated with the entire path as the joint probability of the individual steps:

$$p(v_{1:n}|D) = p(v_1|D) \prod_{i=2}^{n} p(v_i|D, v_{i-1}). \quad (1)$$

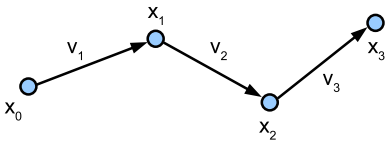

Fig. 3. Fiber path vertices, x, and displacements, v.

2.1 Bayesian Formulation

In order to generate random fibers we will start from a point within some voxel of interest, and draw random steps v_i from some distribution. This distribution will depend on the data, D, and the previous step direction, v_{i-1}. By Bayes' rule we have

$$p(v_i|D,v_{i-1}) = \frac{p(v_{i-1})p(v_i|v_{i-1})p(D|v_i,v_{i-1})}{p(D)p(v_{i-1}|D)}. \quad (2)$$

Assuming D and v_{i-1} are independent we have $p(D|v_i,v_{i-1}) = p(D|v_i)$ and $p(v_{i-1}|D) = p(v_{i-1})$. Then the posterior probability becomes

$$p(v_i|D,v_{i-1}) = \frac{p(v_i|v_{i-1})p(D|v_i)}{p(D)}. \quad (3)$$

Applying Bayes' rule again we can write

$$p(D|v_i) = \frac{p(D)p(v_i|D)}{p(v_i)}. \quad (4)$$

The posterior probability can now be rewritten as

$$p(v_i|D,v_{i-1}) = \frac{p(v_i|v_{i-1})p(v_i|D)}{p(v_i)}. \quad (5)$$

When we have no information about the data or previous direction we assume that all path directions are equally probable, so $p(v_i) =$ constant.

2.2 Parametric likelihood and prior

We will use a very simple model for the likelihood $p(v_i|D)$. We use the same model which governs the water diffusion itself. This model does not take into account noise in the diffusion weighted images or the associated uncertainty in the eigenvector computation. In this case, the posterior distribution represents smoothed water molecule paths.

The molecular diffusion displacement pdf is simply a zero-mean normal distribution, $p(v_i|D) = N(0,D(x))$ with covariance matrix equal to the diffusion tensor at that voxel. The tractography problem can be further simplified by assuming a constrained model of diffusion [13]. This model will result in reduced computational and memory requirements when implemented. The constrained diffusion model results in rotationally symmetric diffusion ellipsoids where the axis of symmetry is parallel to the dominant eigenvector, e_1. Given a tensor, D, with eigenvalues $\{\lambda_1,\lambda_2,\lambda_3\}$, it can be shown [12] that the nearest constrained tensor (in the sense of the Frobenius norm) has eigenvalues $\{\lambda_1, \frac{\lambda_2+\lambda_3}{2}, \frac{\lambda_2+\lambda_3}{2}\}$. The constrained tensor S can then be written in terms of eigenvalues and eigenvectors of D as

$$S = \lambda_1 e_1 e_1^T + \frac{\lambda_2+\lambda_3}{2}(e_2e_2^T + e_3e_3^T). \quad (6)$$

So we have

$$p(v_i|D) = c\exp(-\frac{1}{2}v_i^T S^{-1} v_i) \quad (7)$$

where S^{-1} is the inverse of the constrained tensor S(D), and c is a normalization constant. Note that S^{-1} and D have the same eigenvectors, and the eigenvalues of S^{-1} are $\hat{\lambda}_1 = \frac{1}{\lambda_1}$ and $\hat{\lambda}_2 = \hat{\lambda}_3 = \frac{2}{\lambda_2+\lambda_3}$. Using the fact that $I = e_1e_1^T + e_2e_2^T + e_3e_3^T$, we can write

$$S^{-1} = \hat{\lambda}_1 e_1 e_1^T + \hat{\lambda}_2(I - e_1e_1^T). \quad (8)$$

Since $v^T e_1 e_1^T v = (v\cdot e_1)^2$, the likelihood may be written as

$$p(v_i|D) = c\exp(-\frac{\hat{\lambda}_1-\hat{\lambda}_2}{2}(v_i\cdot e_1)^2 - \frac{\hat{\lambda}_2}{2}||v_i||^2). \quad (9)$$

For arc-length parameterized fibers $\frac{\hat{\lambda}_2}{2}||v_i||^2$ is constant with respect to v_i and can be absorbed into the normalization constant c. Equation (9) only requires one eigenvector and 2 eigenvalues to compute probabilities using the constrained diffusion model. Alternately, we could pass the full tensor and compute $e_1, \lambda_1, \lambda_2$ and λ_3 on the GPU, as done by Kondratieva et al. [21].

The constrained model is not an accurate representation of planar anisotropic diffusion ($\lambda_1 \approx \lambda_2 > \lambda_3$). In this case the oblate tensor can be more accurately modeled as $S = \frac{\lambda_1+\lambda_2}{2}(e_1e_1^T + e_2e_2^T) + \lambda_3 e_3 e_3^T$, where S^{-1} has eigenvectors $\tilde{\lambda}_1 = \tilde{\lambda}_2 = \frac{2}{\lambda_1+\lambda_2}$ and $\tilde{\lambda}_3 = \frac{1}{\lambda_3}$. Model selection can be governed by the linear and planar anisotropy measures (c_l and c_p) given by Westin [57]. An adaptive model for the likelihood is

$$p(v_i|D) = c\exp(-\frac{\delta}{2}(v_i\cdot\mu)^2), \quad (10)$$

where

$$\delta = \begin{cases} \hat{\lambda}_1 - \hat{\lambda}_2, & c_l >= c_p \\ \tilde{\lambda}_3 - \tilde{\lambda}_2, & c_l < c_p \end{cases}, \mu = \begin{cases} e_1, & c_l >= c_p \\ e_3, & c_l < c_p \end{cases}. \quad (11)$$

This is an antipodally symmetric distribution ($p(v_i|D) = p(-v_i|D)$) capable of representing both linear and planar anisotropic diffusion.

The prior distribution imposes a smoothness constraint on the randomly generated paths. A very weak prior assumption is that ($v_i\cdot v_{i-1} > 0$) so the fiber doesn't turn more than 90 degrees between successive steps. A stronger prior may be formulated as

$$p(v_i|v_{i-1}) = c\exp(\kappa(\frac{v_i}{||v_i||})^T(\frac{v_{i-1}}{||v_{i-1}||})). \quad (12)$$

This is a von-Mises Fisher distribution [30] with concentration parameter κ and mean direction $\frac{v_{i-1}}{||v_{i-1}||}$. The parameter κ controls how tightly the distribution is dispersed about the mean direction. The distribution is uniform over the sphere for $\kappa = 0$. Large values of κ give very concentrated distributions and result in smoother paths. For this reason, we have chosen to make κ a decreasing function of FA : $\kappa = k_1(1-FA)+k_2$. The prior has a regularizing effect on fiber paths, allowing them to continue tracking through areas of low FA.

Figure (4) shows probability profiles for the likelihood distribution (left column) and prior distribution (middle column) for different degrees of anisotropy and oblateness/prolateness. When anisotropy is high (top row) the likelihood is very tightly distributed about the mean direction. As FA increases (from top to bottom), the likelihood becomes more dispersed. The likelihood is nearly a uniform distribution in the case where $FA = 0.005$. The prior distributions were computed using a value of κ corresponding to $k_1 = 5.0$, $k_2 = 1.0$. As diffusion becomes more isotropic, κ increases, and the prior becomes more informative (less uniform).

2.3 Parametric posterior

Given the parametric forms for the likelihood (Equation 9) and prior (Equation 12)from the previous section we may write the posterior as

$$p(v_i|D,v_{i-1}) = c\exp[-\frac{\delta}{2}(v_i\cdot\mu)^2 + \kappa(\frac{v_i}{||v_i||})^T(\frac{v_{i-1}}{||v_{i-1}||})]. \quad (13)$$

The posterior distributions shown in the right column of Figure (4) reflect the varying influence of the prior distribution. When FA is high,

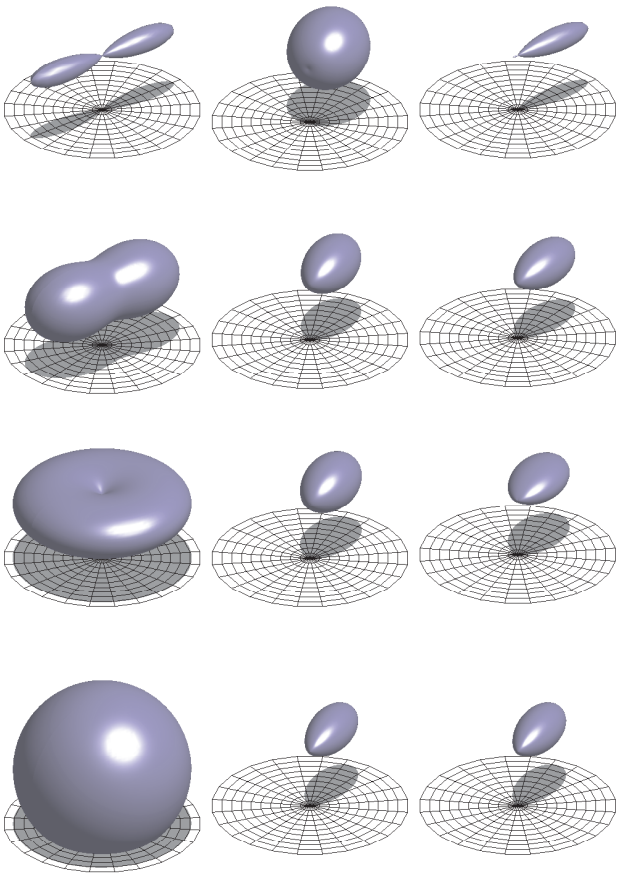

Fig. 4. Displacement probability surfaces of the likelihood (left), prior (middle), and posterior (right) distributions for four cases (top to bottom): FA = 0.9 (prolate) , FA = 0.4 (prolate), FA = 0.46 (oblate) and FA = 0.005.

the prior merely selects one of 2 antipodal directions, keeping the fiber from turning 180° in a single step. When FA is low, the mean direction of the posterior is greatly influenced by the prior. In section 3 we will describe how we draw samples from the posterior distribution to simulate fibers.

2.4 Connectivity Map

A simple connectivity map may be computed by seeding paths in a user-selected voxel and accumulating the number of randomly generated paths passing through each voxel in the dataset. As the algorithm progresses more random fibers will be generated. Let the number of fibers passing through voxel x be denoted as $n(x)$ and the total number of generated fibers be N. The probability that the seed voxel x_0 is connected to each voxel x can be approximated by

$$p_{\text{connect}}(x_0 \rightarrow x) \approx \frac{n(x)}{N}. \tag{14}$$

This is the model used by Friman et al. [12] under the assumption that all fiber lengths are equally probable. This technique is comparable to integrating over the high dimensional space of all possible fibers by rejection sampling. As N increases we sample more of the space of all possible fibers, and the approximation in (14) becomes more accurate. By implementing the fiber generation on the GPU we can generate many more fibers, and better estimate connection probabilities.

3 GPU Implementation

The connectivity model described above is implemented in two stages using the OpenGL shading language [44]. Recent GPU improvements have enabled wider general purpose applications. High precision computing is supported through 32 bit floating point render targets. The ability to render to a slice of a 3D texture simplifies processing of volumetric data. Support for multiple render targets allows fragment programs to write to multiple destination buffers. Such features are available on new hardware, but may be supported through vendor-specific extensions on older hardware. There are still several challenges associated with the task at hand:

- The GPU cannot generate pseudorandom numbers. Although there is a noise function exposed in the fragment programming languages it is not implemented on current hardware.
- There is no random-access read-write memory on the GPU. Fragment programs generally read from textures and write to predetermined locations in the framebuffer.
- Small memory size - typically from 256MB up to 1GB.

The first issue can be overcome by packing pregenerated random numbers into a texture. In this case we have chosen to place vectors with directions uniformly distributed over the unit sphere into a 3D texture. The second issue is commonly addressed in many GPGPU algorithms by ”ping-ponging”. This involves rendering textured fragments into a framebuffer object bound to another texture. After each iteration the input texture and framebuffer texture are swapped. This allows the results that were rendered to the framebuffer object to be read back into the fragment program on the next iteration. The swap is done by changing binding points, not by copying data. In fact, our framework can be implemented without any slow copying operations. The ”ping-ponging” operation does, however, double the required memory for variables, so for large datasets copying may be necessary. The final issue is addressed in part by the simplified model of diffusion which resulted in Equation (13). Fiber tracking with this model requires memory access to the vector μ and scalar parameters FA and δ at each voxel. For a $256 \times 256 \times 256$ dataset stored at 32 bit precision the memory requirements are 320MB. The full tensor model requires D and FA (as termination criterion) resulting in 448MB. Recent support for half precision floating point texture formats can reduce the memory requirements for both models. Evaluating (13) also requires fewer arithmetic operations than the corresponding unconstrained model.

The GPU tractography implementation works in three stages : fiber path simulation, connectivity computation, and display. The stages and the data flow are presented in Figure (5). Some initialization is required on the host CPU before the GPU-based algorithm can run. Eigenvalues and eigenvectors of D are computed on the CPU using Jacobi iteration. Then the required OpenGL textures and buffer objects are created and initialized. The dimensions of the textures x_{i-1}, x_i, v_{i-1} and v_i are all the same, and depend on how many fibers we wish to simulate. We use square ($m \times m$), power-of-2 textures since these are still generally more efficient on modern hardware. The eigenvector μ and the parameters κ, δ are stored in 3D textures having the same dimensions as the dataset. A frame buffer object (FBO) is created so that the rendering in stages 1 and 2 is output to the appropriate buffer, not the screen. The vertex buffer object (VBO) is created to hold m^2 vertices, and bound to `GL_PIXEL_PACK_BUFFER`. We will essentially be rendering to a vertex buffer object which we will later draw as point sprites. The fragment programs at each stage will be described in subsequent sections.

Stage 1 Update particle positions and displacements.

- Using stochastic interpolation, compute parameters of the posterior distribution $\mu, \delta, v_{i-1}, \kappa$.
- Draw a random displacement from posterior distribution.
- Compute displacements v_i, new particle positions x_i and update fiber length.
- Reinitialize particle position if x_i is outside of the white/gray matter domain or length threshold has been exceeded.

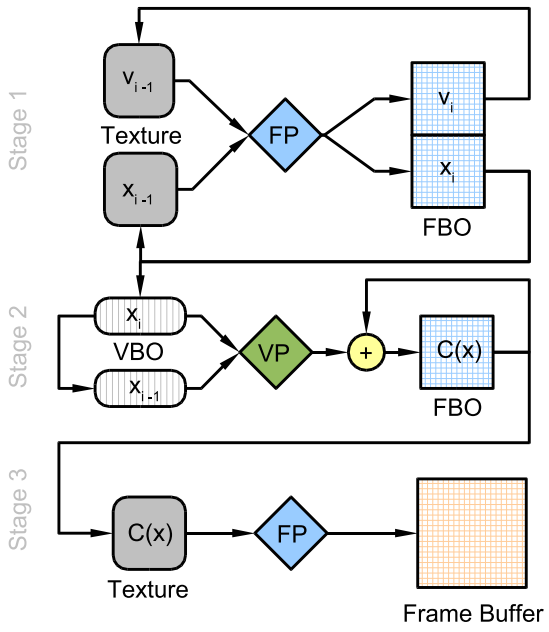

Fig. 5. Overview of GPU computations. Stages are executed in order from top to bottom. Diamonds labeled FP and VP represent fragment programs and vertex programs. Data stores are labeled inside with variable names and outside with the OpenGL buffer type. The variables x_{i-1} and x_i are particle positions, v_{i-1} and v_i are the fiber displacement vectors. Inputs to stages 1 and 3 are textures, the input to stage 2 is a vertex buffer object (VBO). Outputs from stages 1 and 2 are rendered into off-screen frame buffer objects (FBOs).

Stage 2 Accumulate connectivity using additive alpha blending.

- If particle has crossed a voxel boundary, then the point sprite color is white, alpha = α.
- Otherwise the point sprite color is black, alpha = 0.

Stage 3 Visualize connectivity maps.

- If C(x) is nonzero, then compute fragment intensity.
- Otherwise render voxel as black, alpha = 0;

3.1 Stage 1

The first two stages are common to many GPU-based particle advection schemes [19, 21]. The viewport size is set to $m \times m$, the size of texture x_{i-1}. A full-screen textured quadrilateral is drawn so that one fragment is generated per particle. Multiple render targets are attached to the FBO at this stage so that we can update position and displacement variables in a single fragment program. Buffers x_{i-1} and x_i are swapped in ping-pong fashion after each iteration. The displacements are stored in 4 channel RGBA textures and the 4th component is used to track the current fiber length. The novel aspect of our implementation of stage 1 is that the interpolation and displacements are stochastic, not deterministic. The fiber model is specified by the fragment program at this stage. The model - deterministic, constrained or Gaussian - may be changed by substituting the fragment program, and uploading the appropriate data.

Stochastic Interpolation. Since the tensor data only exist on lattice points of a discrete grid, some form of interpolation is necessary when fiber path vertices fall between lattice points. It is possible to interpolate between tensors and recompute the eigenvalues at each iteration, but this adds much computational burden. Stochastic interpolation, however, reflects the underlying uncertainty in the data. This scheme, suggested by Behrens et al. [5] randomly selects a tensor from the nearest neighbors. In 1-D stochastic interpolation is done by

$$D(x) = \begin{cases} D(\lfloor x \rfloor), & \text{if } U(0,1) > x - \lfloor x \rfloor; \\ D(\lceil x \rceil), & \text{otherwise.} \end{cases} \quad (15)$$

where $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ denote the floor and ceiling functions respectively, and $U(0,1)$ returns a random value uniformly distributed between 0 and 1. This means that two fiber paths which go through the same point may compute different interpolated values. This technique can be easily extended to higher dimensional interpolation. Stochastic interpolation in 3D is done using the following sequence of GLSL code:

```
vec3 u = texture3D(umap, pos.xyz + offset.xyz);
vec3 f = fract(pos.xyz);
vec3 c = ceil(pos.xyz);
vec3 ipos = c - step(f, u);
```

where `umap` is a texture containing uniform random values in each color component and `offset` is a random offset which changes each iteration.

Generating random fiber directions. The problem of drawing random samples from nonuniform distributions is discussed in detail by Devroye [9]. Given a univariate probability density function, $f(x)$, the inversion method entails inverting the cumulative distribution to obtain $F^{-1}(x)$. If U is uniform on the interval $[0,1]$, then $F^{-1}(U)$ has distribution function F and density f. We can apply this concept to multivariate distributions by discretizing the continuous density function into bins $f(r_i)$, integrating to obtain $F(r_i)$, drawing a random value u from $U(0,1)$, and finding the bin with associated direction r_i such that $F(r_{i-1}) < u \leq F(r_i)$.

Since it is not possible to generate pseudorandom numbers on the GPU, we generate them on the CPU and pack the values into a 4 channel floating point texture. At each texel the (r,g,b) components are unit vectors uniformly distributed over the unit sphere and the alpha channel contains values from U(0,1). Uniformly distributed unit vectors can be produced by drawing each component from a zero mean normal distribution and normalizing the vector [9].

In the fragment program we discretize the posterior into a fixed number of bins by drawing b randomly distributed unit vectors, $\{r_1, r_2, ...r_b\}$ from the texture map and setting $f(r_i) = p(r_i|D, v_{i-1})$. The value u is also read from the texture map, and the corresponding vector r_i is found by sequential search through the bins.

Drawing random values from the texture map involves computing a texture coordinate which is also randomized. The texture coordinate depends on particle position x_{i-1} and a random offset which is changed on the CPU and uploaded as a uniform variable. This way different particles will obtain a different sequence of sample directions, and the sample directions will change each iteration.

Fiber tracking will cease when the path exits the dataset, when the fractional anisotropy value falls below a certain threshold representing cerebro-spinal fluid (CSF) or when the length exceeds a threshold. When a fiber can no longer be tracked, it is randomly reinitialized within the initial voxel of interest. This way we track a constant number of fibers per iteration. The advecting particles (x_i) can be visualized directly after stage 1 as point sprites, but we instead compute a connectivity map.

3.2 Stage 2

The data is kept entirely on the GPU for rendering by binding the x_i texture to a VBO and calling `glDrawArrays`. Rendering at stage 2 is also off-screen (to a frame buffer object with attached 3D texture C(x)). The previous position (x_{i-1}) buffer is bound as a vertex attribute. We use the vertex normal to hold the previous position, although user-defined attributes may be used. The vertex buffer is rendered as points with additive alpha blending and depth testing disabled.

Since we may only render to a single slice of the attached 3D texture at a time, we accumulate connectivity values in a one voxel thick slab at a time. This is done by setting the the projection matrix to be

orthogonal with the near clipping plane at front face of the current slab and the far clipping plane at the back face of the slab. Stage 2 must be repeated for each slab we wish to compute connectivity information within.

The vertex program for this stage is given below:

```
gl_FrontColor = vec4(0.0, 0.0, 0.0, 0.0);
vec3 ipos = floor(gl_Vertex.xyz);
vec3 iprev = floor(gl_Normal.xyz); //previous position
if( any( notEqual( ipos, iprev) ) )
{
    gl_FrontColor = vec4(1.0, 1.0, 1.0, alpha);
}
```

The effect of this vertex program is that when a particle enters a new voxel, the particle is colored white with a small constant alpha value, α. Since additive alpha blending is used in stage 2, α, gets added to the previous connectivity value. The particles are rendered as 1 pixel points so only the voxel being entered has the connectivity value incremented.

3.3 Stage 3

Stage 3 is a color mapping stage where colors are assigned to connectivity values. Since the actual probability values are small, we use the negative logarithm of the probabilities as a base intensity value. The connectivity values $C(x)$ in the floating point texture output from stage 2 are not actual fiber counts $n(x)$, but scaled counts $\alpha n(x)$. The value of N increases whenever a fiber is reinitialized. We do not keep track of the total number of fibers generated during simulation since the reinitialization happens entirely on the GPU. If the initialized voxel was x_0, then $C(x_0) = \alpha N$, then the connectivity probabilities can be recovered as $p_{\text{connect}}(x_0 \rightarrow x) = \frac{C(x)}{C(x_0)}$ after the value of $C(x_0)$ is read back to the CPU and set as a uniform value of the fragment shader. Alternatively, to eliminate the need for a costly `glReadPixels` call, the value of N can be set by the user as part of the color mapping process.

4 Experimental Results

The connectivity algorithm described above was implemented two ways. First, a C++ implementation was written for execution on the host PC. Second, a GPU-based implementation was written using OpenGL 2.0 and GLSL. Experiments were run on two systems. System 1 was desktop PC featuring Intel Quad Core QX6700 2.66 GHz CPU and 4 GB RAM, and the GPU was a GeForce 8800 GTX with 768 MB VRAM. System 2 was notebook PC featuring Intel Core2 Duo T7500 2.2 GHz CPU and 2 GB RAM, and the GPU was a GeForce 8600M GT with 256 MB VRAM. The dataset studied was a human brain data set acquired from the Scientific Computing and Imaging (SCI) Institute at the University of Utah [52]. The matrix size is $148 \times 190 \times 160$, and the voxels measure $1.0mm \times 1.0mm \times 1.0mm$. We have used 16 bit half float textures to store the vector μ. The error associated with this representation is minimal. For this dataset the maximum error in any vector component was 0.0003, and the maximum angular error was 0.034° when compared with the 32-bit floating point representation.

We can simulate many paths, exploiting the parallelism of the fragment processors, but the size of the fragment programs is limited. To eliminate branching, loops are unrolled by the compiler, when possible, which leads to a longer fragment program. Using too many bins when discretizing the posterior distribution on the GPU leads to a dramatic drop in performance, suggesting that OpenGL has fallen back to a software path for rendering because the fragment program has exceeded the allowable length. In this case it may be necessary to split Stage 1 into multiple passes.

The connectivity images in Figures (6-9) were computed entirely on the GPU and volume rendered. Connectivity values are overlaid on FA images for reference. The connectivity values map to yellow for high probability and red for lower probability. Figure (6) was the result of seeding the fiber tracts near the pons within the brain stem. It is known that this area is connected to the internal capsule from where fibers fan out toward the cortical surface. The connectivity map reflects these connections.

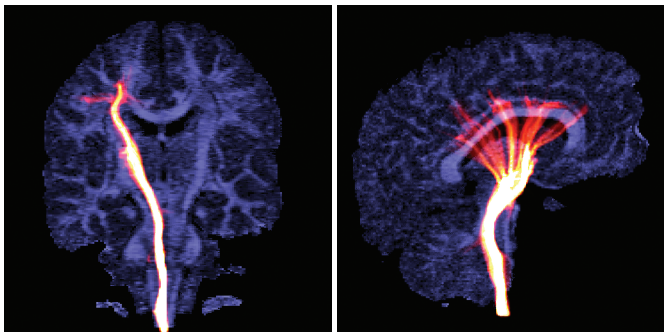

Fig. 6. Connectivity computed for seed point in brain stem after 850 iterations.

Figure (7) was generated by selecting the seed point to be within the cingulum bundle. These fibers arch over the corpus callosum in the anterior-posterior direction. Figure (8) shows the results of seeding in the occipitofrontal fasciculus. As expected, the connectivity map shows that this structure connects the frontal and occipital lobes of the brain. Figure (9) was the result of seeding fibers in the splenium and genu of the corpus callosum. This structure is known to connect the left and right hemispheres of the brain, a fact reflected in the connectivity map. The results shown in this section were computed by simulating over 250000 particles (a 512×512 buffer) advecting stochastically along fiber paths. The numerical results of the reference CPU algorithm agree with those of the GPU algorithm to within 3%. Note that these images are showing fundamentally different things from the visualizations in Figures (1) and (2). The glyph visualization techniques emphasize the nature of the tensor at each voxel

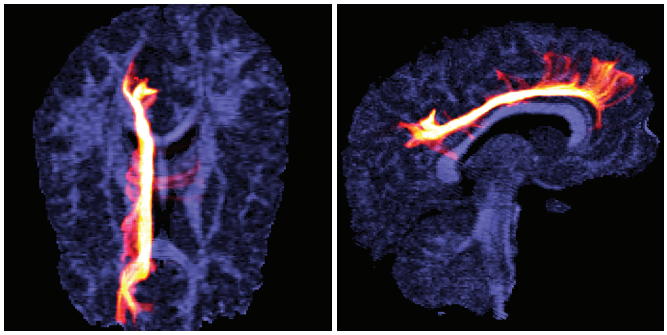

Fig. 7. Connectivity computed for seed point in cingulum bundle after 400 iterations.

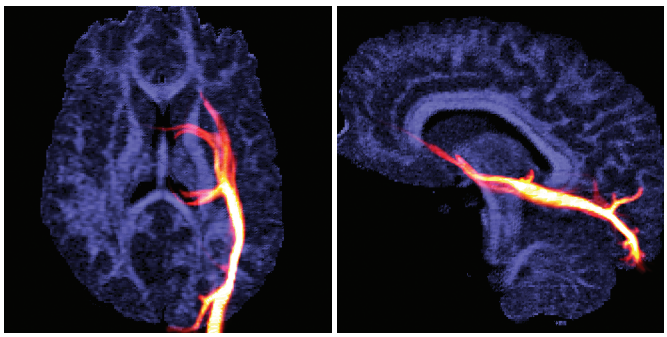

Fig. 8. Connectivity computed for seed point in occipitofrontal fasciculus after 700 iterations.

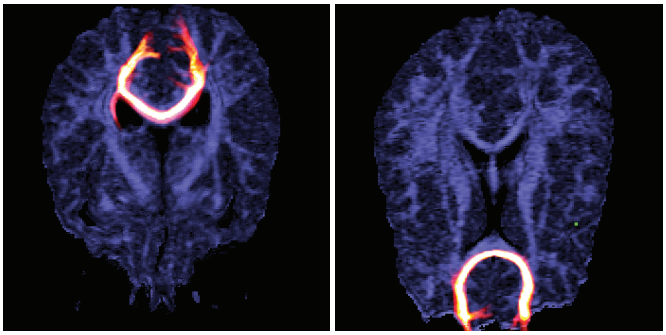

Fig. 9. Connectivity computed for seed points in the splenium (left) and genu of the corpus callosum (right) after 300 iterations .

m	data size	GPU Stage 1 (sec)	GPU Stage 2 (sec)	CPU
128	$128 \times 128 \times 128$	0.005	0.021	0.375
256	$128 \times 128 \times 128$	0.015	0.043	1.508
512	$128 \times 128 \times 128$	0.028	0.130	6.041
128	$256 \times 256 \times 256$	0.004	0.039	0.377
256	$256 \times 256 \times 256$	0.008	0.083	1.517
512	$256 \times 256 \times 256$	0.030	0.257	6.065

Table 1. Timing results for GPU vs. CPU implementation on system 1.

and local structure. The techniques based on deterministic streamlines can give a false impression of preciseness in tract location and direction. The high intensity regions in these images should not be interpreted as pathways. Adjacent voxels with high values do not reflect high connectivity probabilities between these voxels, but instead reflect high connectivity probability between each voxel and the initial seed voxel. Timing results for multiple data sizes and number of particles are shown in Tables (1 and 2). The different sized datasets were obtained by cropping or padding the dataset described above. For these experiments the posterior distribution was discretized into 100 bins. We have observed that the Stage 1 time is linear in the number of bins and that Stage 2 times are unaffected by this choice. Rendering time was constant in all cases (0.035 sec). As expected, Stage 1 times depend strongly on the number of particles being simulated. Since Stage 2 involves rendering the particles to a 3D texture the same size as the dataset, both data size and number of particles influence the timing. Note that for the largest dataset the available GPU memory for system 2 was exceeded. Omitting this case (*) we see a speedup of between $10\times$ and $27\times$ when using the GPU algorithm.

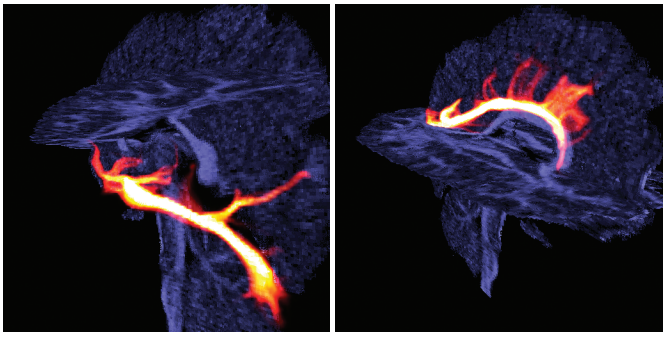

Fig. 10. Rotated views of occipitofrontal fibers (left) and cingulum fibers (right).

m	data size	GPU Stage 1 (sec)	GPU Stage 2 (sec)	CPU
128	$128 \times 128 \times 128$	0.010	0.021	0.459
256	$128 \times 128 \times 128$	0.033	0.049	1.836
512	$128 \times 128 \times 128$	0.119	0.158	7.398
128	$256 \times 256 \times 256$	0.012	0.036	0.462
256	$256 \times 256 \times 256$	0.033	0.093	1.848
512	$256 \times 256 \times 256$	0.121	0.745^*	7.413

Table 2. Timing results for GPU vs. CPU implementation for system 2.

5 Conclusion

We have presented a fast DT-MRI connectivity mapping algorithm and a framework for GPU implementation. This framework is general enough to accommodate fiber models other than the one presented here. It was demonstrated that this model can reconstruct plausible connectivity values, and that the GPU can compute these values much faster than the CPU. Future work will involve refining the fiber model and validating the connectivity results.

Acknowledgements

This work was supported in part by West Virginia University start-up funding and Siemens Corporate Research. Brain dataset courtesy of Gordon Kindlmann at the Scientific Computing and Imaging Institute, University of Utah, and Andrew Alexander, W. M. Keck Laboratory for Functional Brain Imaging and Behavior, University of Wisconsin-Madison.

References

[1] P. J. Basser. Inferring microstructural features and the physiological state of tissues from diffusion weighted images. *NMR in Biomed.*, 8:333–344, 1995.

[2] P. J. Basser. New histological and physiological stains derived from diffusion-tensor MR images. *Ann. N. Y. Acad. Sci.*, 820:123–138, 1997.

[3] P. J. Basser and C. Pierpaoli. Microstructural and physiological features of tissue elucidated by quantitative-diffusion-tensor MRI. *J. Magn. Reson. B*, 110:209–219, 1996.

[4] T. Behrens, H. Johansen-Berg, M. Woolrich, S. Smith, C. Wheeler-Kingshott, P. Boulby, G. Barker, E. Sillery, K. Sheehan, O. Ciccarelli, A. Thompson, J. Brady, and P. Matthews. Non-invasive mapping of connections between human thalamus and cortex using diffusion imaging. *Nature Neuroscience*, 6(7):750–757, July 2003.

[5] T. Behrens, M. Woolrich, M. Jenkinson, H. Johansen-Berg, R. Nunes, S. Clare, P. Matthews, J. Brady, and S. Smith. Characterization and propagation of uncertainty in diffusion-weighted MR imaging. *Magnetic Resonance in Medicine*, 50:1077–1088, 2003.

[6] M. Björnemo, A. Brun, R. Kikinis, and C. F. Westin. Regularized stochastic white matter tractography using diffusion tensor MRI. In *Fifth Intl. Conf. on MICCAI*, volume 1, pages 435–442, 2002.

[7] B. Cabral and L. Leedom. Imaging vector fields using line integral convolution. In *Proceedings of Siggraph*, pages 263–270, New York, 1993. ACM Press, New York.

[8] J. Cates, A. Lefohn, and R. Whitaker. GIST: an interactive, GPU-based level set segmentation tool for 3d medical images. *Medical Image Analysis*, 8(3):217–231, 2004.

[9] L. Devroye. *Non-uniform random variate generation*. Springer-Verlag New York, 1986.

[10] R. A. Drebin, L. Carpenter, and P. Hanrahan. Volume Rendering. In *Computer Graphics, Proceedings of SIGGRAPH 88*, volume 22, pages 65–74, 1988.

[11] F. Enders, N. Sauber, D. Merhof, P. Hastreiter, C. Nimsky, and M. Stamminger. Visualization of White Matter Tracts with Wrapped Streamlines. *IEEE Visualization*, pages 51–58, 2005.

[12] O. Friman, G. Farneback, and C.-F. Westin. A Bayesian approach for stochastic white matter tractography. *Transactions on Medical Imaging*, 25(8):965–978, 2006.

[13] O. Friman and C.-F. Westin. Uncertainty in fiber tractography. In *Eighth International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI'05)*, Lecture Notes in Computer Science 3749, pages 107–114, Palm Springs, CA, USA, October 2005.

[14] E. Hsu. Generalized Line Integral Convolution Rendering of Diffusion Tensor Fields. *Proceedings of the International Society for Magnetic Resonance in Medicine (ISMRM)*, page 790, 2001.

[15] D. K. Jones, A. Simmons, S. C. R. Williams, and M. A. Horsfield. Non-invasive assessment of axonal fiber connectivity in the human brain via diffusion tensor MRI. *Magn. Reson. Med.*, 42:37–41, 1999.

[16] G. Kindlmann. Superquadric tensor glyphs. In *Proceedings of IEEE TVCG/EG Symposium on Visualization 2004*, pages 147–154, May 2004.

[17] G. Kindlmann and C. Westin. Diffusion tensor visualization with glyph packing. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1329–1335, 2005.

[18] G. L. Kindlmann and D. M. Weinstein. Hue-balls and lit-tensors for direct volume rendering of diffusion tensor fields. In *IEEE Visualization '99*, pages 183–190, 1999.

[19] P. Kipfer, M. Segal, and R. Westermann. UberFlow: a GPU-based particle engine. *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 115–122, 2004.

[20] M. Koch, D. Norris, and M. Hund-Georgiadis. An investigation of functional and anatomical connectivity using magnetic resonance imaging. *Neuroimage*, 16(1):241–250, 2002.

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

[22] J. Kruger and R. Westermann. Acceleration Techniques for GPU-based Volume Rendering. *Proceedings of the 14th IEEE Visualization 2003*, pages 287–292, 2003.

[23] D. H. Laidlaw, E. T. Ahrens, D. Kremers, M. J. Avalos, C. Readhead, and R. E. Jacobs. Visualizing diffusion tensor images of the mouse spinal cord. In *Proceedings of IEEE Visualization 1998*, pages 127–134. IEEE Computer Society Press, New York, October 1998.

[24] M. Lazar and A. Alexander. Bootstrap white matter tractography (BOOT-TRAC). *Neuroimage*, 24(2):524–532, 2005.

[25] M. Lazar, D. M.Weinstein, J. S.Tsuruda, K. M.Hasan, K. Arfanakis, M. E. Meyerand, B. Badie, H. A.Rowley, V. Haughton, A. Field, and A. L.Alexander. White matter tractography using diffusion tensor deflection. *Human Brain Mapping*, 18:306–321, 2003.

[26] A. Lefohn, J. Cates, and R. Whitaker. Interactive, GPU-Based Level Sets for 3D Segmentation. *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, pages 564–572, 2003.

[27] A. Lefohn, J. Kniss, C. Hansen, and R. Whitaker. A streaming narrow-band algorithm: interactive computation and visualization of level sets. *Visualization and Computer Graphics, IEEE Transactions on*, 10(4):422–433, 2004.

[28] M. Levoy. Display of surfaces from volume data. *IEEE Comput. Graph. Appl.*, 8(3):29–37, 1988.

[29] J.-F. Mangin, C. Poupon, Y. Cointepas, D. Rivière, D. Papadopoulos-Orfanos, C. A. Clark, J. Régis, and D. L. Bihan. A framework based on spin glass models for the inference of anatomical connectivity from diffusion-weighted MR data. *NMR in Biomedicine*, 15:481–492, 2002.

[30] K. V. Mardia and P. Jupp. *Directional Statistics*. John Wiley and Sons Ltd., New York, 2nd edition, 2000.

[31] T. McGraw, B. Vemuri, Z. Wang, Y. Chen, M. Rao, and T. Mareci. Line integral convolution for visualization of fiber tract maps from DTI. In *Fifth Intl. Conf. on MICCAI*, pages 615–622, Tokyo, Japan, 2002.

[32] D. Merhof, M. Sonntag, F. Enders, C. Nimsky, P. Hastreiter, and G. Greiner. Hybrid visualization for white matter tracts using triangle strips and point sprites. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1181–1188, 2006.

[33] B. Moberts, A. Vilanova, and J. van Wijk. Evaluation of Fiber Clustering Methods for Diffusion Tensor Imaging. *IEEE Visualization*, pages 65–72, 2005.

[34] S. Mori, B. J. Crain, V. P. Chacko, and P. C. M. van Zijl. Three-dimensional tracking of axonal projections in the brain by magnetic resonance imaging. *Ann. Neurol.*, 45:265–269, 1999.

[35] S. Mori and P. C. van Zijl. Fiber tracking: principles and strategies - a technical review. *NMR Biomed*, 15(7-8):468–480, 2002.

[36] J. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. Lefohn, and T. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1 or 2):to appear, 2007.

[37] G. Parker and D. Alexander. Probabilistic monte carlo based mapping of cerebral connections utilising whole-brain crossing fibre information. In *Proc. of Information Processing in Medical Imaging*, pages 684–695, 2003.

[38] G. Parker and D. Alexander. Probabilistic anatomical connectivity derived from the microscopic persistent angular structure of cerebral tissue. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 360(1457):893–902, 2005.

[39] C. Pierpaoli and P. J. Basser. Toward a quantitative assessment of diffusion anisotropy. *Magn. Reson. Med.*, 36:893–906, 1996.

[40] C. Pierpaoli, P. Jezzard, P. J. Basser, A. Barnett, and G. D. Chiro. Diffusion tensor mr imaging of the human brain. *Radiology*, 201:637–648, 1996.

[41] C. Poupon, C. A. Clark, V. Frouin, J. Régis, I. Bloch, D. L. Bihan, and J.-F. Mangin. Regularization of diffusion-based direction maps for the tracking of brain white matter fascicles. *Neuroimage*, 12(2):184–195, 2000.

[42] C. Poupon, J. Mangin, C. Clark, V. Frouin, J. Regis, D. L. Bihan, and I. Bloch. Towards inference of human brain connectivity from MR diffusion tensor data. *Med. Image Anal.*, 5(1):1–15, 2001.

[43] G. Reina, K. Bidmon, F. Enders, P. Hastreiter, and T. Ertl. GPU-Based Hyperstreamlines for Diffusion Tensor Imaging. In *Proceedings of EUROGRAPHICS - IEEE VGTC Symposium on Visualization 2006*, pages 35–42, 2006.

[44] R. Rost. *OpenGL Shading Language*. Addison-Wesley, 2006.

[45] M. Rumpf and R. Strzodka. Level set segmentation in graphics hardware. *Image Processing, 2001. Proceedings. 2001 International Conference on*, 3, 2001.

[46] M. Rumpf and R. Strzodka. Nonlinear diffusion in graphics hardware. *Proceedings of Eurographics/IEEE TCVG Symposium on Visualization*, pages 75–84, 2001.

[47] T. Schiwietz, T. Chang, P. Speier, and R. Westermann. MR image reconstruction using the GPU. *Proceedings of the SPIE, Advanced Optical and Quantum Memories and Computing III.*, 6142:1279–1290, 2006.

[48] A. Sorensen, O. Wu, W. Copen, T. Davis, R. Gonzalez, W. Koroshetz, T. Reese, B. Rosen, V. Wedeen, and R. Weisskoff. Human acute cerebral ischemia: Detection of changes in water diffusion anisotropy by using MR imaging. *Radiology*, 212:785–792, 1999.

[49] R. Strzodka, M. Droske, and M. Rumpf. Fast image registration in DX9 graphics hardware. *Journal of Medical Informatics and Technologies*, 6(43-49):143, 2003.

[50] R. Strzodka, M. Droske, and M. Rumpf. Image Registration by a Regularized Gradient Flow. A Streaming Implementation in DX9 Graphics Hardware. *Computing*, 73(4):373–389, 2004.

[51] T. Sumanaweera and D. Liu. Medical image reconstruction with the FFT. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, 2005.

[52] The Scientific Computing and Imaging (SCI) Institute, University of Utah. Diffusion tensor MRI datasets (http://www.sci.utah.edu/∼gk/DTI-data/). 2000.

[53] T.Preußer and M. Rumpf. Anisotropic nonlinear diffusion in flow visualization. In *IEEE Visualization*, pages 325–332, 1999.

[54] D. Werring, C. Clark, G. Barker, D. Miller, G. Parker, M. Brammer, E. Bullmore, V. Giampietro, and A. Thompson. The structural and functional mechanisms of motor recovery: complementary use of diffusion tensor and functional magnetic resonance imaging in a traumatic injury of the internal capsule. *Journal of Neurology, Neurosurgery & Psychiatry*, 65(6):863–869, 1998.

[55] D. Werring, C. Clark, G. Barker, A. Thompson, and D. Miller. Diffusion tensor imaging of lesions and normal-appearing white matter in multiple sclerosis. *Neurology*, 52:1626, 1999.

[56] D. Werring, A. Toosy, C. Clark, G. Parker, G. Barker, D. Miller, and A. Thompson. Diffusion tensor imaging can detect and quantify corticospinal tract degeneration after stroke. *J Neurol Neurosurg Psychiatry*, 69:269–272, 2000.

[57] C.-F. Westin, S. E. Maier, H. Mamata, A. Nabavi, F. A. Jolesz, and R. Kikinis. Processing and visualization of diffusion tensor MRI. *Medical Image Analysis*, 6(2):93–108, 2002.

[58] F. Xu and K. Mueller. Accelerating popular tomographic reconstruction algorithms on commodity pc graphics hardware. *IEEE Transactions on Nuclear Science*, 52(3):654–663, 2005.

[59] S. Zhang, Ç. Demiralp, and D. H. Laidlaw. Visualizing diffusion tensor MR images using streamtubes and streamsurfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(4):454–462, 2003.

[60] X. Zheng and A. Pang. Hyperlic. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 33, Washington, DC, USA, 2003. IEEE Computer Society.