Interactive Visualization of Volumetric White Matter Connectivity in DT-MRI Using a Parallel-Hardware Hamilton-Jacobi Solver

Won-Ki Jeong, *Student Member, IEEE*, P. Thomas Fletcher, Ran Tao, and Ross T. Whitaker *Member, IEEE*

Abstract—In this paper we present a method to compute and visualize volumetric white matter connectivity in diffusion tensor magnetic resonance imaging (DT-MRI) using a Hamilton-Jacobi (H-J) solver on the GPU (Graphics Processing Unit). Paths through the volume are assigned costs that are lower if they are consistent with the preferred diffusion directions. The proposed method finds a set of voxels in the DTI volume that contain paths between two regions whose costs are within a threshold of the optimal path. The result is a volumetric optimal path analysis, which is driven by clinical and scientific questions relating to the connectivity between various known anatomical regions of the brain. To solve the minimal path problem quickly, we introduce a novel numerical algorithm for solving H-J equations, which we call the Fast Iterative Method (FIM). This algorithm is well-adapted to parallel architectures, and we present a GPU-based implementation, which runs roughly 50-100 times faster than traditional CPU-based solvers for anisotropic H-J equations. The proposed system allows users to freely change the endpoints of interesting pathways and to visualize the optimal volumetric path between them at an interactive rate. We demonstrate the proposed method on some synthetic and real DT-MRI datasets and compare the performance with existing methods.

Index Terms—Diffusion tensor visualization, graphics hardware, interactivity.

1 Introduction

Diffusion tensor magnetic resonance imaging (DT-MRI) is a powerful technique for imaging *in vivo* properties of white matter tissue in the human brain. Visualizations of the white matter pathways and of possible abnormalities along these pathways are an important tool for clinicians studying neuropsychiatric disorders. The physical principle behind diffusion imaging is that the motion of water is impeded in directions that are not parallel to the axons. In DT-MRI a diffusion tensor at each voxel gives an estimated model of the pattern of water diffusion aggregated over a point-spread function of the measurements. The neural fiber orientation is typically inferred from the principal eigenvector of the diffusion tensor, which is the direction of highest probability of water motion.

Several methods based on the Hamilton-Jacobi (H-J) equation have recently been introduced as a means for describing connectivity in white matter. A considerable amount of research has focused on developing numerically efficient algorithms to solve the H-J equation. Most solvers are based on either iterative schemes using pre-defined update orders or one-pass schemes using sorted data structures. However, solving the H-J equation on highly anisotropic speed volumes, such as those found in DT-MRI, usually requires at least a few minutes even on the fastest processor, making the application less interactive. Therefore, there remains a need for fast solutions to the H-J equation in a variety of visualization-related applications..

GPUs have evolved into a massively parallelized multiprocessor machine over the last few years. The current GPUs have up to 128 microprocessors running in a SIMD fashion, producing throughput of several hundreds GFLOPS (Giga Floating Point Operations Per Second). The GPU supports high precision computation up to 32bit floating point (64bit is on its way), has extremely wide memory bandwidth, and is fully programmable. In addition, the new generation GPUs based on DirectX 10 support much more flexible flow controls and memory management, and provide rich instruction sets and high level languages for GPGPU (General Purpose computation on the GPU) that help to reduce the overhead of executing graphics APIs and provide more flexible programming models. Due to their increasing computational power and programming flexibility, current GPUs are becoming a very powerful parallel computing platform for GPGPU problems [17].

In this paper we introduce an interactive system to compute and visualize volumetric white matter connectivity in DT-MRI volumes. Our method is based on the framework for volumetric quantification of DTI connectivity previously presented in [6]. While that work focused on quantification of connectivity and the computations were done offline on the CPU, this paper extends the visualization aspect of the method and focuses on an efficient, interactive computational algorithm on the GPU. Existing CPU-based methods to extract DT-MRI white matter connectivity rely on the pre-computed distance volume because distance computation on the anisotropic speed volume is a highly time-consuming process, and therefore, up to our knowledge, there are no existing systems that can extract the pathways in the DT-MRI volumes at an interactive rate. The proposed system uses the graphics processors to solve the H-J equation quickly, roughly 50-100 times faster than traditional CPU-based solvers, allowing users to freely change the endpoints of interesting pathways and to visualize the volumetric connectivity between them at an interactive rate. The main contribution of this paper is introducing a novel numerical algorithm to solve the H-J equation that can be well-adapted to various parallel architectures, an improved Godunov Hamiltonian computation, and a GPU implementation of the proposed H-J solver.

2 Related Work

2.1 DT-MRI connectivity

Much of the work in DT-MRI connectivity focuses on fiber tractography [3], in which streamlines are computed, by forward integration from a seed point, of the field of vectors defined by the principal eigenvector of the tensor at each point (interpolated between voxels), and where the twofold ambiguity of eigenvector directions is resolved by the continuity of paths. As a clinical tool for analyzing white matter pathways, tractography suffers from several drawbacks. First, imaging noise can cause fiber tracts to stray due to accumulating errors in the integration. The second issue is partial voluming. The finite size of a voxel measurement at fiber crossings (combined with sensor noise) can cause the direction of the major eigenvector to be ambiguous, further misleading the streamlines. This problem is aggravated by the fact that streamlines are often computed, displayed, and analyzed at sub-voxel resolution—suggesting a level of precision that is not warranted

• *Won-Ki Jeong, P. Thomas Fletcher, Ran Tao, and Ross T. Whitaker are with the Scientific Computing and Imaging Institute, School of Computing, University of Utah, 72 S. Central Camput Dr., Salt Lake City, UT 84112. E-mail: {wkjeong, rantao, whitaker}@cs.utah.edu, fletcher@sci.utah.edu.*

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.

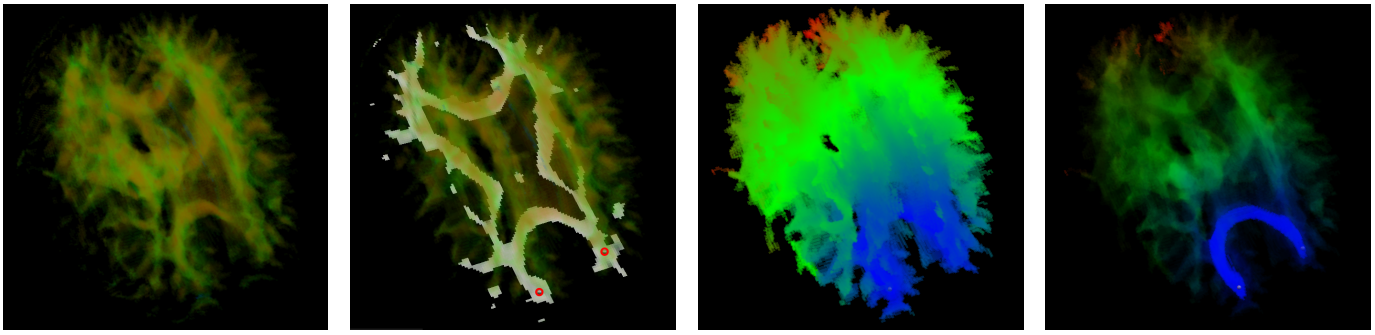


Fig. 1. System Overview. From left to right: a) Input DT-MRI volume; b) Seed points (marked as red circles); c) Cost volume (blue to red : low to high); d) Volumetric path along genu (blue).

by the data. Finally, region-to-region analysis with conventional tractography is challenging, because there is no way to steer tracts from a seed point toward a particular target region. To address these problems, several researchers propose tractography algorithms that rely on a stochastic integration, in which flow vector are chosen from a distribution around the principal eigenvector. These stochastic techniques can be combined with Monte-Carlo simulations, which may include tens of thousands of paths from a single seed, of which only a small fraction will typically reach the target [13, 4, 19, 14].

Several H-J methods for white matter connectivity have been proposed to overcome some of the difficulties arising in tractography. These methods compute the cost of the shortest path from a seed region to every pixel in the volume (usually a white matter mask). This cost function consists of an integral that depends on path position and orientation, and typically penalizes paths that do not agree with the tensors. These H-J formulations result in first-order partial differential equations (PDEs) which model evolving fronts whose speeds are determined by information from the diffusion tensor. These methods are inherently more robust to noise in the diffusion weighted measurements than standard tractography. Parker et al. [18] evolve a front with speed related to the inner product of the front normal with the principal eigenvector of the tensor. O'Donnell et al. [16] propose using the diffusion tensor as a Riemannian metric in the image domain and compute a front representing arrival time of geodesics beginning at a single seed point. Connectivity to that point is defined as a ratio of Euclidean path length to Riemannian distance. Jackowski et al. [8] use a speed derived as a function of the diffusivity magnitude in the front normal direction. They solve this Hamiltonian equation using a Lax-Friedrichs scheme, also beginning with an initial seed point. Pichon et al. [20] define a directionally dependent local cost function that extends the H-J framework to high-angular diffusion data. In all of these works, the end result is either a dense field of connectivities to regions or a set of optimal paths emanating from a seed region, which are determined by integrating the characteristics of the PDEs.

2.2 Hamilton-Jacobi equation solver

A number of different numerical strategies have been proposed to efficiently solve the H-J equation. These methods can be classified into two groups. One is a class of iterative methods based on a fixed-point update using Jacobi or Gauss-Seidel schemes. An early work by Rouy et al. [23] solves the Eikonal equation, a special case of H-J equation, by updating the solutions of the grid using a pre-defined updating order and Godunov upwind Hamiltonian until they converge. Zhao [30] proposed the Fast Sweeping method, which uses a Gauss-Seidel updating order for fast convergence. Tsai et al. [29] employed the Fast Sweeping method and a Godunov upwind discretization of the class of convex Hamiltonians to solve anisotropic H-J equations. The proposed Godunov Hamiltonian uses only 1-neighborhood pixels, so it maps well on iterative schemes. However, there are many cases to check for the correct solution of the Hamiltonian. Kao et al. [11] introduced a new interpretation of Hamiltonians based on the Legendre transformation, and in a subsequent paper [10] they employed the Lax-Friedrichs Hamiltonian for arbitrary static H-J equations. The proposed method is simple to implement and can be used widely on both convex and non-convex H-J equations, but it requires many more iterations than the Godunov Hamiltonian and the solution shows excessive diffusion due to the nature of the scheme.

Another class of H-J solvers is based on adaptive updating schemes and sorting data structures. An earlier work by Qin et al. [22] and later Sethian et al. [24, 25, 26] used a Dijkstra-type shortest path algorithm to solve H-J equations, which is generally referred to as the Fast Marching method. The main idea behind this method is that solutions for a convex Hamiltonian depend only on the upwind neighbors along the characteristics, so the causality relationship can be determined uniquely and the correct solutions can be computed by only a single pass update. The complexity of the Fast Marching method is $O(N\log N)$, which is worst-case optimal, and the running time is not much affected by the complexity of the speed. However, for a class of general H-J equations [26], tracing the characteristics can cause expensive searching among a wider range of neighborhoods than solving equations using an iterative numerical method. In addition, the method uses a global sorting data structure, e.g., a heap, and therefore the parallelization is not straightforward.

Even though there has been much research effort on general purpose computing using the GPU, so far no one has developed a parallel algorithm for solving general H-J equations on the GPU. The closest work has focused on the distance or Voronoi diagrams computations on the GPU, which is equivalent to solving the Eikonal equation with a constant speed, a special case of the H-J equation. Hoff et al. [7] first used OpenGL API and hardware rasterization in the fixed graphics pipeline to compute approximated Voronoi diagrams. Sigg et al. [27] employed a scan conversion algorithm and used fragment programs to compute the distance to triangular meshes within narrow bands around the mesh. Sud et al. [28] proposed a method to reduce unnecessary distance computations by using culling and clamping algorithms. The distance transform approaches shown above are all based on the assumption of isotropic constant speed over the domain, so they are not applicable to solving the H-J equation based on anisotropic tensor speed functions.

3 Region-to-Region Connectivity

Our formulation of region-to-region connectivity is based on the principle of minimal cost paths. Using information from the entire diffusion tensor, we construct a local cost function based on the current position and directionality of a path. This leads to a first-order nonlinear PDE that computes the minimal cost from a starting region to each point in the image. Unlike previous front-propagation methods for DT-MRI, we then solve for minimal cost from a second target region. The two solutions are then combined, giving the minimal cost through each voxel of paths restricted to travel between the two target regions. The minimal cost function can then be used to threshold the data for visualization of only the pertinent white matter pathways. Also, the cost function serves as a measure of the strength, or integrity, of the connection. Visualizations of this cost function can be used to

locate abnormalities, or weaknesses, in the white matter pathways in individuals with neuropsychiatric disorders.

An overview of our interactive region-to-region connectivity visualization system is shown in Figure 1. First, a volume rendering of the fractional anisotropy (FA), which is a measure of the anisotropy or elongation of the tensors, is displayed. Next, two starting seed points are chosen on 2D orthogonal slices of the FA map as the endpoints of the desired white matter pathway. Next, the cost function is computed using the GPU H-J solver as described in this paper. The cost function is volume rendered, displaying the areas of high connectivity between the two regions. Finally, the current pathway can be rendered by thresholding the cost function.

Given a path $c : [a,b] \to \Omega$, where Ω is a compact image domain, we define the total cost of c as

$$E(c) = \int_a^b \psi(c(t), T(t))dt, \tag{1}$$

where $T(t) = c'(t)/\|c'(t)\|$ is the unit tangent vector of c. The total cost is defined as the integral of a *local cost function*, $\psi : \Omega \times S^1 \to \mathbb{R}$, where $\psi(x,v)$ gives the cost of moving in the unit direction $v \in S^1$ from the point $x \in \Omega$. We require that the local cost be symmetric, $\psi(x,v) = \psi(x,-v)$, which is generally consistent with the model of diffusion through passive media.

This metric in (1) allows for a wide range of cost functions ψ that incorporate tangents. Pichon et al. [20] describe the properties of this metric, the choices of ψ for high-angular diffusion data, and the relationship between this cost function and the corresponding *speed* that controls the motion of the wavefront in the H-J formulation. In this work we use a quadratic (bilinear) local cost function, with the understanding that all of the results in this paper generalize to high-angular data using the methods described in [20]. Thus we have

$$\psi(x,v) = v^T M(x) v, \tag{2}$$

where $M(x)$ is a symmetric, positive-definite matrix defined at each point $x \in \Omega$.

The relationship between the measured diffusion tensor field, D, and the metric for the path cost, M, must be considered carefully. We find that using the inverse of the original tensor field, $M = D^{-1}$, as proposed in most H-J DTI approaches [18, 8, 20], does not sufficiently penalize paths in directions perpendicular to the major eigenvector. In fact, shortest paths using this method typically look almost identical to the paths defined by isotropic metrics constrained the white matter mask. Thus, we use a *sharpened* tensor field, which is the original tensor field raised to a power α. This must be combined with a normalization, and for this work we normalize by the tensor volume. If we consider the sharpened tensor to be speed (in the H-J formulation), which gives low cost along the principal eigen directions, the cost is the inverse, and we have

$$M(x) = |D(x)|^{-\frac{1}{3}} \left(\frac{D(x)}{|D(x)|^{\frac{1}{3}}} \right)^{-\alpha}, \tag{3}$$

where $\alpha > 1$ is a constant and $|D(x)|$ denotes the determinant of $D(x)$. We used $\alpha = 3$ for all of the experiments in this paper.

We can consider all paths emanating from a region $R_1 \subset \Omega$. Let $u_1(x)$ denote the minimal cost as defined by (1) over all paths beginning in the region R_1 and terminating at the point x. Then u_1 satisfies the convex H-J equation given as follows:

$$H(\nabla u_1, \mathbf{x}) = \sqrt{(\nabla u_1)^T M (\nabla u_1)} = 1, \quad \forall \mathbf{x} \in \Omega, \tag{4}$$

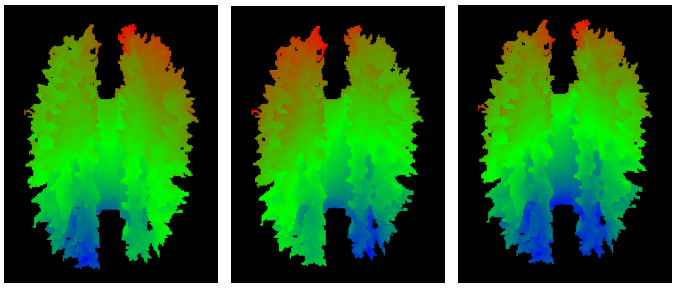
where Ω is a domain in R^n, $u_1(\mathbf{x})$ is the travel time or distance from the source, R_1, and M is the speed matrix defined on Ω by Equation 3. We use the Hamiltonian defined below for our 3D DT-MRI connectivity problem:

$$H(p,q,r) = \sqrt{ap^2 + dq^2 + fr^2 + 2(bpq + cpr + eqr)} \tag{5}$$

$$M = \begin{bmatrix} a & b & c \\ b & d & e \\ c & e & f \end{bmatrix}, \; p = \frac{\partial u_i}{\partial x}, q = \frac{\partial u_i}{\partial y}, r = \frac{\partial u_i}{\partial z}$$

where p, q, and r are partial derivatives of u_i at $\mathbf{x}$ along x, y, and z axis, and a, b, c, d, e, and f are triangular elements of the matrix M. Equation 4 becomes the Eikonal equation when M is an identity matrix.

While u_1 gives us a measure of the connectivity from the region R_1 to any point in the image, we would like to assess the specific connectivity to a second target region. To do this, we define a second region $R_2 \subset \Omega$ and corresponding minimal cost function u_2 also satisfying (4). Consider all paths beginning in the region R_1 and terminating in R_2. Now we define the *total cost function* for regions R_1 and R_2 to be $u(x) = u_1(x) + u_2(x)$. The value of $u(x)$ is the minimal cost of all paths between R_1 and R_2 *that are constrained to pass through* x. Figure 2 shows the cost functions for paths along genu ((a) u_1, (b) u_2, and (c) the total cost $u = u_1 + u_2$).



(a) Cost u_1 (b) Cost u_2 (c) Total cost u

Fig. 2. The cost functions for genu. Blue to red: low cost to high cost.

We use the total cost function u to define which voxels in the image are contained in the pathway of interest. Let γ be the minimal total cost path, and fix a threshold $\varepsilon \geq 0$, which is the tolerance of paths relative to the optimum. We define an *ε-point* as a point whose constrained minimum cost is less than $(1+\varepsilon)E(\gamma)$. The set of all such ε-points defines an *volumetric pathway* between R_1 and R_2. This region is the set of voxels that belong to the fiber connection between R_1 and R_2. By definition, a volumetric pathway must contain γ for any value of $\varepsilon \geq 0$.

4 Parallel Hamilton-Jacobi Equation Solver

4.1 Fast Iterative Method

FIM is a numerical algorithm to solve PDEs, such as Equation 4, on parallel architectures. The main idea of FIM is to solve the H-J equation selectively on the grid nodes without maintaining expensive data structures. FIM maintains a narrow band, called the *active list*, for storing the index of grid nodes to be updated. Instead of using a special data structure to keep track of exact causal relationships, it maintains a looser relationship and update all nodes in the active list simultaneously (i.e., Jacobi update). During each iteration, we expand the list of active nodes, and the band thickens or expands to include all nodes that could be influenced by the current updates. A node can be removed from the active list when the solution is converged, and re-inserted when any changes of its adjacent neighbors affect the solution of the current node. Note that newly inserted nodes must be updated in the following update iteration to ensure a correct Jacobi update. To compute the solutions of the nodes in the active list, we use the Godunov upwind discretization of the Hamiltonian (section 4.2). The key ideas of the proposed algorithm are two fold: allowing multiple updates per node by reinserting nodes to the active list, and using a Jacobi update for parallel computation. It turns out that the proposed algorithm is an example of a class of *label-correcting* algorithms [21]. Algorithm 4.1 is the pseudo code of FIM ($U_{\mathbf{x}}$ is a discrete approximation of $u(\mathbf{x})$, and $g(U_{\mathbf{x}})$ is a new solution at $\mathbf{x}$ that satisfies Equation 4 computed using a Godunov Hamiltonian H_G in Equation 6).

Algorithm 4.1: FIM(**X**)

```
comment: 1. Initialization (X : set of all grid nodes, L : active list)
for each x ∈ X
do { if x is source
     then U_x ← 0
     else U_x ← ∞
for each x ∈ X
do { if any neighbor of x is source
     then add x to L

comment: 2. Update nodes in L
while L is not empty
do { for each x ∈ L
     do { T_old ← U_x
          T_new ← g(U_x)
          if T_old > T_new
            then { U_x ← T_new
          if |T_old − T_new| < ε
            then { for each 1-neighbor x_nb of x
                   do { if x_nb is not in L
                        then { T_old ← U_{x_nb}
                               T_new ← g(U_{x_nb})
                               if T_old > T_new
                                 then { U_{x_nb} ← T_new
                                        add x_nb to L
                   remove x from L
```

4.2 Godunov Hamiltonian

To efficiently solve convex H-J equations, we employ a similar Godunov upwind Hamiltonian as introduced in [29]. The Godunov Hamiltonian H_G for the H-J equation on 3D grid can be defined as follows:

$$H_G(p,q,r) = \text{ext}_{p\in I[p_-,p_+]}\text{ext}_{q\in I[q_-,q_+]}\text{ext}_{r\in I[r_-,r_+]}H(p,q,r) \qquad (6)$$

where

$$\text{ext}_{x\in I[a,b]} = \min_{x\in[a,b]} \text{ if } a \le b$$
$$\text{ext}_{x\in I[a,b]} = \max_{x\in[b,a]} \text{ if } a > b$$

$p_\pm = D^x_\pm u$, $q_\pm = D^y_\pm u$, $r_\pm = D^z_\pm u$, and $I[a,b]$ is the closed interval bounded by a and b. This definition of the Godunov Hamiltonian looks complicated, but the main idea is evaluating the Hamiltonian $H(p,q,r)$ with all possible combination of $p = \{p_-, p_+, p_\sigma\}, q = \{q_-, q_+, q_\sigma\}$, and $r = \{r_-, r_+, r_\sigma\}$ where p_σ, q_σ, and r_σ are critical points (because the extremum of a convex Hamiltonian occurs only on either the end of the interval or the critical point), and taking the valid minimum solution that satisfies Equation 4. To check the validity of the solution for $H(p,q,r)$, Tsai et al. proposed the following conditions [29].

$$H(\text{sgn}\max\{(p_- - p_\sigma)^+, (p_+ - p_\sigma)^-\} + p_\sigma, q, r) = 1$$
$$H(p, \text{sgn}\max\{(q_- - q_\sigma)^+, (q_+ - q_\sigma)^-\} + q_\sigma, r) = 1$$
$$H(p, q, \text{sgn}\max\{(r_- - r_\sigma)^+, (r_+ - r_\sigma)^-\} + r_\sigma) = 1$$

Even though the above test to check the validity of the solution looks mathematically clean and works well, practically it is not efficient due to two reasons. First, this test requires three evaluations of the Hamiltonian, which is an expensive operation. Second, we need to use a threshold to numerically check the floating point equality ($|H-1| < \varepsilon$), which may induce numerical errors. The new validity test we propose is based on the observation that if the solution is valid then p, q, or r used to compute the solution must be a correct value. For example, if we use $p = p_-$, then $\text{sgn}\max\{(p_- - p_\sigma)^+, (p_+ - p_\sigma)^-\} + p_\sigma = p_-$ must hold. Checking equality for this equation can be done efficiently because we can encode the left and the right side of the equation using integers, +1, 0, and -1, and compare equality of the integers. The right side index is determined by p, and the left side index is determined by p_-, p_+, and p_σ based on the new solution.

$$\text{Right side index} = \begin{cases} 0 & \text{if } p = p_\sigma \\ +1 & \text{if } p = p_+ \\ -1 & \text{if } p = p_- \end{cases}$$

$$\text{Left side index} = \begin{cases} 0 & \text{if } p_- < p_\sigma < p_+ \\ +1 & \text{else if } (p_- + p_+)/2 < p_\sigma \\ -1 & \text{else} \end{cases}$$

The proposed test does not entail an extra burden of Hamiltonian computations, and can be done using only simple integer equality and float inequality comparisons. Our experiments show that using the new validity test can increase the performance about 50% compared to the original method [29]. The pseudo code for computing a new solution $g(U_\mathbf{x})$ is given in [9].

5 GPU Implementation

5.1 GPU FIM for H-J Solver

We have chosen the GPU to implement FIM to solve the H-J equation. The major difference between the CPU and the GPU implementation of FIM is that the GPU employs a block-based updating scheme, as proposed by Lefohn et al. [15] in the context of interactive level set computations, because the GPU architecture favors coherent memory access and control flows. The original node-based FIM (Algorithm 4.1) can be easily extended to a block-based FIM as shown in Algorithm 5.1. For a block-based update, the domain is decomposed into pre-defined size blocks (we use a 4^3 cube for 3D in the GPU implementation), and solutions of the pixels in the same block are updated simultaneously with a Jacobi update scheme. Therefore, the active list of the GPU maintains the list of active *blocks* instead of nodes.

Algorithm 5.1: GPU FIM(L, V)

```
comment: Update blocks b in active list L, V:list of all blocks
while L is not empty
do { comment: Step 1 - Update Active Blocks
     for each b ∈ L
     do { for i = 0 to n
          do { (b, C_p(b)) ← g(b)
          C_b(b) ← reduction(C_p(b))

     comment: Step 2 - Check Neighbors
     for each b ∈ L and
     do { if C_b(b) = true
          then { for each 1-neighbor b_nb of b
                 do { (b_nb, C_p(b_nb)) ← g(b_nb)
                      C_b(b_nb) ← reduction(C_p(b_nb))

     comment: Step 3 - Update Active List
     clear(L)
     for each b ∈ V
     do { if C_b(b) = false
          then { Insert b to L
```

The GPU FIM algorithm consists of three steps. First, each active block is updated with a pre-defined number of iterations. During each iteration, a new solution of Equation 4 is computed, replace the old solution if the new solution is smaller, and its convergence is encoded as a boolean value. After the update step, we perform a reduction

(Section 5.2.3) on each active block to check whether it is converged or not. If a block is converged, we mark it as *to-be-removed*. The second step is checking which neighbor blocks of to-be-removed blocks need to be re-activated. To do this, all the adjacent neighbor blocks of to-be-removed blocks are updated once, and another reduction operation is applied on each of the neighbor blocks. The final step is updating the active list by checking the convergence of each block. Only non-converged blocks (i.e., C_b is false) will remain in the active list. The pseudo code for GPU FIM is given in Algorithm 5.1 (C_p and C_b are introduced in Section 5.2).

5.2 Implementation Detail

Our GPU H-J solver is implemented on an NVIDIA GeForce 8800 GTX graphics card. NVIDIA CUDA [2] is used for GPU programming, and we will explain the GPU implementation details based on the CUDA programming model (please refer to the CUDA programming guide [2] for more details about the GPGPU programming using CUDA). Computation on the GPU entails running a kernel with a batch process of a large group of fixed size thread blocks, which fits well the block-based update method used in the FIM algorithm. We fix the block size to 4^3, so 64 threads share the same shared memory and are executed in parallel on the same processor unit.

It is not necessary to use special data structures, e.g., list or vector, to implement the active list on the GPU. Therefore, we use a simple 1D integer array whose size is the total number of blocks to store active blocks. Only the array elements of index ranging between 0 to (number of total active blocks-1) are valid at any given time. For each CUDA kernel call, the grid size is adjusted to the current number of active blocks, and when a block is being processed, its block index is retrieved from the active list on the GPU. Updating solutions and reductions, which are computationally dominant in the overall process, are done entirely on the GPU.

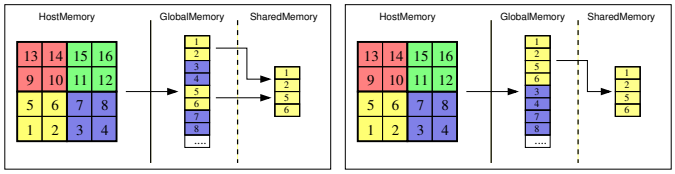
In the GPU memory, we create two sets of boolean arrays, one C_p with a size of the number of pixels (i.e., nodes), and the other C_b with a size of the number of blocks, to store convergence of pixels and blocks, in addition to a float array with a size of the number of pixels to store solutions. To check the convergence of blocks, we run a reduction on C_p to get C_b. Managing the active list, e.g., inserting or deleting blocks from the list, is efficiently done on the CPU by reading back C_b to the CPU and looping over it to insert only non-converged blocks to the active list. When the list is completely updated on the CPU, it is copied to the GPU, but only a small part of the active list is actually used at any given time (index 0 to (number of active blocks-1)), so only a small fraction of contiguous memory needs to be copied to the GPU.

5.2.1 Data Packing for Coalesced Global Memory Access

To efficiently move data from global to shared memory on the GPU, we need to pack the data in the GPU memory space in a certain way to access global memory as a single contiguous, aligned memory access (coalesced memory access [2]). Volumes are stored in memory as an 1D array with a certain traversing order. Figure 3 shows an example of two different cases of storing a 4x4 image in the GPU global memory space as 1D array when a block is copied to shared memory. Host memory is the CPU side memory, and global / shared memory is the GPU side memory. Color represents the pixels in the same block. Usually pixels are stored from the fastest to the slowest axis order, as shown in Figure 3 (a). In this case, a block is split into two regions in global memory space, which leads to split block accesses. However, if we re-order global memory as shown in Figure 3 (b), accessing a block can be a single coalesced memory access, which is the most efficient way to access global memory on the GPU. Hence, whenever input volumes are copied from the CPU to the GPU memory, a proper re-ordering should be applied so that the block access can be done through a coalesced memory access.

5.2.2 Efficient Neighbor Access using Shared Memory

Another factor that affects the GPU performance is accessing shared memory. The shared memory space in the NVIDIA G80 architecture is divided into 16 banks, and 16 shared memory accesses can be done simultaneously as long as all the memory requests refer to different memory banks or to the same memory bank. If any two memory requests, but not all, refer to the same memory bank, i.e., bank conflict, then this request must be serialized, and this impairs the performance. Because the block size is fixed to 4^3, there is no bank conflict to access pixels inside blocks (block size is a multiple of warp size [2]). However, we need adjacent neighbor pixels to solve the PDEs, so we should set up an additional shared memory space for left/right/up/down/top/bottom neighbors of the boundary pixels of each block.



(a) Non-coalesced (b) Coalesced

Fig. 3. Example of coalesced/non-coalesced global memory access

To avoid bank conflicts, we assign the neighbor pixels to predefined banks, which requires a slightly larger extra shared memory space. Figure 4 shows a 2D example of the bank assignment that avoids bank conflicts for neighbor pixel access. The block size for this example is 16 (4x4), which is drawn as a yellow box on the leftmost image in Figure 4. The extra four pixels on each left/right/up/down side of the block are neighbor pixels. The number on each pixel represents the bank number to be assigned. By assigning pixels to shared memory in this pattern, memory requests for left/right/up/down neighbors can be done simultaneously without a bank conflict (Figure 4 red : left neighbors, cyan : right neighbors, green : up neighbors, blue : down neighbors). We need shared memory of size 3*blocksize to store a block and its neighbors because some bank numbers appear twice (1, 4, 13, and 16 in Figure 4). Figure 5 shows an example of actual pixel assignment in shared memory. Figure 5 (a) shows a 2D block diagram with pixel indices (not bank numbers). Figure 5 (b) shows which bank each pixel is actually assigned to. Figure 5 (c) shows a snapshot of a shared memory access pattern when left neighbors are accessed (same case as the second diagram from left in Figure 4). Pixels colored in red are accessed by 16 threads in parallel. Because there is no bank conflict, this memory request can be processed simultaneously. The 2D bank assignment technique can be easily extended to 3D by using slightly larger shared memory for storing top and bottom neighbors.

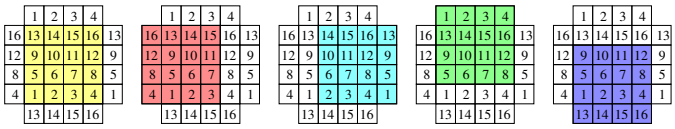
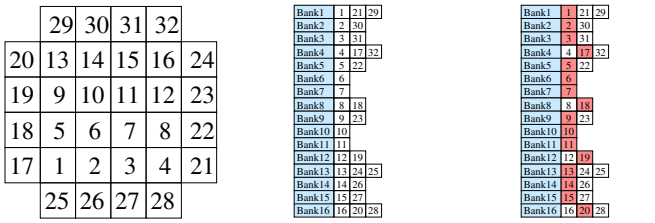


Fig. 4. Neighbor pixel access without shared memory bank-conflict



(a) Pixel indices (b) Bank assignment (c) Left neighbors

Fig. 5. Bank assignment example

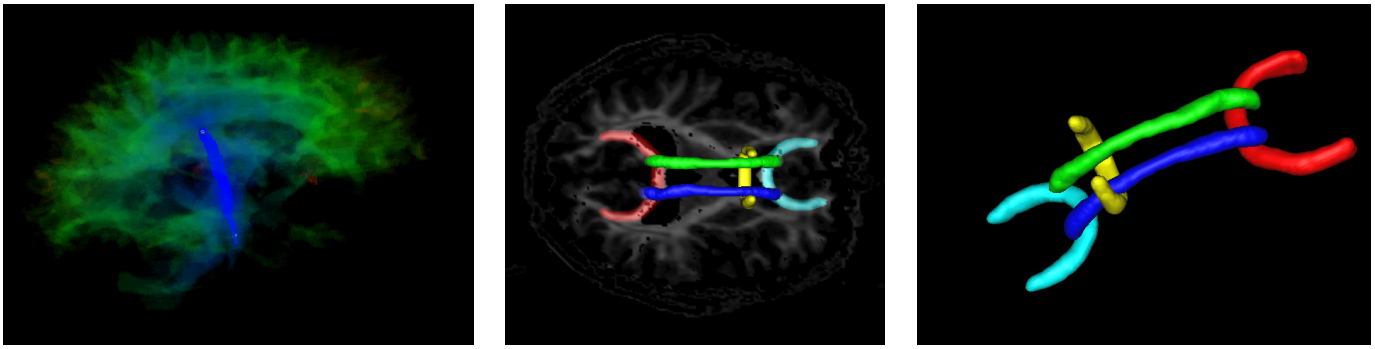


Fig. 6. Volumetric pathways extractions using the proposed method. From left to right: a) cortocospinal tract; b) isosurface rendering of volumetric paths shown with FA slice; c) isosurface rendering of volumetric paths only. GCC(cyan), BCC(yellow), SCC(red), LCG(green), and RCG(blue). The SCIRun software [1] is used to render final isosurfaces.

5.2.3 Reduction

Reduction is one of the commonly used computational techniques in the streaming programming model to produce a smaller stream from a larger input stream. To check the convergence of a block, we need to check the convergence of every pixel in the block. Therefore, we need to reduce a block down to a single pixel that represents the convergence of the block. Lefohn et al. proposed a reduction on the GPU using a fragment shader to check the activity in blocks in [15]. We implemented a parallel reduction in a single kernel execution using a block-level thread synchronization provided by CUDA. To reduce a block of size n, start with $\frac{n}{2}$ threads. For each iteration, every thread participating in reduction reads two convergence values from the current block and write a true or false to one of the original locations (both converge : true, else false). In the next iteration, the number of participating threads is halved and the same reduction is performed. This process is repeated until a block is reduced to a single pixel.

6 Results

6.1 Connectivity Visualization

To demonstrate our interactive DTI connectivity visualization for real clinical data, we applied our method to a single high-resolution ($2 \times 2 \times 2.5\text{mm}^3$) 3T image from a database of healthy controls. Using an interface in which points can be selected interactively on the tensor data set, we selected the terminal regions R_1 and R_2 at the white/grey matter interface for each tract we analyzed.

Figure 6 (a) shows the interactive path extraction using two selected points along the cortocospinal tract. A volume rendering of the total cost function u is displayed with blue representing lowest cost regions. The total cost function represents the strength, or integrity, of the connection, and this visualization could be used by clinicians to locate degradations in the white matter. The cortocospinal tract, which is a major pathway connecting the spinal cord to the motor cortex, has high connectivity and thus shows up as bright blue. Next, we selected five tracts for visualization: three bundles through the genu (GCC), splenium (SCC), and body (BCC) of the corpus callosum, and the left (LCG) and right (RCG) cingulum bundles. Figure 6 (b) and (c) show isosurface renderings of the resulting volumetric pathways from the five selected tracts. These volumetric pathways can then be used to define a culling region for displaying the tensor data only in the pathways of interest, as shown in Figure 7. In [6] the authors describe nonparametric methods for quantifying DTI and geometric information along tracts.

6.2 Running time comparison

Table 1 and Figure 8 show the running time of three H-J equation solvers (GPU, CPU Fast Sweeping with Godunov Hamiltonian, and CPU Fast Sweeping with Lax-Friedrichs Hamiltonian) and their solutions on three synthetic and real tensor volumes. We have tested H-J solvers on a PC equipped with a Intel Core 2 Duo 2.4GHz processor and an NVIDIA GeForce 8800 GTX graphics card.

Example 1 is a 64^3 volume with a constant tensor elongated along the diagonal direction ($a = d = f = 1.0$ and $b = c = e = 0.9$). The level sets of the solution on this volume is shown in Figure 8 (a). The GPU solver took only 1 sec while the CPU solvers take about 1–2 minutes to compute the solution on this volume.

Example 2 is a 64^3 volume with tensors aligned to a helix. We built a tensor whose dominant eigenvector is parallel to the tangent vector of the helix curve, and set the dominant eigenvalue as 1.0 and the other two eigenvalues as 0.0001. Figure 8 (b) is the solution and characteristic paths tracing from randomly distributed points to the seed point placed on the center of the bottom slice. The GPU solver took 1.5 second, while the CPU solvers took 1–3 minutes on this volume.

Example 3 is a DT-MRI brain volume of size 256x256x100, with the number of effective pixels is 196K (we only run the solver inside the white matter mask), and the solution is given in Figure 8 (c). We put a seed at the center of the white matter region. The GPU solver runs less than 3 seconds while the CPU solver took 5 minutes on this volume. Overall, the proposed GPU H-J solver runs roughly 50–100 times faster than the commonly used CPU-based methods, allowing users for interactive volumetric paths extraction in DT-MRI volumes.

	Example 1	Example 2	Example 3
GPU FIM	1 sec	1.5 sec	2.8 sec
CPU FS Gdv	54 sec	76 sec	301 sec
CPU FS L-F	142 sec	220 sec	N/A

Table 1. Running time on 3D tensor volumes. L-F was not tested on example 3 due to its complex boundary.

6.3 Discussion

The proposed GPU-FIM algorithm, in contrast to other label-correcting algorithms [21], scales well on massively parallel architectures by adopting a block-based parallel updating scheme. The proposed method uses the same discretization of Godunov Hamiltonian that is used in [29], so the solutions are exactly same as the fast sweeping methods. However, due to the hardware floating point implementation on different architectures (CPU and GPU), there is a small numerical difference between the solutions of those methods, but the difference is minimal and does not affect the quality of the extracted paths.

Even though this paper focuses mainly on the computational aspect of the white matter pathway visualization, it is worth mentioning the pros and cons of the proposed method compared to the existing tractography methods. Global tractography gives interesting holistic visualizations of white matter architecture, which may be useful to the uninitiated. In contrast, the proposed method can provide a systematic method to query the quality of a particular path of interest, which is useful to experienced neuroscientists who already know the white matter architecture but wish to be able to easily and repeatably iden-

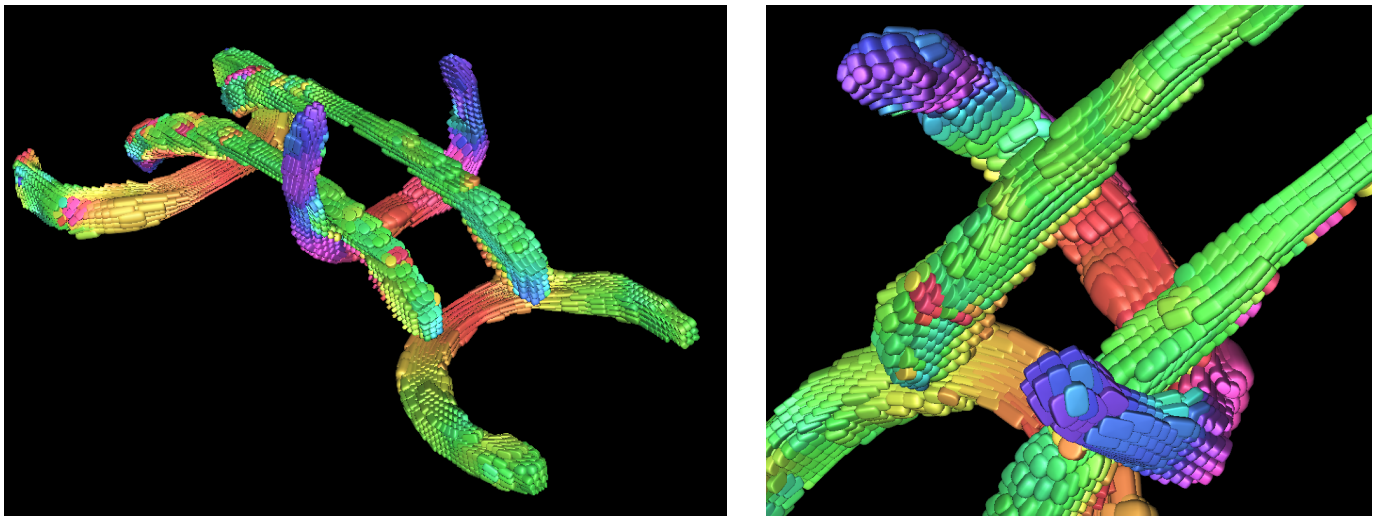


Fig. 7. Tensor field visualization along the volumetric pathways using superquadric glyphs [12] using the SCIRun software [1]. From left to right: a) tensor visualization; b) zoom in view near BCC.

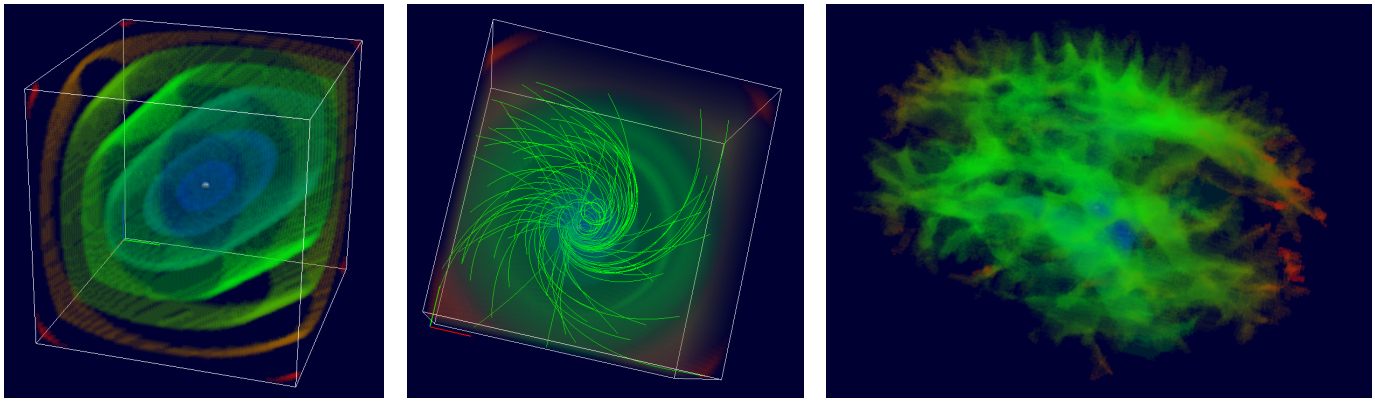


Fig. 8. Visualization of distance from a seed point. From left to right: a) Example 1 : tensor elongated toward diagonal direction; b) Example 2 : helix; c) Example 3: DT-MRI brain data.

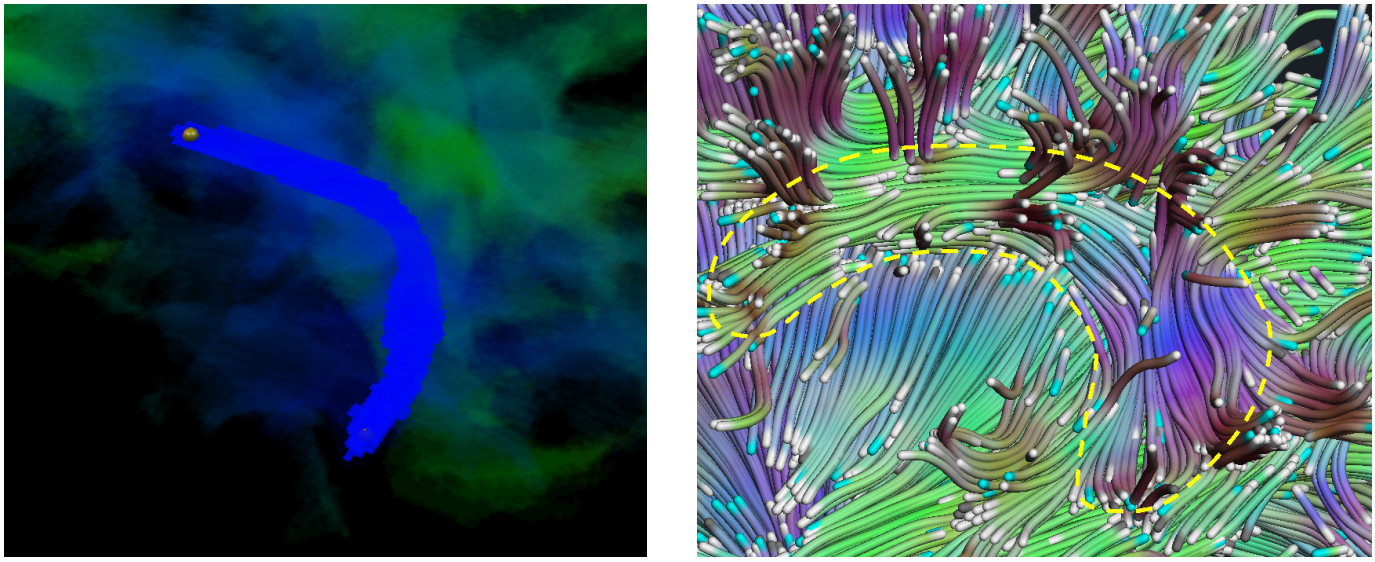


Fig. 9. Comparison between the proposed method and tractography method to extract the arcuate fasciculus connecting frontal and temporal cortex. Left : Volumetric path from the proposed method. Right : Paths from the tractography method. Yellow dotted curve represents the arcuate fasciculus area.

tify white matter circuitry across populations of subjects or patients. In addition, tractography is sensitive to noise and oblique shapes in DT-MRI, while the proposed method is robust to them. A nice example is the arcuate fasciculus, which connects Wernicke's and Broca's areas (frontal and temporal cortex) shown in figure 9. This tract is not easy to find because it crosses through some ambiguous regions as it descends to the temporal lobe (temporoparietal junction). Figure 9 right shows a close up picture of a typical result from tractography that shows the difficulty of locating tracts that connect these two relatively small regions, especially where the alignment of the tensors become ambiguous. The tractography result ends abruptly, and none of the many thousands of tracks we seeded successfully connected these two regions. In contrast, the proposed method robustly finds the optimal path between the end points that minimizes the total cost value, which are much less affected by noise and ambiguity (figure 9 left).

7 Conclusion

A fast method to compute and visualize white matter connectivity in DT-MRI volume using graphics hardware is presented. The proposed method computes the connectivity as the minimum distance from a given source region, and define the total cost by summing the connectivity value from each source region. The proposed method finds a volumetric optimal path as a set of voxels in the DTI volume that contain paths between two regions whose costs are within a threshold of the optimal path. To quickly compute the cost function, we introduce a novel GPU H-J solver, which runs roughly 50-100 times faster than existing CPU-based solvers, allowing users to freely change the endpoints of interesting pathways and to visualize the optimal volumetric path between them at an interactive rate.

Introducing a fast volumetric connectivity extraction tool opens up numerous interesting future research directions. Because the GPU implementation of the FIM allows rapid computation of white matter connections, this makes computation of a full brain connectivity map feasible. This process could be automatically initialized with starting regions from a parcellation of the cortical surface, such as that produced by Freesurfer [5]. The proposed GPU H-J solver can be also applied to wider range of applications other than DT-MRI image analysis. For example, seismic wave propagation simulation in an anisotropic speed volume will be a direct application of the GPU H-J solver for geoscience research. Extensive performance analysis and comparison of the GPU solver with existing solvers will be another interesting future work.

Acknowledgements

This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149. This work has been supported in part by ExxonMobil Upstream Research Company. The visualizations in Figure 6 and 7 in this paper were produced with the BioPSE/SCIRun software package released by the Center for Integrative Biomedical Computing, NIH NCRR Project 2-P41-RR12553-07

References

[1] SCIRun: A scientific computing problem solving environment, Scientific Computing and Imaging institute (SCI), 2002. http://software.sci.utah.edu/scirun.html.

[2] NVIDIA CUDA programming guide, 2007. http://developer.nvidia.com/object/cuda.html.

[3] P. J. Basser, S. Pajevic, C. Pierpaoli, J. Duda, and A. Aldroubi. In-vivo fiber tractography using DT-MRI data. *Magnetic Resonance in Medicine*, 44:625–632, 2000.

[4] 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.

[5] B. Fischl, A. van der Kouwe, C. Destrieux, E. Halgren, F. Sgonne, D. H. Salat, E. Busa, L. J. Seidman, J. Goldstein, D. Kennedy, V. Caviness, N. Makris, B. Rosen, and A. M. Dale. Automatically parcellating the human cerebral cortex. *Cerebral Cortex*, 14(1):11–22, 2004.

[6] P. T. Fletcher, R. Tao, W.-K. Jeong, and R. T. Whitaker. A volumetric approach to quantifying region-to-region white matter connectivity in Difusion Tensor MRI. In *Information Processing in Medical Imaging 2007 Conference Proceedings (to appear)*, 2007.

[7] K. E. Hoff III, J. Keyser, M. Lin, D. Manocha, and T. Culver. Fast computation of generalized Voronoi diagrams using graphics hardware. In *SIGGRAPH 1999 Conference Proceedings*, pages 277–286, 1999.

[8] M. Jackowski, C. Y. Kao, M. Qiu, R. T. Constable, and L. H. Staib. Estimation of anatomical connectivity by anisotropic front propagation and diffusion tensor imaging. In *MICCAI*, pages 663–667, 2004.

[9] W.-K. Jeong and R. T. Whitaker. A fast iterative method for a class of Hamilton-Jacobi equations on parallel systems. Technical Report UUCS-07-010, University of Utah, 2007. http://www.cs.utah.edu/research/techreports.shtml.

[10] C. Kao, S. Osher, and J. Qian. Lax-friedrichs sweeping scheme for static Hamilton-Jacobi equations. *Journal of Computational Physics*, 196(1):367–391, 2004.

[11] C. Kao, S. Osher, and Y. Tsai. Fast sweeping methods for static Hamilton-Jacobi equations. Technical report, Department of Mathematics, University of California, Los Angeles, 2002.

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

[13] M. A. Koch, D. G. Norris, and H.-G. M. An investigation of functional and anatomical connectivity using magnetic resonance imaging. *NeuroImage*, 16:241–250, 2002.

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

[15] A. Lefohn, J. Kniss, C. Hansen, and R. Whitaker. Interactive deformation and visualization of level set surfaces using graphics hardware. In *IEEE Visualization 2003 Conference Proceedings*, pages 75–82, 2003.

[16] L. O'Donnell, S. Haker, and C.-F. Westin. New approaches to estimation of white matter connectivity in diffusion tensor MRI: elliptic PDEs and geodesics in a tensor-warped space. In *MICCAI*, pages 459–466, 2002.

[17] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):80–113, March 2007.

[18] G. Parker, C. Wheeler-Kingshott, and G. Barker. Estimating distributed anatomical connectivity using fast marching methods and diffusion tensor imaging. *Transactions on Medical Imaging*, 21:505–512, 2002.

[19] G. J. M. Parker, H. A. Haroon, and C. A. M. Wheeler-Kingshott. A framework for a streamline-based probabilistic index of connectivity (PICo) using a structural interpretation of MRI diffusion measurements. *Journal of Magnetic Resonance Imaging*, 18:242–254, 2003.

[20] E. Pichon, C.-F. Westin, and A. Tannenbaum. A Hamilton-Jacobi-Bellman approach to high angular resolution diffusion tractography. In *MICCAI*, pages 180–187, 2005.

[21] L. C. Polymenakos, D. P. Bertsekas, and J. N. Tsitsiklis. Implementation of efficient algorithms for globally optimal trajectories. *IEEE Trans. Automatic Control*, 43(2):278–283, 1998.

[22] F. Qin, Y. Luo, K. Olsen, W. Cai, and G. Schuster. Finite-difference solution of the eikonal equation along expanding wavefronts. *Geophysics*, 57(3):478–487, 1992.

[23] E. Rouy and A. Tourin. A viscosity solutions approach to shape-from-shading. *SIAM Journal of Numerical Analysis*, 29:867–884, 1992.

[24] J. Sethian. A fast marching level set method for monotonically advancing fronts. In *Proc. Natl. Acad. Sci.*, volume 93, pages 1591–1595, February 1996.

[25] J. Sethian. Fast marching methods. *SIAM Review*, 41(2):199–235, 1999.

[26] J. A. Sethian and A. Vladimirsky. Ordered upwind methods for static Hamilton-Jacobi equations: Theory and algorithms. *SIAM Journal of Numerical Analysis*, 41(1):325–363, 2003.

[27] C. Sigg, R. Peikert, and M. Gross. Signed distance transform using graphics hardware. In *IEEE Visualization 2003 Conference Proceedings*, pages 83–90, 2003.

[28] A. Sud, M. A. Otaduy, and D. Manocha. Difi: Fast 3d distance field computation using graphics hardware. *Computer Graphics Forum*, 23(3):557–566, 2004.

[29] Y.-H. R. Tsai, L.-T. Cheng, S. Osher, and H.-K. Zhao. Fast sweeping algorithms for a class of Hamilton-Jacobi equations. *SIAM Journal of Numerical Analysis*, 41(2):659–672, 2003.

[30] H. Zhao. A fast sweeping method for eikonal equations. *Mathematics of Computation*, 74:603–627, 2004.