

EE 574 PROJECT

**FIBER TRACTOGRAPHY USING BRAIN DT-MRI DATA AND  
VTK VISUALIZATION**

*MID-REPORT*

*Submitted to*  
**Dr. Burak ACAR**

*Submitted by*  
**Sancar ADALI, Murat AKSOY, Hasan AYAZ**

# 1-Introduction

Diffusion Tensor Imaging (DTI) is an MRI technique that gives information about the random thermal motion of water molecules in the human brain. The diffusion of water molecules in the human brain is different in white and gray matter regions. In white matter regions, where axons are similarly aligned, diffusion is mostly anisotropic in the direction of axons of the neurons whereas in gray matter regions, where less ordered tissue structure is observed, it is mostly isotropic. This property enables to distinguish between white and gray matter regions and with DTI, pathological changes such as those occurring in multiple sclerosis, tumors etc. can be observed. [1]

Despite the fact that DTI data gives information about the direction of each voxel, it gives no information about the connections between different voxels. Different techniques are employed in constructing this connection, which will be the main idea of this project. Tensor data assigns a matrix to each point to the 3D brain image. The eigenvalues of these matrices give information about the amount of diffusion and the eigendirections are associated with the directions of the diffusion. Two methods will be employed in tracting fiber trajectories :

The first method assumes the dominance of one of these eigenvalues relative to the others, which means that the other eigenvalues have values 0 and the diffusion is anisotropic exactly in the direction of the eigenvector corresponding to the highest eigenvalue. These eigenvectors corresponding to the highest eigenvalues will be tractred using Euler's and 4th order Runge-Kutta methods. The stopping criteria in this method are reaching the image boundaries or anisotropy falling under a specific threshold. [2] & [3]

Fast marching algorithm will be implemented as the second step of the project. The fast marching algorithm utilizes rules of the level set theory and includes the evolution of a closed region up to the boundaries of the structure which it's in over time. The boundary of this region (which is also called the front) propogates in the direction where the eigenvector corresponding to the highest eigenvalue and the normal to the front at that point are most similar (i.e. are in the same direction). By starting from many seed points and propogating according to these rules, every voxel in the image is assigned an arrival time, which can be used as a measure for the connectedness of these points and the seed such that, points that have smaller arrival times are more related to the seed points than those that have higher arrival times. The connection between these points within the arrival time and the seeds points are established by minimizing the arrival time T between these points by steepest descent method. [4] B-Spline interpolation will also be implemented to provide smoothness of the fiber tracts.

## 2- Methodology

To describe the method of fiber tractography mathematically, the voxels are represented as 3D vectors in the following formula:

$$r_{n+1} = r_n + h * V_n$$
 where  $r_n$  is the vector corresponding to the voxel at nth step when starting from  $r_0$  , h is the step size parameter and  $V_n$  is the vector in the direction the tract will follow [3]. The objective here is to find  $V_n$  for any n given initial conditions  $r_0$  and  $V_0$ . The most basic method to do that is Euler's Method.

Euler Method solves the problem of ordinary differential equations by assuming a constant step size and evaluating the function  $f(x,y)$  at  $y_n$  in the differential equation  $dy/dx=f(x)$  in order to find the finite difference and add to  $y_n$ . Of course we will be using generalization of

this method to 3D and our finite difference  $V_n$  will be the eigenvector corresponding to the maximum eigenvalue at the nth step.

Runge-Kutta Methods are generalizations of Euler's method for finding smoother approximations of the solution of differential equations. We have started implementing fourth order Runge-Kutta Methods that could be formulated as:

$$V_n = \frac{1}{6}(\vec{k}_1 + 2\vec{k}_2 + 2\vec{k}_3 + \vec{k}_4) \quad \text{where } k_1, k_2, k_3, k_4 \text{ are found as:}$$

$$\vec{k}_1 = \frac{\vec{V}_0 \cdot \vec{E}(\vec{r}_0)}{|\vec{V}_0 \cdot \vec{E}(\vec{r}_0)|} E(\vec{r}_0)$$

$$\vec{k}_2 = \frac{\vec{V}_0 \cdot E(\vec{r}_0 + \frac{h}{2} * \vec{k}_1)}{|\vec{V}_0 \cdot E(\vec{r}_0 + \frac{h}{2} * \vec{k}_1)|} E(\vec{r}_0 + \frac{h}{2} * \vec{k}_1)$$

$$\vec{k}_3 = \frac{\vec{V}_0 \cdot E(\vec{r}_0 + \frac{h}{2} * \vec{k}_2)}{|\vec{V}_0 \cdot E(\vec{r}_0 + \frac{h}{2} * \vec{k}_2)|} E(\vec{r}_0 + \frac{h}{2} * \vec{k}_2)$$

$$\vec{k}_4 = \frac{\vec{V}_0 \cdot E(\vec{r}_0 + h * \vec{k}_3)}{|\vec{V}_0 \cdot E(\vec{r}_0 + h * \vec{k}_3)|} E(\vec{r}_0 + h * \vec{k}_3)$$

where  $\vec{E}(\vec{r}_0)$  is the continuous 3D function interpolated from DT-MRI maximum eigenvector data.

Fast Marching Method will be implemented as the third part of the project. In the fast marching algorithm, the main purpose is to track the motion of an interface as it evolves. If we denote  $\gamma(t)$  as the family of curves as the initial curve  $\gamma(0)$  evolves in the direction of the normal vector with speed F, then this evolution of the curve can be formulated as :

$$\vec{x}_t = F(\kappa) \cdot \vec{n}$$

Where  $\kappa$  is the curvature of the curve at that point and  $n$  is the unit normal to the curve. This function simply states that the change in the curve with respect to time is the most in areas where the curvature-dependent speed function and the unit normal to the curve are most aligned.

However, in such a curve evolution, for  $F=1$ , almost in all of the curves, the smoothness of the curve is lost, and we require a weak solution, because the solution weakly satisfies the definition of differentiability.(A weak solution is the one that satisfies the integral form of the original equation. Due to the fact that such a solution does not require the solution

to be differentiable with respect to the independent variable, the possible range of solutions broadens).

If we think of the curve as an interface separating two regions, the solution we want is the shortest distance or the first arrival. In another sense, the set of points in the front of the curve must always correspond to the points that are located at a distance  $t$  away from the original curve. A way to obtain this solution is through enforcing an “entropy” condition for the propagating interface.

If we take the speed function in the form :

$$F = 1 - \varepsilon \kappa$$

For  $\varepsilon > 0$ , we will have a smooth flow. The limit of this solution as  $\varepsilon \rightarrow 0$ , which is also known as the viscous limit, will be the entropy solution for the constant speed case. In order to find this solution, techniques used in hyperbolic conservation laws are used. An equation for  $u(x,t)$  of the form :

$$u_t + [G(u)]_x = 0$$

is known as the hyperbolic conservation law. The solution to these equations can develop discontinuities, known as “shocks”. In order to avoid these sudden compressions or expansions, a diffusive term is added to the right side of this equation. For example, in the case of Burger’s equation :

$$u_t + uu_x = \varepsilon u_{xx}$$

The term  $\varepsilon u_{xx}$  acts as a smoothing term and for  $\varepsilon > 0$ , the solution remains smooth for all time.

In the solution of the “hyperbolic conservation law”, the curves on which the solution  $u(x,t)$  always stays the same are called characteristics. In some cases, these characteristics may exhibit converging behaviour, known as ‘shocks’; or they may diverge, leaving a “gap” in the  $(x,t)$  plane. The solution to this problem is to choose  $u(x,t)$  to be the one obtained as the limit of the solution as the diffusive term added to the right of the equation (the term  $\varepsilon u_{xx}$  in the case of Burger’s equation) vanishes. This solution is similar to the one that we want to obtain for the curve evolution equation. Thus, the solution schemes obtained for the hyperbolic conservation law can also be applied to our problem.

Integration of the hyperbolic equation allows us to get the conservation form of the equation:

$$\frac{d}{dt} \int_a^b u dx = G(u(a,t)) - G(u(b,t))$$

Our aim is to construct weak solutions to this conservation form which also satisfy the entropy condition and don’t contain any discontinuities. A scheme is in conservation form if there exists a numerical flux function  $g(u_{i-1}, u_i)$  or  $g(u_i, u_{i+1})$  which approximates the values for  $G_{i-1/2}$  or  $G_{i+1/2}$  such that

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{G_{i+1/2} - G_{i-1/2}}{\Delta x}$$

Such a solution confines the shocks to a few grid points. The entropy condition can also be satisfied by choosing a scheme that takes three arguments  $u_i^n, u_i^{n-1}, u_i^{n+1}$ , and that is a monotonically nondecreasing function of its arguments. Such a conservative, monotone scheme also satisfies the entropy condition. Thus, in order to construct a viable scheme, we need to make sure that it is in conservation form and it is a monotone increasing function of its arguments. One of the approximating functions used in a scheme is :

$$g(u_1, u_2) = (\max(u_1, 0))^2 + (\min(u_2, 0))^2$$

This entropy-satisfying and relatively few diffusing causing scheme in the conservation form allows us to approximate the gradients in both the initial value and boundary value formulations. After finding this gradient, we can come back to the relation between the level set methods and fast marching.

The main challenge in this part of the project is to track a surface that is evolving. Assume that our initial surface is denoted by  $\Gamma$ . Now consider the function  $\phi(x, t) = \mp d$ , where  $d$  is the distance of the point  $x$  from the evolving curve at time  $t$ . Plus sign indicates that  $x$  is outside the surface, and minus sign says it is inside. Thus, it is possible to say that the zero level set of the function  $\phi(x, t = 0)$  gives us the original curve  $\Gamma$ . By chain rule, an equation for the evolution of the interface can be produced :

$$\begin{aligned} \phi_t + F|\nabla\phi| &= 0 \\ \phi(x, t = 0) &= \text{given} \end{aligned}$$

This partial differential equation states that our initial curve changes (that is, evolves) with respect to time, and this evolution is in the direction of the negative gradient of the evolving surface and weighted by the speed function. This provides an outward propagation when  $F > 0$ .

As described before, a careful approximation of the gradient must be used in order to get a correct weak solution. The application of the result found above to the gradient in our case produces :

$$\phi_{ij}^{n+1} = \phi_{ij}^n - \Delta t (\max(D_{ij}^{-x}\phi, 0)^2 + \min(D_{ij}^{+x}\phi, 0)^2 + \max(D_{ij}^{-y}\phi, 0)^2 + \min(D_{ij}^{+y}\phi, 0)^2)$$

$$\text{where } D_{ij}^{+x}\phi = (\phi_{i+1,j} - \phi_{ij}) / \Delta x$$

However, in the formulation above, when we try to get the solution for  $\phi(x, t)$ , we find the solution for all  $x$ 's, not just for the level set that corresponds to our evolving curve (remember our curve corresponds to level set for  $x=0$ ). This procedure is computationally expensive. Narrow band approaches are applied in order to make up for this cost.

In the narrow band approach, operations are only performed on the neighbouring the zero level set. The main idea in this approach is to name the points as alive, if they are inside the band; "land mines", if they are near its boundary or "far away", if they are outside the band. Work is performed only on the alive points, and the band is reconstructed once land mine points are reached. An extreme one cell version of this method leads to the "fast marching algorithm".

Consider a front moving with the speed function  $F=F(x,y,z)$

$$\begin{aligned} \phi_t + F(x, y, z)|\nabla\phi| &= 0 \\ \phi(x, t = 0) &= \Gamma \end{aligned}$$

Now consider the two dimensional case where the evolving structure is a curve. If we denote  $T(x,y)$  as the time in which the evolving curve crosses the point  $(x,y)$  :

$$|\nabla T|F = 1$$

This is a form of the Eikonal equation, and the viscosity solutions are connected with the solution of this equation, where, as has already been stated above, selection of monotone and consistent schemes will lead to the schemes that select correct viscous limit.

In the fast marching equation, for the approximation to the gradient for  $T$ , the following method, proposed by Rouy and Tourin, is used (instead of the difference approximation that was proposed above):

$$\left[ \begin{array}{l} \max(D_{ijk}^{-x}T, -D_{ijk}^{+x}T, 0)^2 + \\ \max(D_{ijk}^{-y}T, -D_{ijk}^{+y}T, 0)^2 + \\ \max(D_{ijk}^{-z}T, -D_{ijk}^{+z}T, 0)^2 \end{array} \right]^{\frac{1}{2}} = \frac{1}{F_{ijk}}$$

The equation above is a quadratic equation for  $T_{ijk}$ , and can be solved by iteration. One can start with initial  $T_{ijk}$  values for all grids, and by looking at the neighbouring  $T_{ijk}$  values, (namely  $T_{i-1,j,k}$   $T_{i+1,j,k}$   $T_{i,j-1,k}$   $T_{i,j+1,k}$   $T_{i,j,k-1}$   $T_{i,j,k+1}$ ), every  $T$  value for each grid is updated, and this process containing all the grid points is repeated  $n$  times until convergence. Such an operation requires one loop for the iteration and three loops for the coordinates, thus making the computational complexity as high as  $O(N^4)$ .

The fast marching algorithm proposes the use of the causality relation in this algorithm. In order to explain this, we should first consider the nonnegativity of the left side of the equation (it can be seen that the outcome of any of the max operations is a number that is equal to or greater than zero). In order to understand this more clearly, consider only one dimensional version of this equation, i.e. :

$$\max(D_i^{-x}T, -D_i^{+x}T, 0) = \frac{1}{F_i}$$

If the curve (or in our case the point) is propogating to the right, the first term in the max operation is taken, and the equation becomes :

$$\frac{T(x) - T(x - \Delta x)}{\Delta x} = \frac{1}{F_i}$$

In this case,  $T(x)$  is always greater than  $T(x - \Delta x)$  (of course  $F_i > 0$ ). Thus, the value of  $T$  always increases as the point propogates to right. In the case of a propogating curve to the left, the term in the middle is always chosen, and the same reasoning applies. The 0 term only has a regularizing role.

Thus, the Fast Marching method relies on propogating the curve starting from the smallest  $T$  value. By looking at the narrow band values around our initial curve, we expand the curve by taking the grid that has the smallest  $T$  value, and freeze this point. After that, new points are added to the set of narrow band points, and the same operation is repeated over and over. Since

the recomputation of T at any of the new narrow band points cannot yield results smaller than the frozen points, we can systematically move forward. This means that we won't have to go back and revisit a frozen point and compare its T value with the new T values.

The fast marching algorithm can be summarized as follows :

- 1) Set the initial points as "known"
- 2) Tag the points that are 1 pixel away as "trial"
- 3) Tag all other points as "far away"
- 4) Determine the trial point with the smallest T value
- 5) Add this point to the known, remove it from trial
- 6) Tag as "trial" that are the neighbours of the new added point and that are not "known"
- 7) Recompute the T values of the new points by solving the quadratic equation
- 8) Return to 4

For the fiber tractography application of this algorithm, the speed function F will be :

$$F(r) = A|\varepsilon_1(r) \cdot n(r)|$$

where  $\varepsilon_1(r)$  is the eigenvector corresponding to the largest eigenvalue, and  $n(r)$  is the unit normal to the curve.

The propagation of the curve will start from initial "seed" points. The arrival times for all the other points from these initial points will be calculated by the algorithm defined above, and we are going to have a map filled with arrival times. Among these, the points that are well connected to the seeds will have smaller arrival times. After choosing another point on the map, the most likely connection between this point and the seeds will be formed by following the steepest descent of T from the chosen point and the seed point. Thus, the most likely connection between a point and the seed point will be determined.

### **3-Visualization & Programming Environment**

Working environment is selected to be Windows Operating System. There are two main methods to create a C++ executable in Microsoft Visual Studio. First method uses Win32 SDK and the second uses MFC (Microsoft Foundation Classes). Both has pros and cons and both are the most efficient and flexible methods for creating Windows executables. Among these two methods, Win32 SDK selected because of gives direct access Win32 API (Application Programming Interface). Win 32 API is the underlying core component of Windows Operating System over the Hardware Abstraction Layer. MFC, also access it through Win 32 SDK and appends extra code for automatic handling of the hierarchical item and garbage collection. Thus using Win32 SDK, we can have a more efficient executables in terms of both performance and executable size.

VTK 4.0 (Visualization Toolkit) Library is being used for rendering and processing 2D and 3D images for the program. VTK is used as a set of C++ class library in the project. The library visualization system that supports a wide variety of visualization algorithms including

scalar, vector, tensor, texture, and volumetric methods; and advanced modeling techniques like implicit modelling and mesh smoothing.

The user interface designed so far has been given in figure 1. In the program, for demonstration purposes, only one frame is loaded. So the image shown in axial, coronal and sagittal views are the same frame in different resolutions.

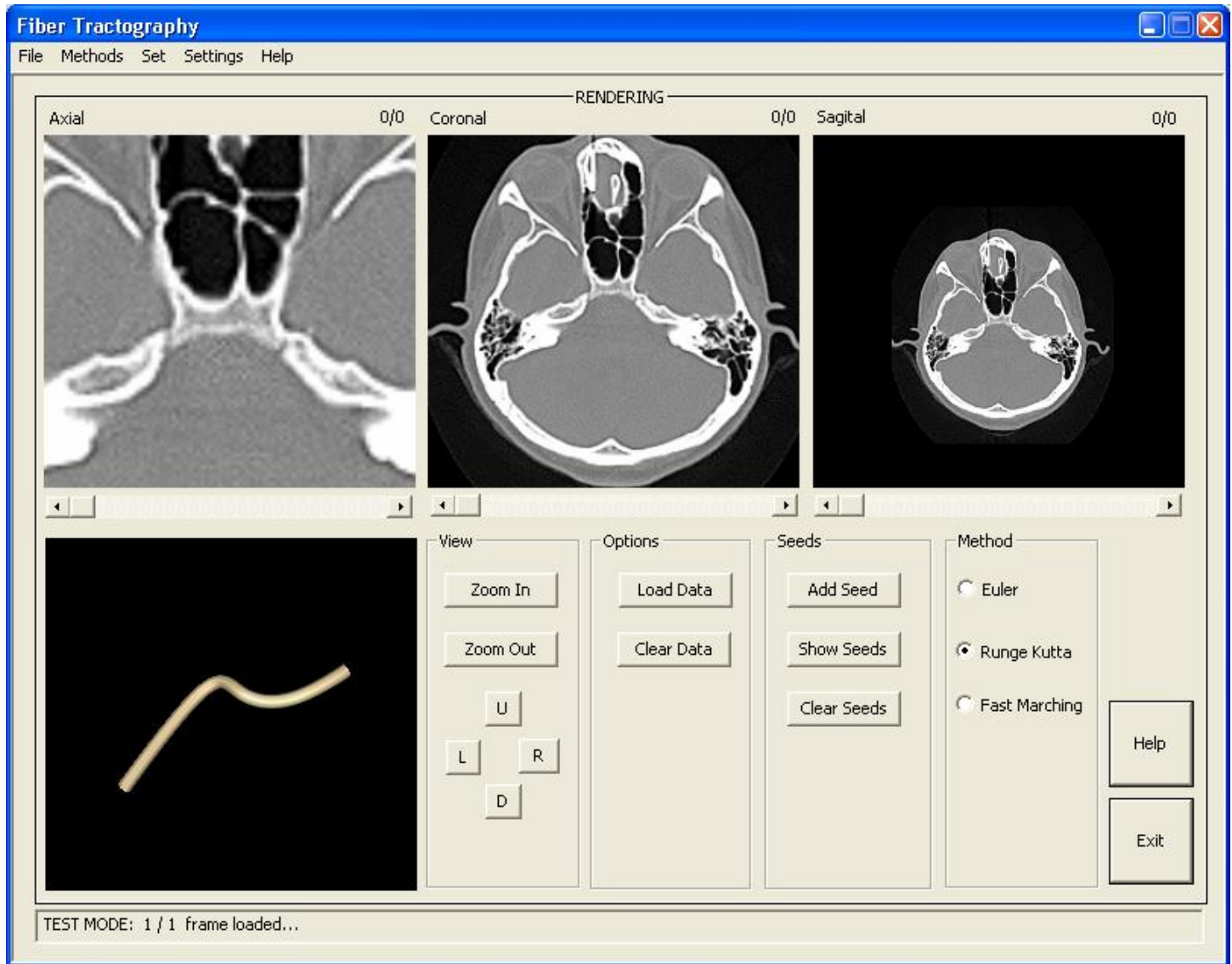


Figure 1. FT main dialog (user interface)

The image shown in the bottom left image is a 3D model of a stream tube. It is interpolated from 7 points that are entered manually to the program code for demonstration purposes. These seven input points are then increased to 80 points using cardinal spline interpolation.

Looking at the basic structure of the vtk library, it is understood that it is a higher visualization system than OpenGL which is a low level computer graphics rendering system. The architecture of our system is summarized in figure 2. As seen in the figure, our application has both accesses to Win32 API and Vtk library.

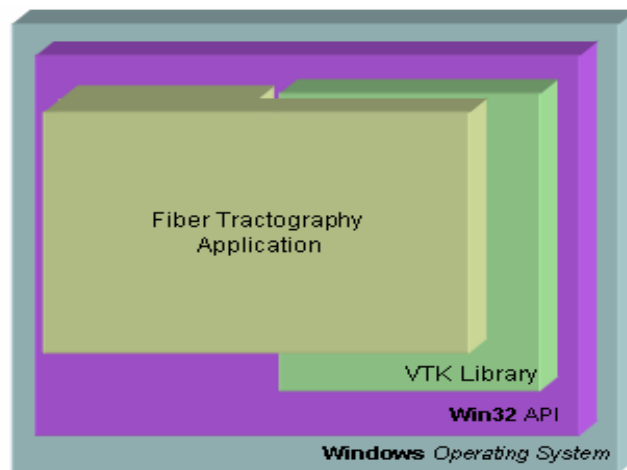


Figure 2

To create a rendering pipeline, we have to interact with some vtk classes. Among these `vtkRenderWindow` provides connection between Win32 and vtk by specifying the window area where graphics drawing takes place. `vtkRender` class is combined to `vtkRenderWindow` class instance at run-time. This class has methods that execute rendering or calculation & rastering of pixels.

Another fundamental and useful class to be used is `vtkRenderWindowInteractor` class. This class provides some kind of interaction between user and the rendering pipeline. It accepts simple commands from the user and transmits these to rendering classes. These commands are zooming in/out and padding. Also, interactors can take input from keyboard if the focus is on the drawing window.

Our system uses Actors to input data to rendering pipeline. Actors serve to group rendering attributes such as surface properties (e.g. ambient, diffuse, specular color), representation (e.g. , surface or wireframe), texture maps, and geometric definitions.

After defining the actor, we need to define the data structure of vtk library since, the data is needed to be handled by these. `vtkPoints` class represents 3D points. The data model for `vtkPoints` is an array of vx-vy-vz triplets accessible by (point or cell) id. `vtkCellArray` class is used to represent cell connectivity. The cell array structure is a raw integer list of the form: (n,id1,id2,...,idn, n,id1,id2,...,idn, ...) where n is the number of points in the cell, and id is a zero-offset index into an associated point list. This class is used in combination with `vtkPoints` class in forming `vtkPolyData` class. `vtkPolyData` is a data object that is a concrete implementation of `vtkDataSet`. `vtkPolyData` represents a geometric structure consisting of vertices, lines, polygons, and triangle strips. Point attribute values (e.g., scalars, vectors, etc.) also are represented. In forming the stream tubes, we have used `vtkPolyData` class for storing the coordinate information.

`vtkImageData` class is used as the storage of the image pixel values. This class can get input from a 3D array of dimension 3. This image, then, can be displayed by transferring this data to `vtkImageActor` class which is a child class `vtkActor`. Finally, the data in `vtkImageActor` is fed into the current rendering class.

For the main dialog of our application, there are three separate image views are placed at the top row as seen in figure 1. These views will display axial, sagittal and coronal 2D images of the brain. Slide-bars, under them will provide navigation through the frames. Each of these frames are a separate instance of the 'pencere' class that forms an individual rendering pipeline separate from each other.

The bottom left view will represent a 3D view of the combination of the previous 3 views. Three planes perpendicular to each other will be drawn and the corresponding slices images will be texture mapped on to these planes using bilinear interpolation. Thus, the rendering system uses hardware acceleration to model the whole system in three dimensions.

There's another crucial topic; setting seeds and observing them in the images. vtkPolyData class will be used to save picked regions in the image. This data will be separate from image data, so seed points can be easily added and removed. Also, using the indices of the vtkPolyData, sequential and random access will be easily done to specific seed points.



Figure 3

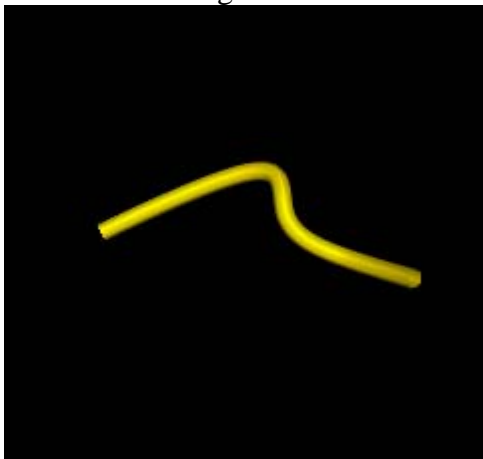


Figure 4

To create the stream tube shown left in figure 3, first the 4 voxel point coordinates are entered manually in a vtkPoint class instance. Then a vtkCellArray is created to indicate the access sequence which is 1,2,3,4. Afterwards, both vtkPoint and vtkCellArray instance are used to form a vtkPolyData class instance. Next, this class is fed into vtkTubeFilter class, which thickens the lines to be drawn between specified points. Finally the output of the vtkFilterTube is fed into an actor class and rendered.

A smooth and better stream tube can be created by interpolating the initial input point set to a higher resolution. In figure 4, cardinal spline interpolation is applied to the initial 4 point data set to acquire 80 point data set. Afterward, the same procedure is continued as explained above but much more data points.

The DT-MRI data provided for the project by Dr. Burak Acar, is in binary file format. The file structure was also provided by Dr. Burak Acar. Accordingly, the necessary file input/output is achieved by standard C-run time IO Library. The significant byte ordering of

the float data types in the data files were mirrored versions that are used in PCs. So to get the correct values, we have to mirror the 4 bytes of every float. The program given in listing 1, shows the C implementation of the described procedure.

```
#include <iostream>
#include <fstream>
#include <conio.h>
#include <stdio.h>
#define ROW 128
#define COL 128
using namespace std;
char buffer[5];
main()
{
    FILE *fptr;
    buffer[4]=NULL;
    float** max_eig=(float**)malloc(ROW*sizeof(float*));
    for (int i=0;i<ROW;i++){
        max_eig[i]=(float*)malloc(COL*sizeof(float));
        for (int j=0;j<COL;j++){
            max_eig[i][j]=0;
        }
    }
    fptr=fopen ("C:\\Documents and
Settings\\admin\\Desktop\\ee547pr\\mri\\Tensor.float.001", "rb");
    float her;
    char *adr;
    adr=(char*)&her;
    char temp;
    if (fptr==NULL)
    {
        cout<<"file cannot be opened..."<<endl;
    }
    else{
        fseek(fptr,ROW*COL*sizeof(float),0);

        for (int i=0;i<ROW;i++){
            for(int j=0;j<COL;j++){
                fread(buffer,sizeof(char),4,fptr);
                // Flipping bytes of float
                *adr=buffer[3];
                *(adr+1)=buffer[2];
                *(adr+2)=buffer[1];
                *(adr+3)=buffer[0];
                if(her!=0)
                    cout<<her*(1E-6)<<" ";
            }
        }
        getch();
    }
}
```

Listing 1

## 4-References

[1] O. Ciccarelli,<sup>a</sup> G.J.M. Parker,<sup>b</sup> A.T. Toosy,<sup>a</sup> C.A.M. Wheeler-Kingshott,<sup>a</sup> G.J. Barker, <sup>a</sup> P.A. Boulby,<sup>c</sup> D.H. Miller,<sup>a</sup> and A.J. Thompson<sup>a</sup>, “From diffusion tractography to quantitative white matter tract measures: a reproducibility study”

[2] Peter J Basser, Sinisa Pajevic, Carlo Pierpaoli and Akram Aldroubi, “Fiber Tract Following in the Human Brain Using DT-MRI Data”, IEICE Trans . Inf. & Syst. Vol E85-D January 2002

[3] C.R. Tench, P.S. Morgan, M. Wilson, and L.D. Blumhardt, “White Matter Mapping Using Diffusion Tensor MRI”, Magnetic Resonance in Medicine 47:967–972 (2002)

[4] G.J.M. Parker, “Tracing Fiber Tracts Using Fast Marching”

[5] S. Zhang , C . Demiralp , M. DaSilva , D. Keefe , D. Laidlaw , B. D. Greenberg , P.J. Basser , C. Pierpaoli , E.A. Chiocca , T. S. Deisboeck , “Toward Application of Virtual Reality to Visualization of DT-MRI Volumes”