Final Report

MAB681 Advanced Visualisation and Data Analysis

Visualisation of the Brain

Team Members:

Tim Fan	05122414
Christopher Savini	05741467

Supervisor: Joe Young

Table of Contents

1. Summary

In the past, imaging the brain has been restricted to post-mortem studies. MRI scans have made it possible for researchers to study the structure of the brain in a living patient.

Our objective is to take the data from a diffusion MRI scan and produce some visualisations using tensor information to highlight neural pathways in the brain. These visualisations will be useful for learning about the structure of the brain, specifically the Tractographic reconstruction of neural connections.

Visualizing neural pathways can be used to measure changes in white matter, for example during aging. One of the most important initial applications in the visualising of neural pathways is the localization of tumors in relation to the white matter tracts (such as path deflections around a tumor). It is also possible to use these types of visualizations in the surgical planning for some types of brain tumors; surgery is aided by knowing the proximity and relative position of the corticospinal tract and a tumor.

2. Project Description

1. Data Format

The data is a 3D tensor field. Each voxel has a 3 x 3 matrix associated with it. This represents the diffusion of water at each point.

The diffusion matrix is symmetric so only 6 of these numbers are actually

$$\mathbf{D} = \begin{pmatrix} D_{xx} & D_{xy} & D_{xz} \\ D_{yx} & D_{yy} & D_{yz} \\ D_{zx} & D_{zy} & D_{zz} \end{pmatrix}$$

required to be stored in the file. The data file also stores a 'confidence' value with each tensor. This value is used to mark tensors which are important to draw. A value of 1.0 means that it is important to draw this tensor, while a value of 0.0 means that this tensor is not very important. (Confidence also makes computation faster because it reduces the number of glyphs we need to draw). Confidence values can be between 0.0 and 1.0 . All the numbers are stored as 32 bit float. The data comes with a header file (.nhdr) which specifies the dimensions and axis order of the grid.

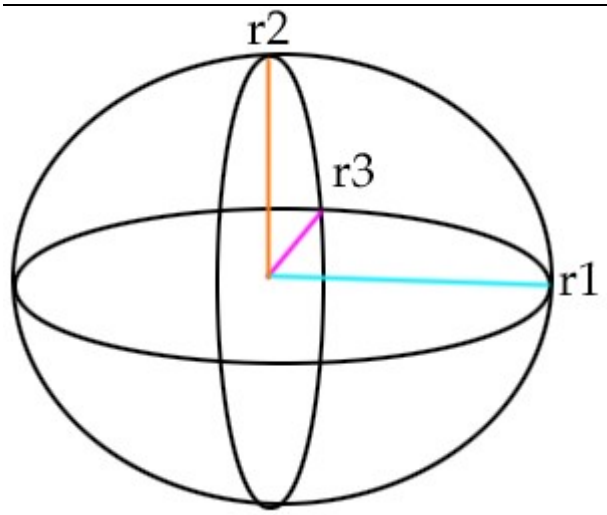
From this data which we load from the file, additional information is calculated. Firstly, each diffusion matrix has three orthogonal eigenvectors and three corresponding eigenvalues. The diffusion matricies also have a location in 3D space which is calculated from the location in the grid. The eigenvectors are used to calculate rotation matricies, the eigenvalues are used to calculate scaling matricies which stretch the glyph and the locations are used to calculate translation matricies. So there is a fair bit of data which is not explicitly stored in the file.

The data came from this website http://www.sci.utah.edu/~gk/DTI-data/ .

2. Visualisation Techniques

Glyphs

The diffusion of water in three dimensional space represented by the tensors can be represented by three vectors. Elipsoid Tensor Glyphs are suitable for showing this in the visualisation. This is because the length, width and height of the elipsoid can be used to represent the diffusion vectors.



Fibre Tracking

Using the diffusion information, we can track fibres within the brain. We can do this by converting the tensor field into a vector field (this can be done by finding an average of the eigenvectors), and putting a particle into the field to see where is goes.

3. *Project Tools*

3.1. 3D Slicer

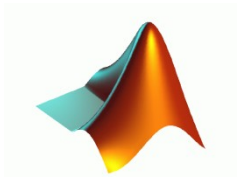
This is a tool for visualising medical data, converting to different data formats and other usefull jobs. (http://www.slicer.org/)

3.2. XNA/DirectX

XNA is a set of tools released by Microsoft which provide graphics capabilities for .Net. XNA can be used with the C# language. It is intended for games developers however its ability to do realtime rendering and provide access to the GPU makes it useful for scientific visualisation as well.

(http://msdn.microsoft.com/en-au/xna/default.aspx)

3.3. MATLAB



While none of our visualisations have been done in MATLAB, matlab has been useful for learning about the structure of the data.

3.4. Iridium

Iriduim is an open source maths functions library for .net languages such as C#. We have used this for performing the eigenvector and eigenvalue computations. http://www.mathdotnet.com/Iridium.aspx

3.5. Doxygen

Doygen is a tool for generating documentation. http://www.stack.nl/~dimitri/doxygen/

3.6. Visual Studio C#

Programming IDE and language by Microsoft http://www.microsoft.com/express/vcsharp/

3.7. OpenDX

We've used this program to produce a few visualisations as well. http://www.opendx.org/

3.8. TortoiseSVN

Tortoise is a version control system used for project management. http://tortoisesvn.tigris.org/

4. Results and Outputs

4.1. Data Massaging

We used some MATLAB scripts to convert the data into OpenDX format. See the last section of the appendix for the code.

By using BrainViewer, we can exact all the positions where the confidence is equal to zero, the rotation matricies, eignvalues, eignvectors and fibre paths into sepreate txt files. Matlab is able to read all these txt files into data and convert into a .dx file along with the connections and colors.

The structure of .dx file contains the positions of each glyph (after the rotation done by Matlab using the rotation Matrix), the connections of each glyph (using quads to connect each position of indivdual ellipsiod) and the colors (RBG values using the eignvalues). Each ellipsoid is constructed from scratch using the ellipsoid equation. Higher detailed ellipsoid meshes give better looking ellipsoid models however the file size becomes much larger. This can become too large for our computer to draw if you are not careful. We have also noticed that text files seem to have a line number limit which can also be restricting.

4.1. Algorithm Descriptions

The fibre tracking algorithm works by randomly placing a number of particles in the brain and then moving them based on the diffusion. On each timestep we find the closest diffusion tensor and calculate an average of its eigenvectors scaled by the eigenvalues.

(1/3) *(+-v1 * l1 +- v2 * l2 +- v3 * l3)

We have eight different combinations to try here since v1, v2 and v3 can be positive or negative since both positive and negative solutions are valid eigenvectors. (The physical interpretaion of this is that diffusion is always away from the said point in space).

Of these eight posibilities we choose the direction which takes the particle furthurest from its previous position. This prevents particles from getting trapped.

4.2. Things we learnt

An important thing that we learnt from this project is how to write efficient code.

Don't allocate memory inside a loop if you can avoid it. It is much faster to prealocate the memory beforehand and then overwrite those data locations. This prevents the computer from having to repeatedly alocate and delete memory. Also we had to be careful how we stored data. Not only was the datafile itself large, each diffusion tensor has 3 eigenvalues, 3 eigenvectors (each a 3 -vector), a 3x3 rotation matrix, a 3x3 translation matrix, and a 3x3 stretching matrix. So the we have to be carefull how we handle all all this.

We also used the confidence values stored in the file to help decide which values to skip. While there were a lot of data values with high confidence, they still only made up a small portion of the total data values. Most of the data values were noise. Even with this filtering it was still neccesary to skip some data values due to the large size of the data.

The position of each data point was not explicitly stored in the data file. Each tensor is stored in order of position in the grid. So if your grid size is 256x256x38, a data value stored at location [x y z] = [120 34 12] this data value will be the 'Indexth' item in the list where Index = 120 + 256*34 + 256*256*12

In general this equation is index = x + X*y + X * Y *z

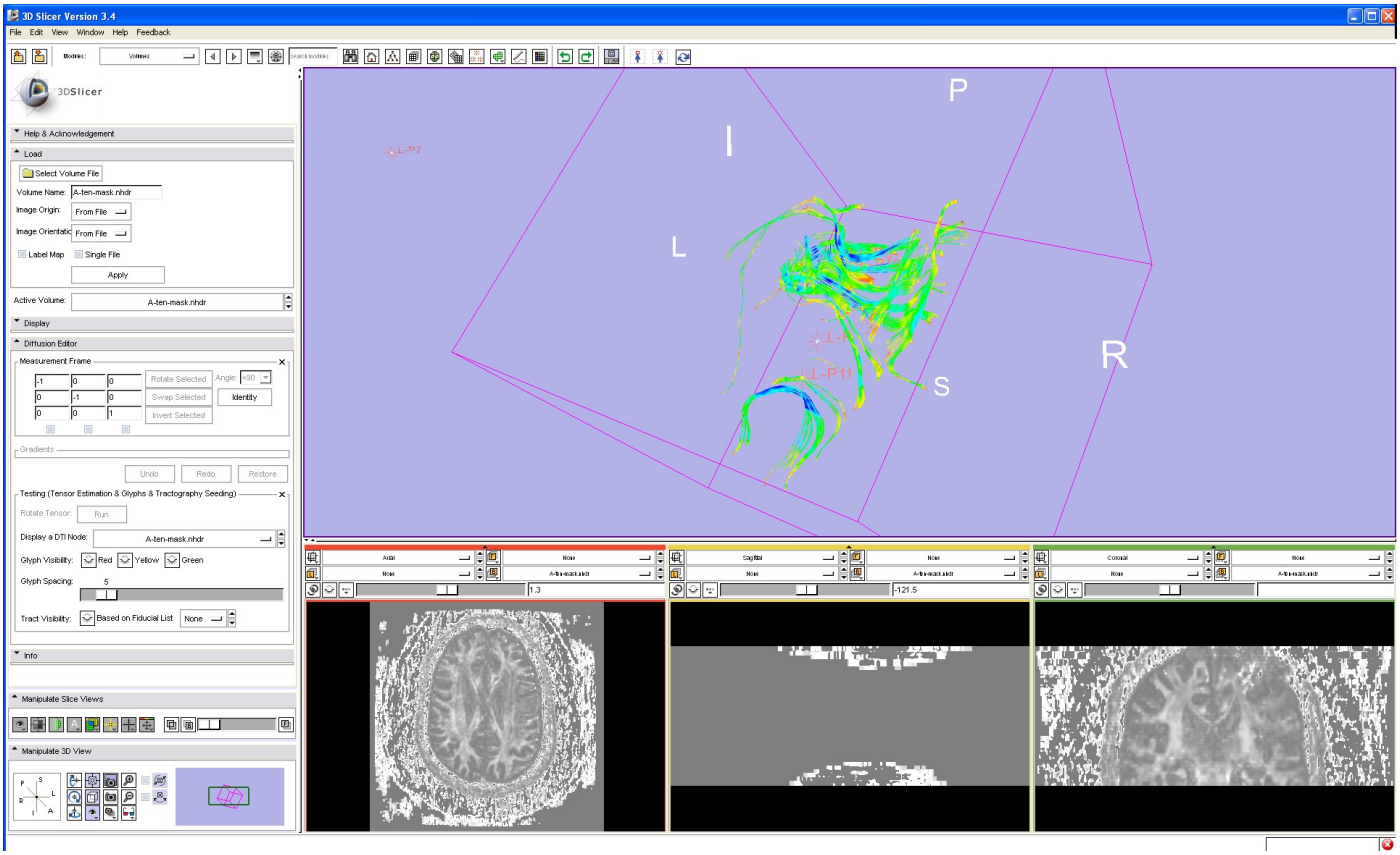
However we normaly want to preform this calculation in reverse. Ie: we have the index, but we want to calculate [x y z] positions. To do this efficiently we use the equation:

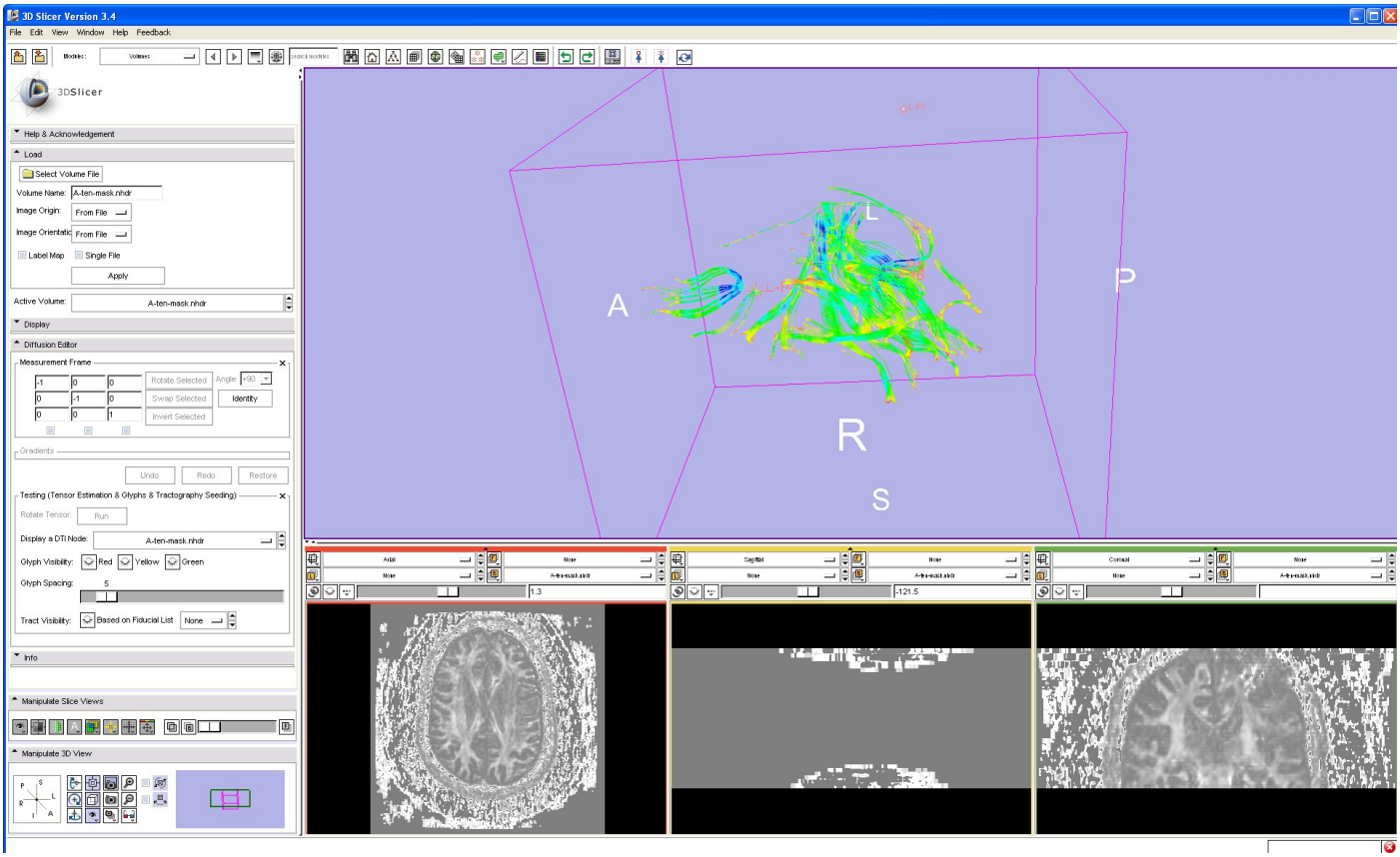
3.

1.1. Visualizations

1.1.1. 3D Slicer

4. Fibre Tracking in 3D Slicer




3D Slicer Version 3.4
File Edit View Window Help Feedback
3DSlicer
Help & Acknowledgement
Load
Select Volume File
Volume Name: A-ten-mask.nhdr
Image Origin: From File
Image Orientatic: From File
Label Map Single File
Apply
Active Volume: A-ten-mask.nhdr
Display
Diffusion Editor
Measurement Frame
Rotate Selected
Swap Selected
Invert Selected
Identity
Gradients
Undo
Redo
Restore
Testing (Tensor Estimation & Glyphs & Tractography Seeding)
Rotate Tensor:
Run
Display a DTI Node: A-ten-mask.nhdr
Glyph Visibility: Red Yellow Green
Glyph Spacing: 5
Tract Visibility: Based on Fiducial List None
Info
Manipulate Slice Views
Manipulate 3D View
A
P
R
S
L

5.

1.1.1. Interactive Visualization in XNA

The interactive brain viewer can be found in the directory

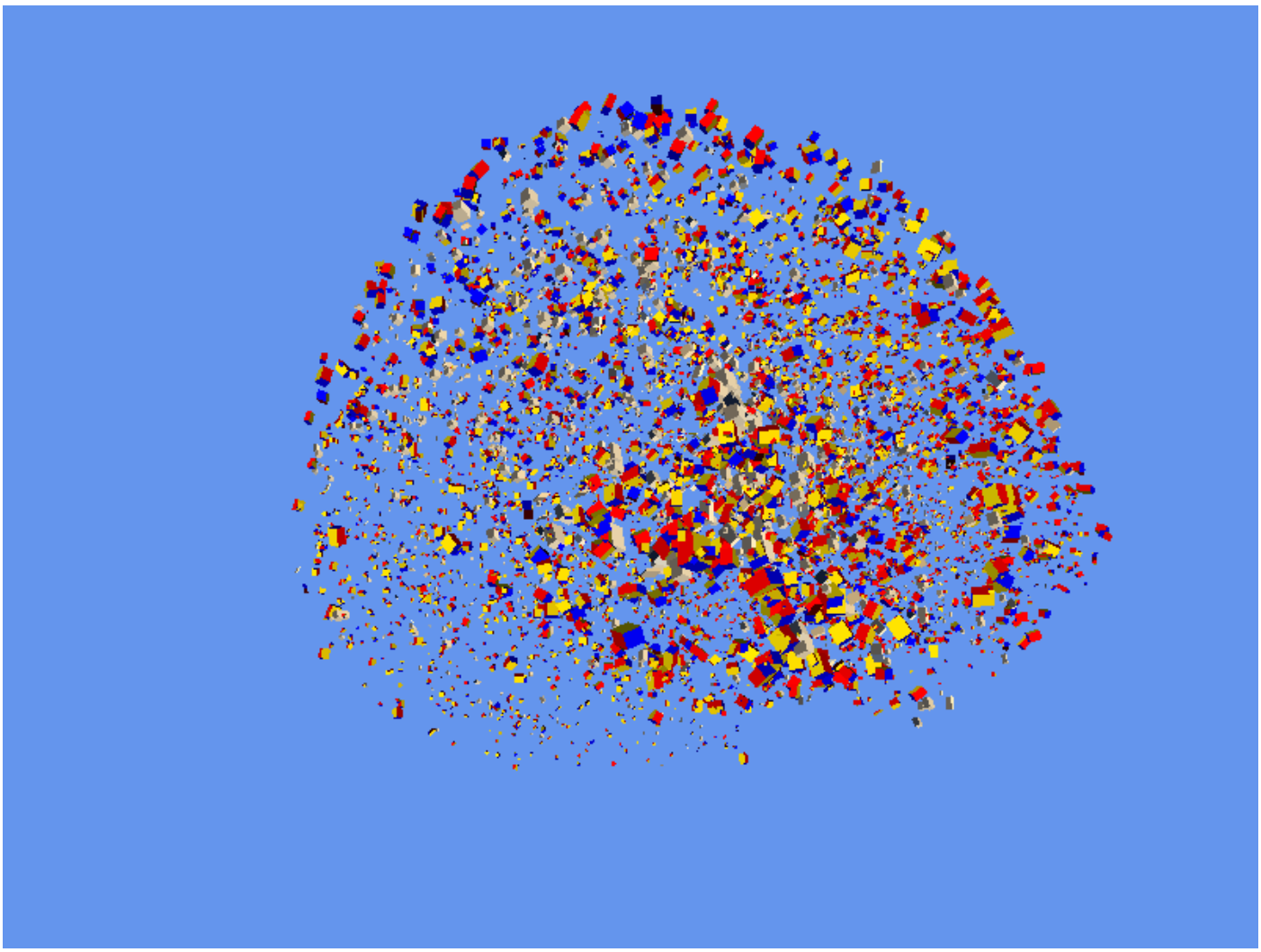
Brain Viewer\XNA\BrainViewer\BrainViewer\bin\x86\Release\BrainViewer.exe

The executable will run on a windows machine.

If you wish to compile the project yourself you will need to install XNA and have the latest version of visual studio. (The visual studio project file is BrainViewer\BrainViewer.sln)

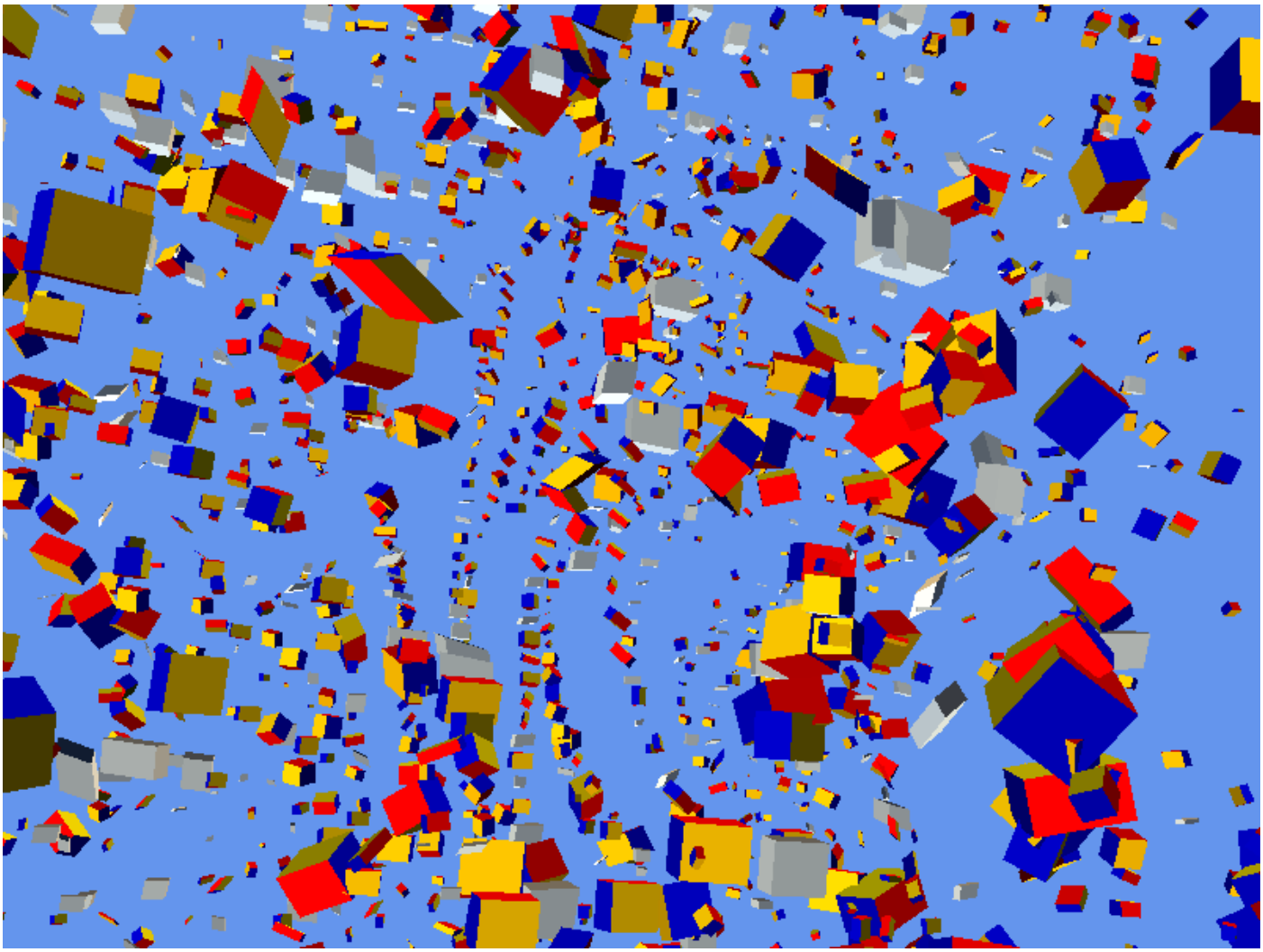
The controls for the interactive visualization are :

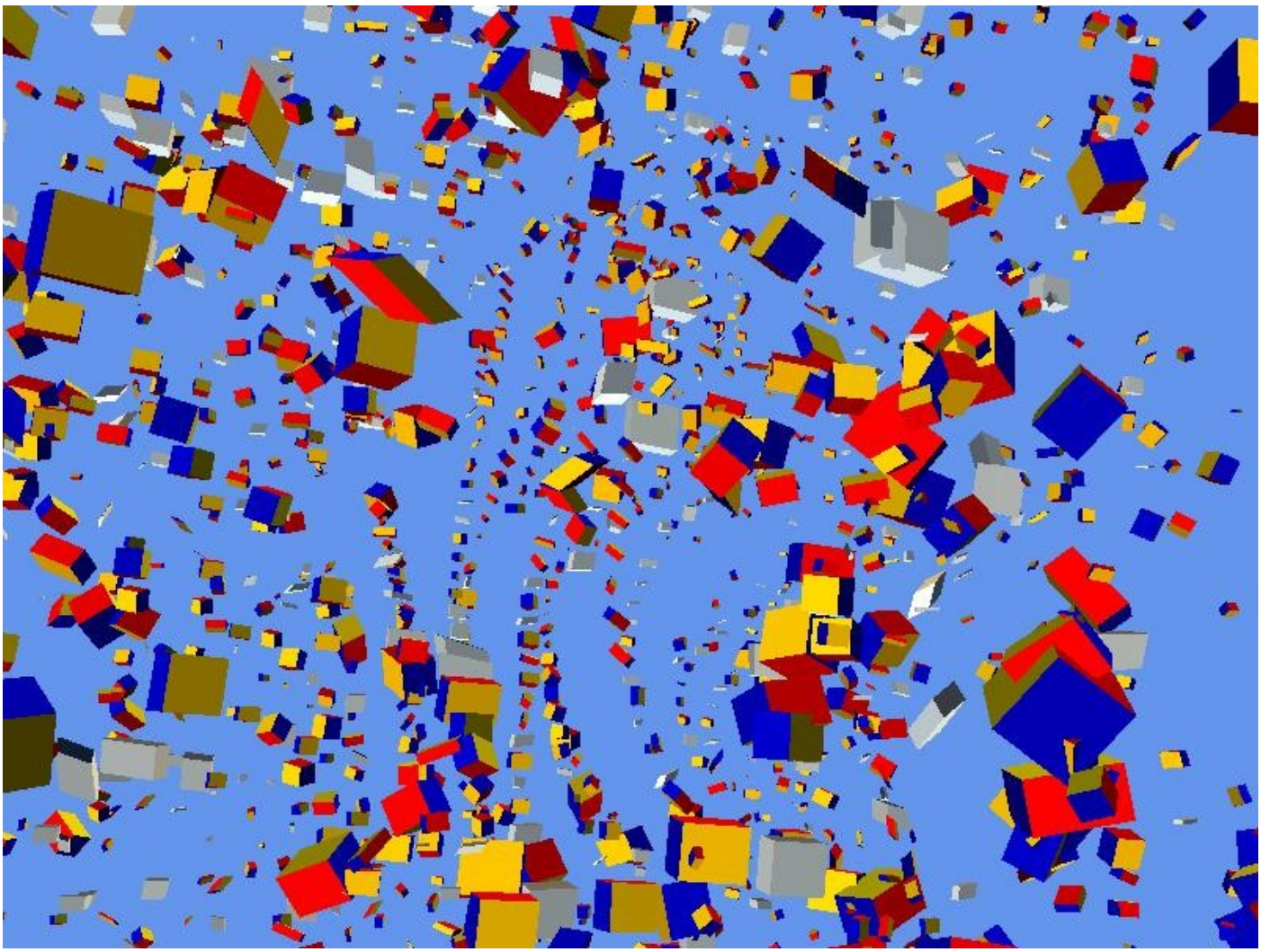
Left/Right shift	Rotate
W	Move Forward
A	Strafe Left
S	Move Backward
D	Strafe Right
Up Arrow	Look Up
Down Arrow	Look Down
Right Arrow	Rotate Right
Left Arrow	Rotate Left
Page Up	Hover Up
Page Down	Hover Down
Esc	Quit

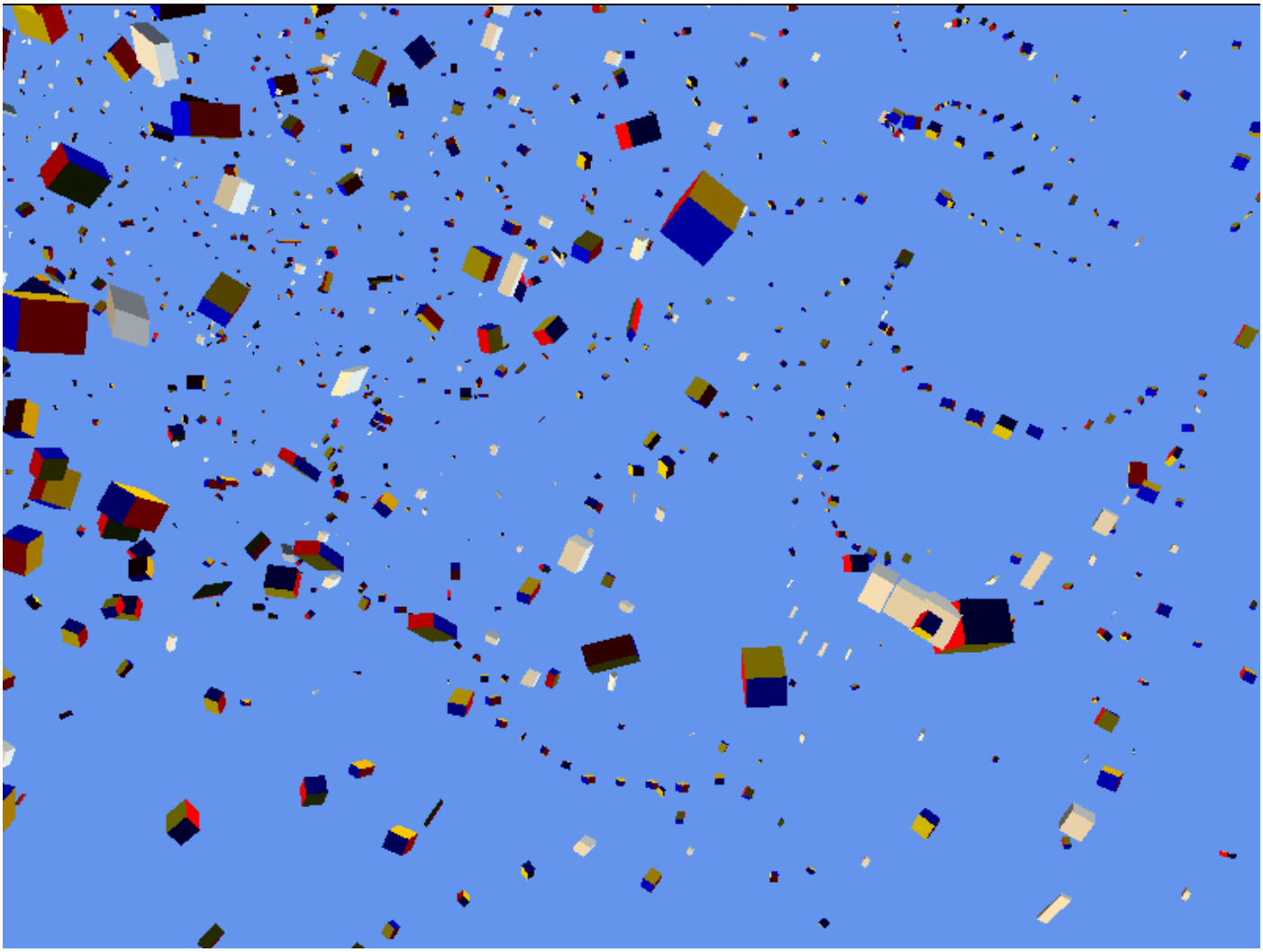


This image shows that there is more diffusion in the centre of the brain.

Even without fibre tracking, the fibres are beginning to show up as shown by the following images.

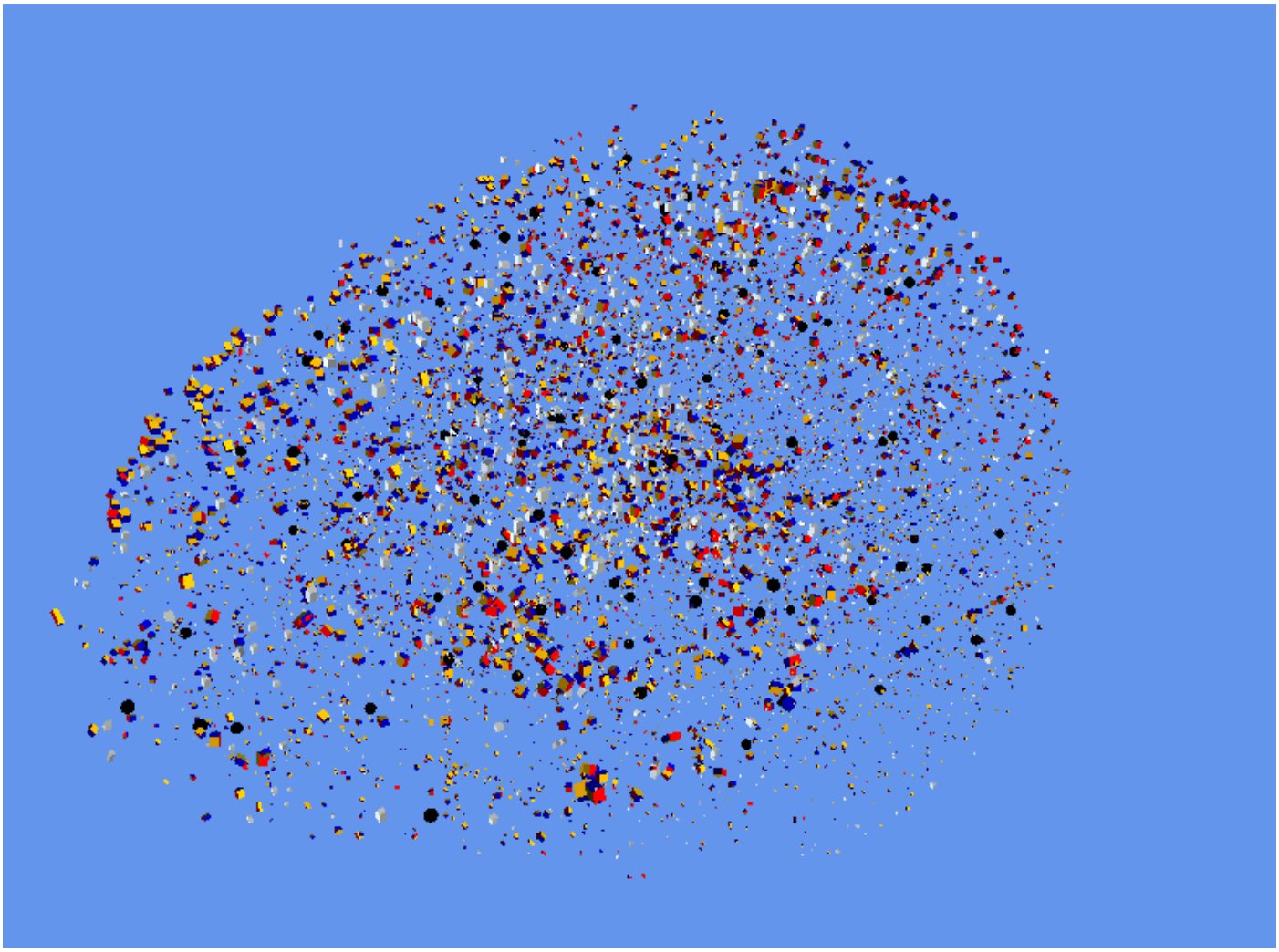




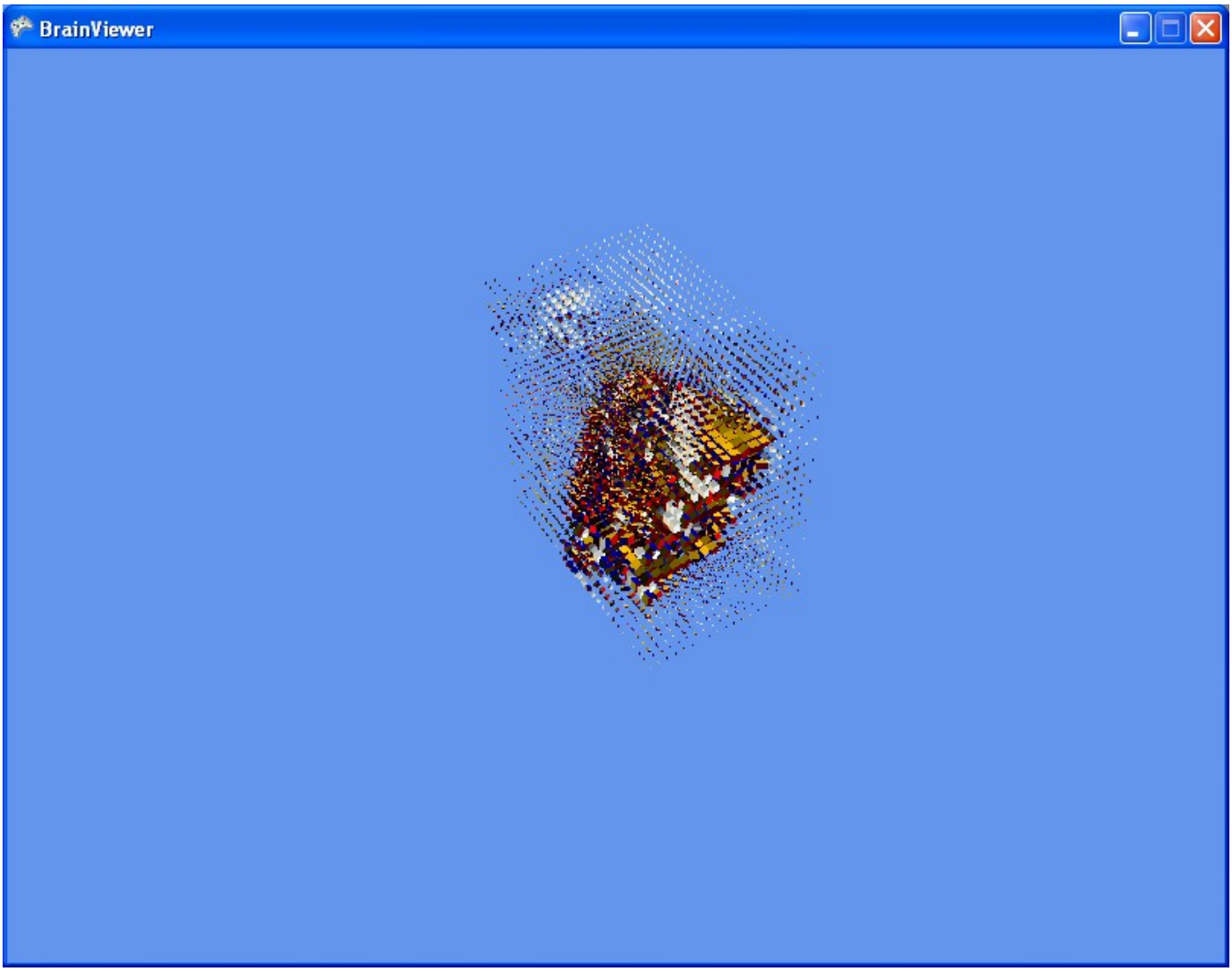


The Brain Viewer was also used to generate our fibre tracks, although they are better visualized in OpenDX

The particles of the fibre tracking algorithm are shown as black balls stepping through time.



Due to the large size of the data, we have cut out a small region so that we do not have to skip values in the file.(We are still only using values with confidence of 1.0) This allows us to get a better representation of the diffusion in the region for the purposes of fibre tracking.

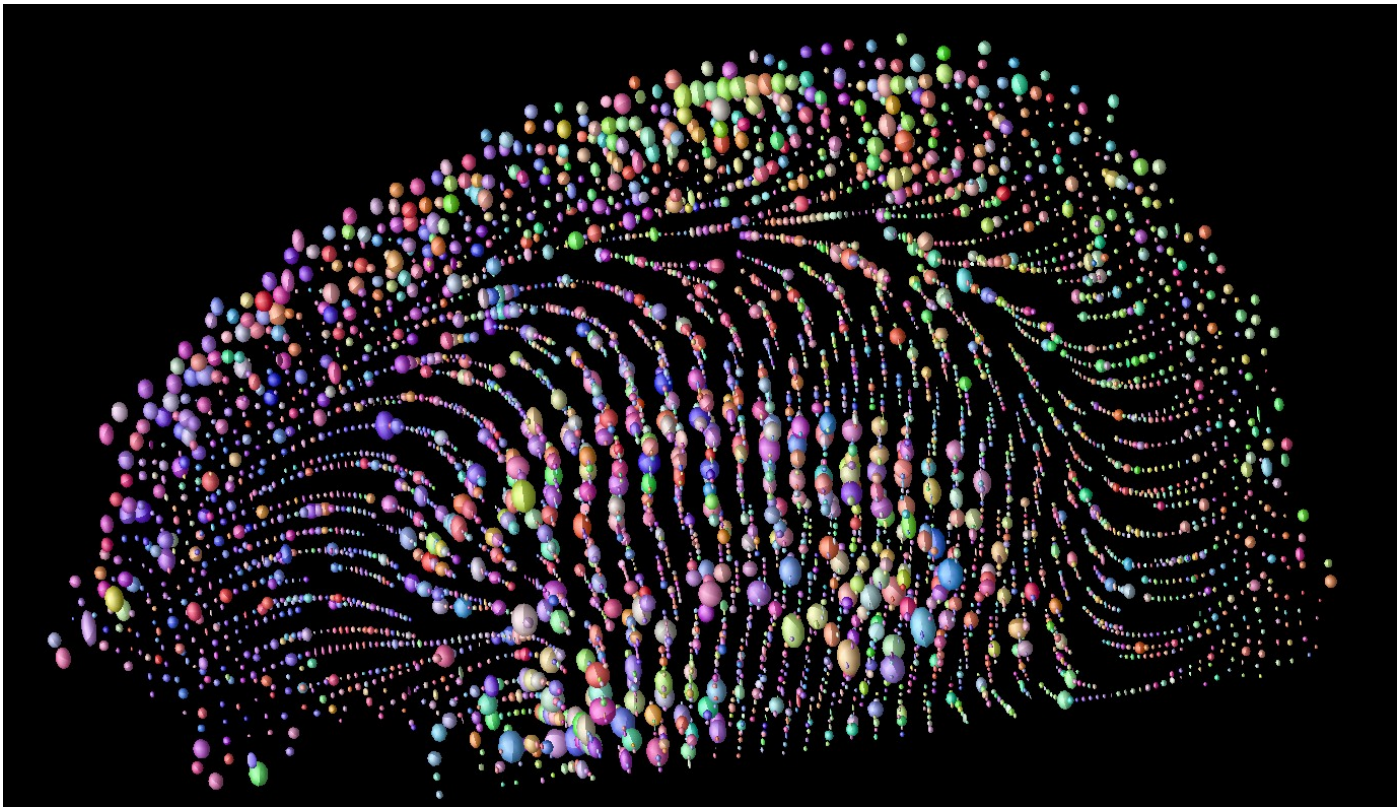


1.1.2. OpenDX

6.

7. The fibres can be seen very clearly even though we have not done any fibre tracking. (Also we have not done any colour mapping yet.)

8.



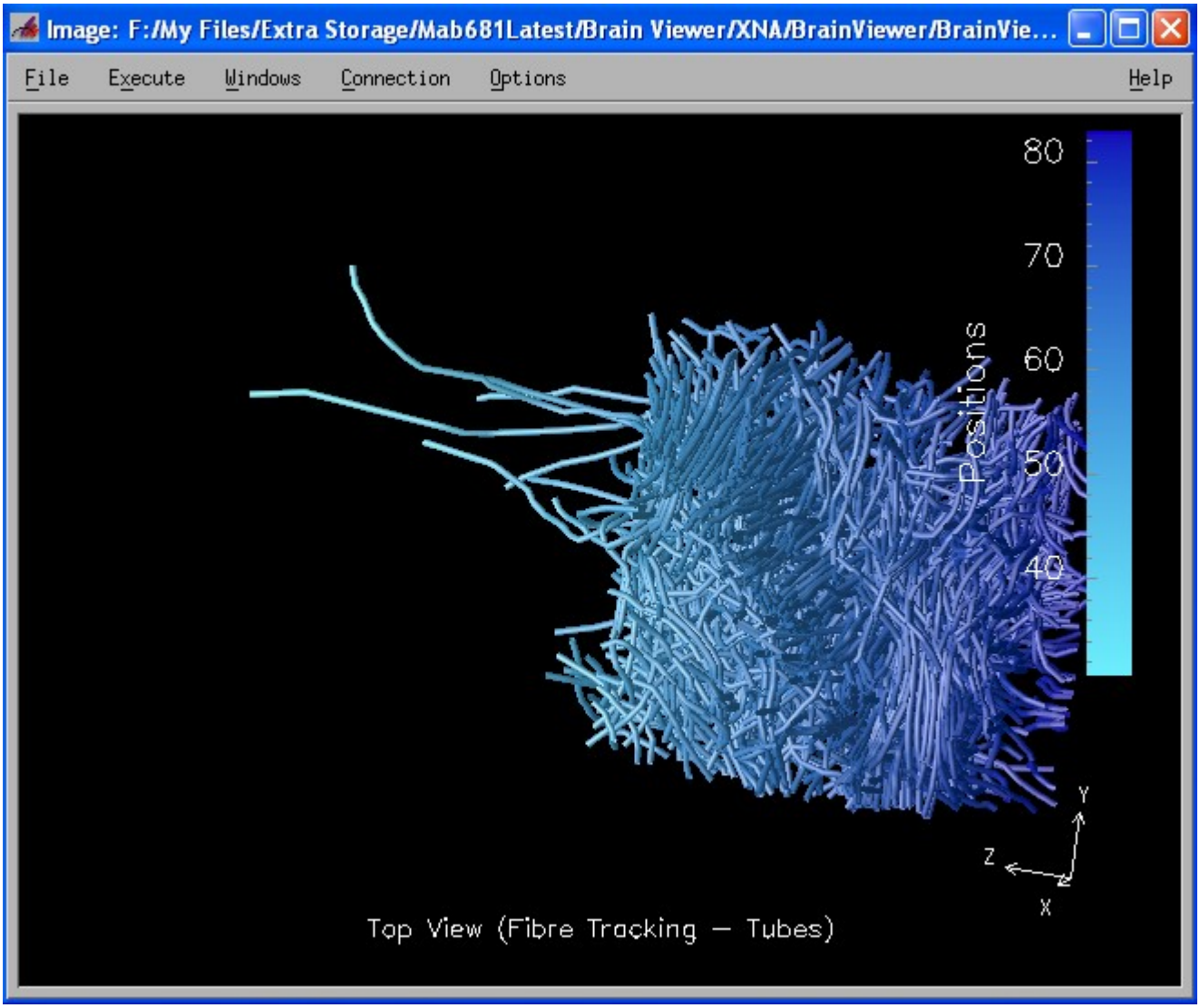
9.

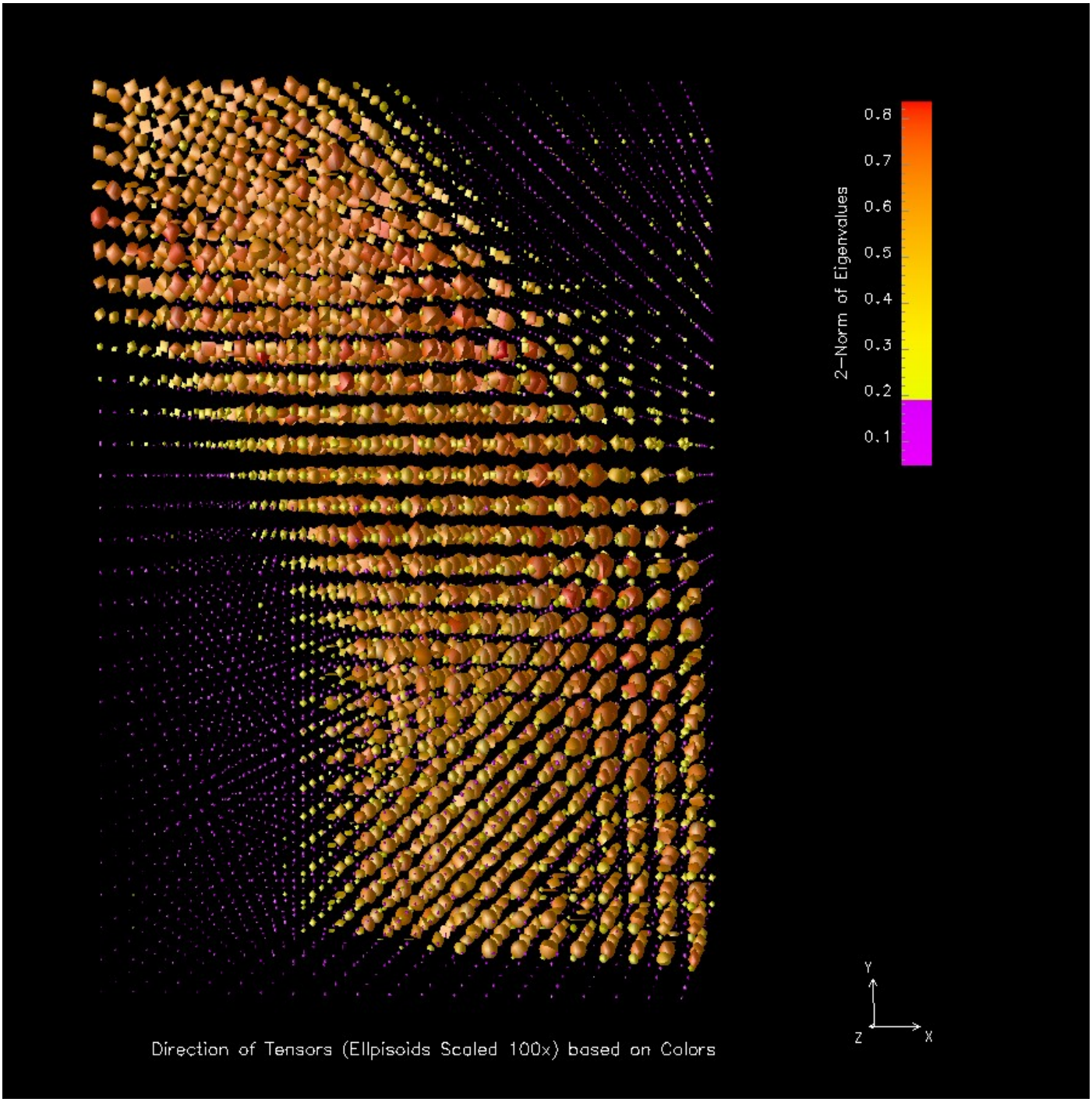
1.1.1. Fibre tracking and movies

10.

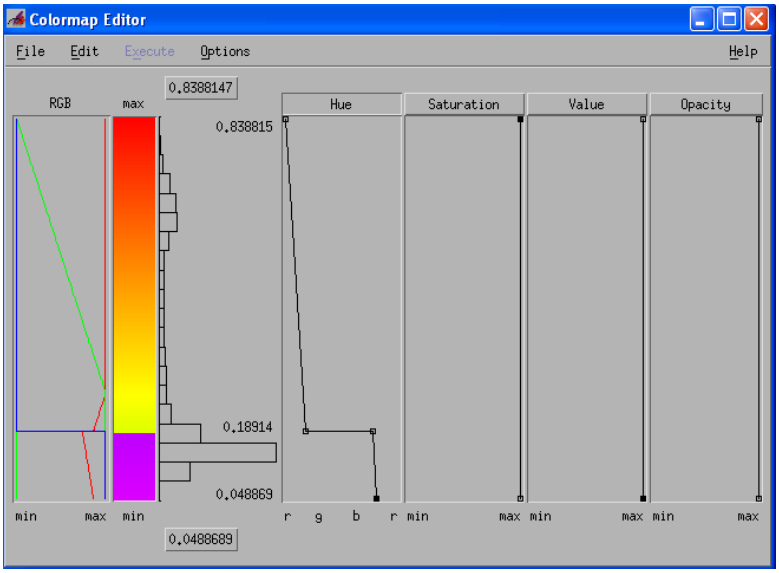
11. Fibre Tracking on a small region

12.



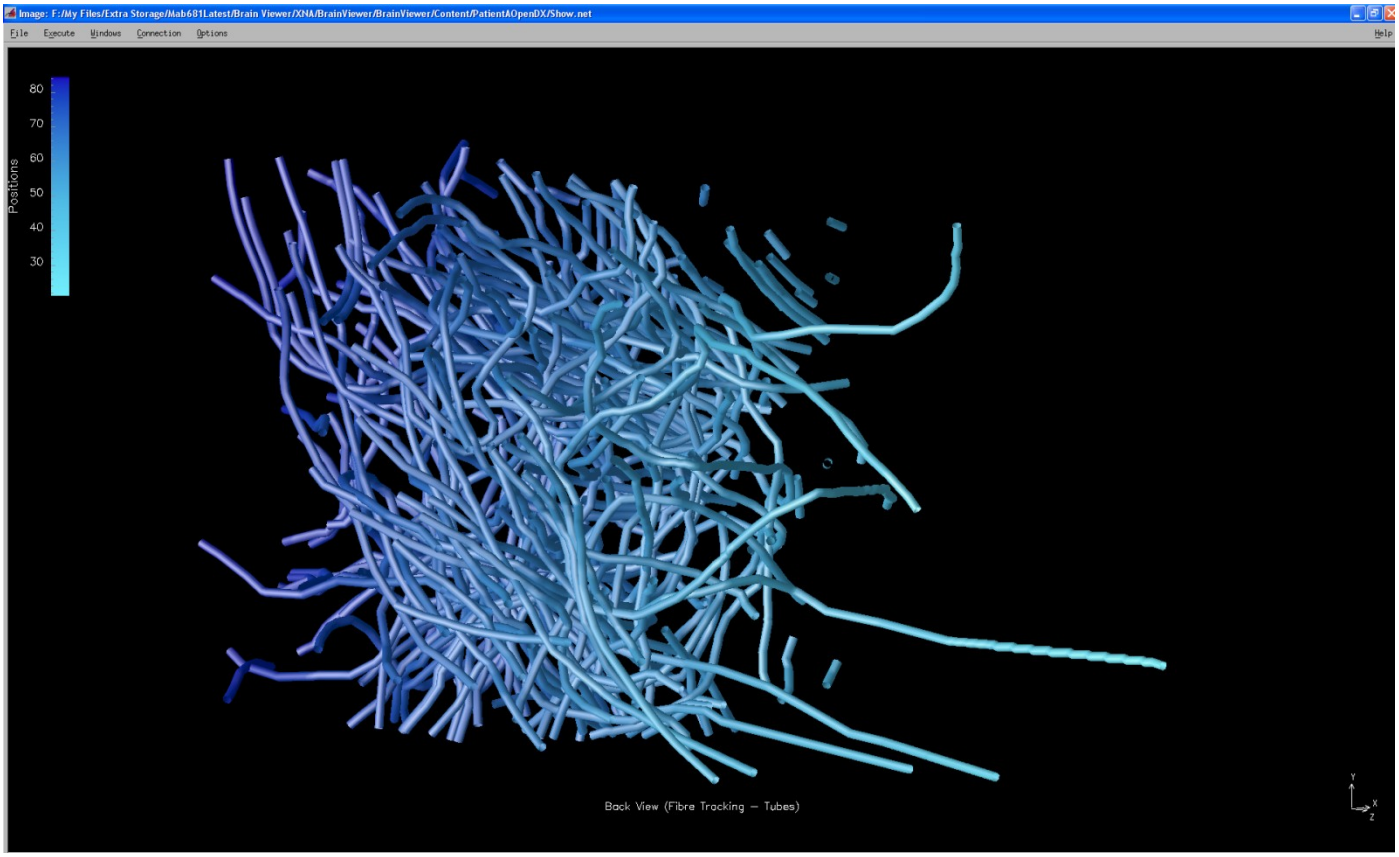

2-Norm of Eigenvalues
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
Y
Z
X
Direction of Tensors (Ellpisoids Scaled 100x) based on Colors

The tensors have been coloured by the 2-norm of the eigenvalues. This is to distinguish areas of large and small diffusion, regarless of the direction.

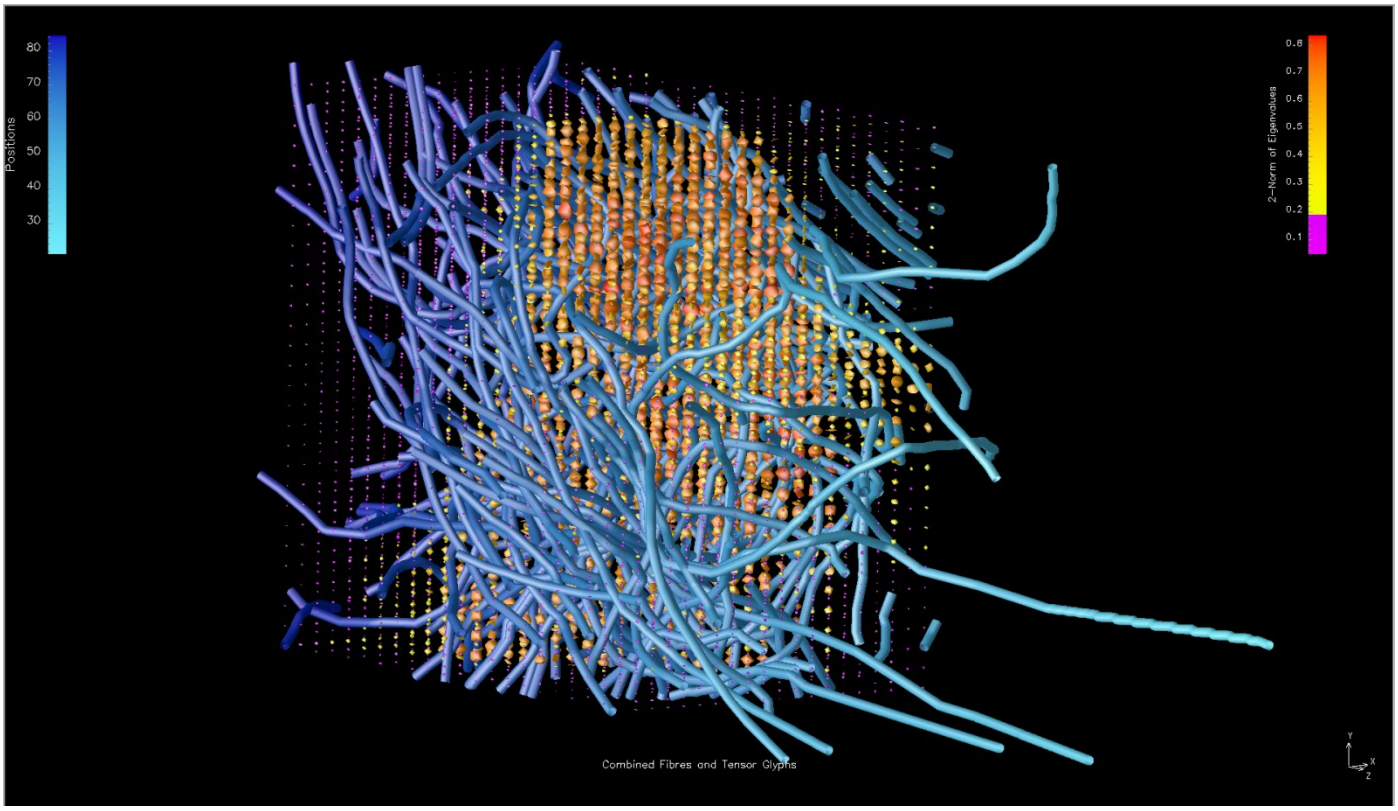


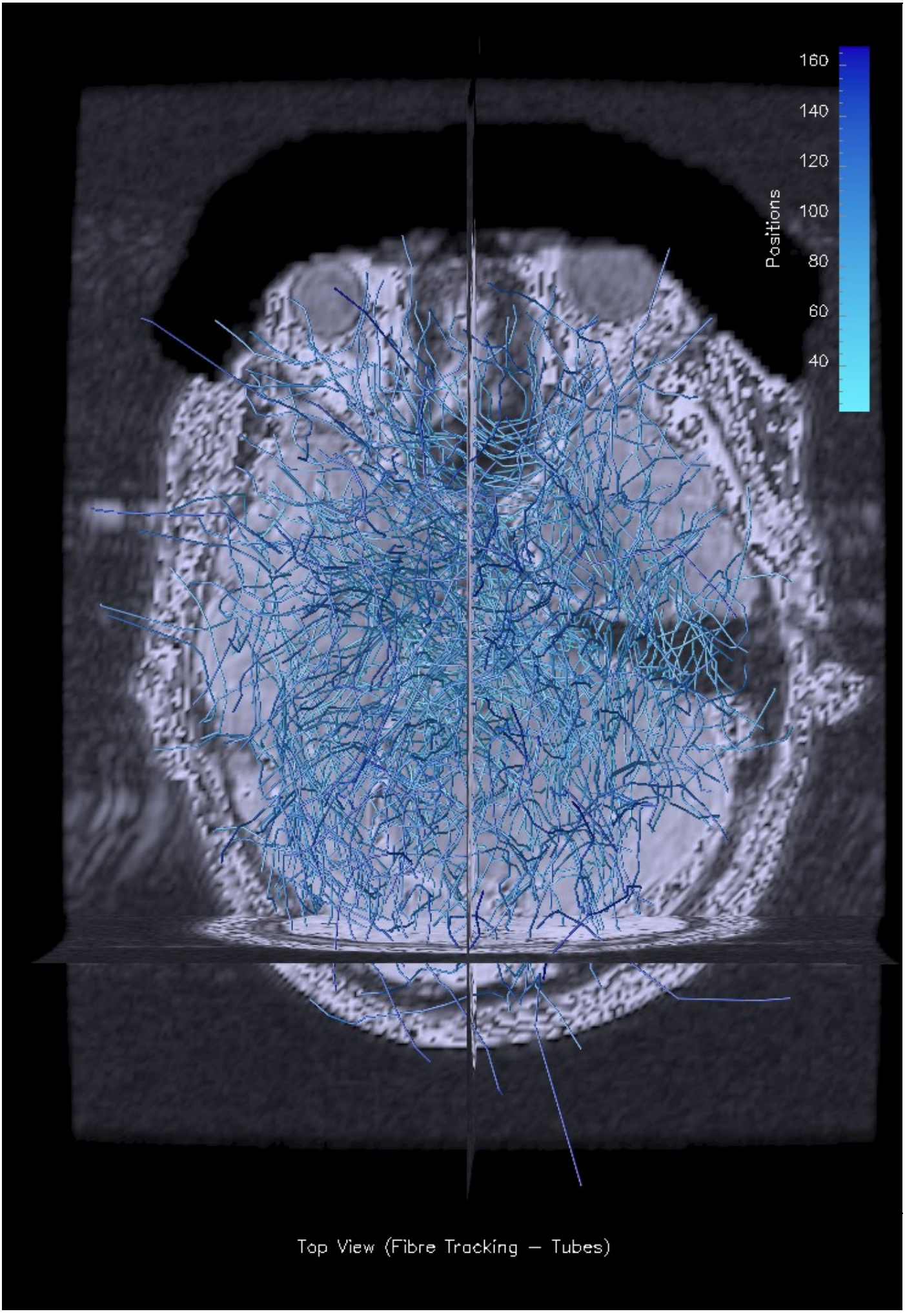
We have used the purple in this colour map to distinguish the region of very small diffusion tensors from the rest of the data.

Fibre Tracking Coloured by the position



Combined fibres and tensor glyphs





Top View (Fibre Tracking – Tubes)

13.

1.1. Website and documentation

14.

15.

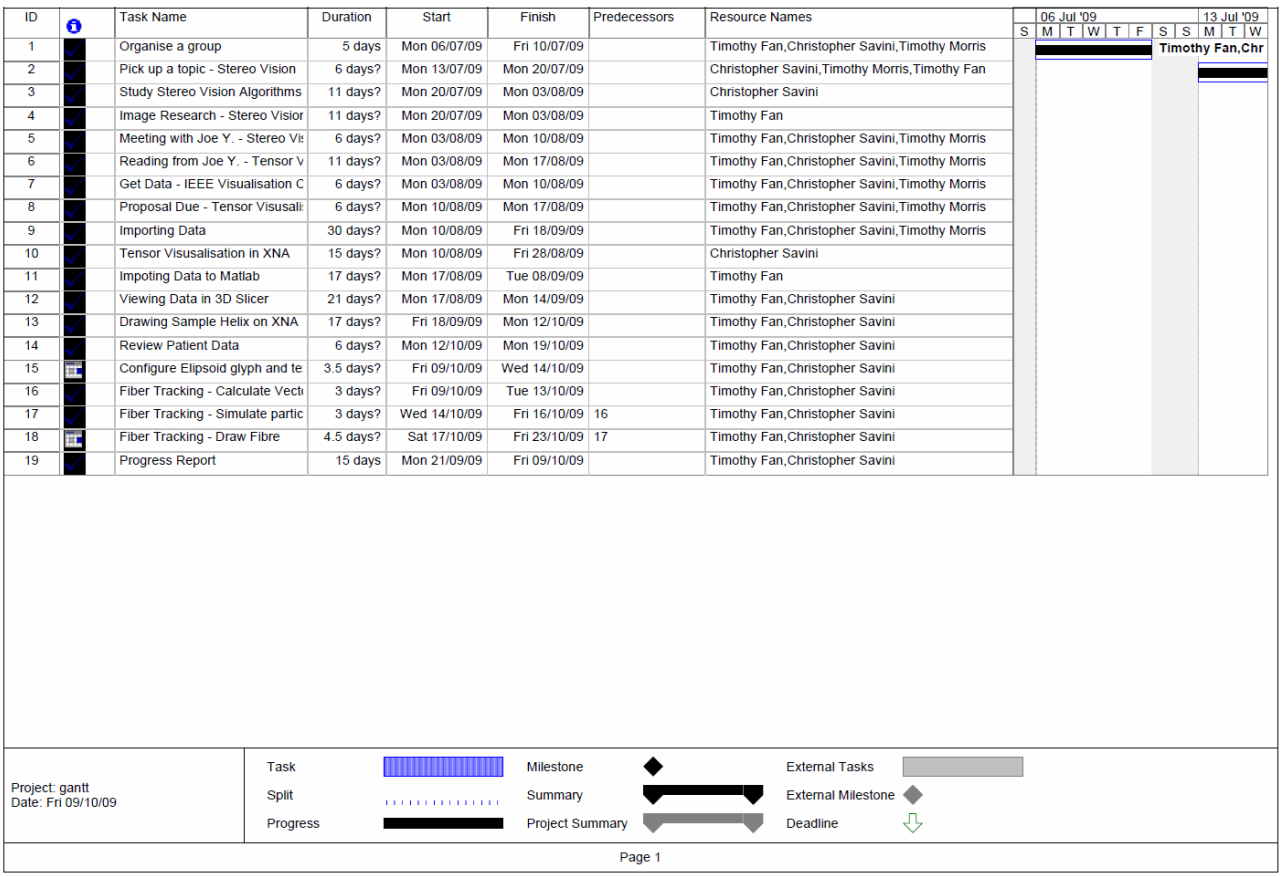
16. Effectiveness of Visualizations

17. Project Timeline

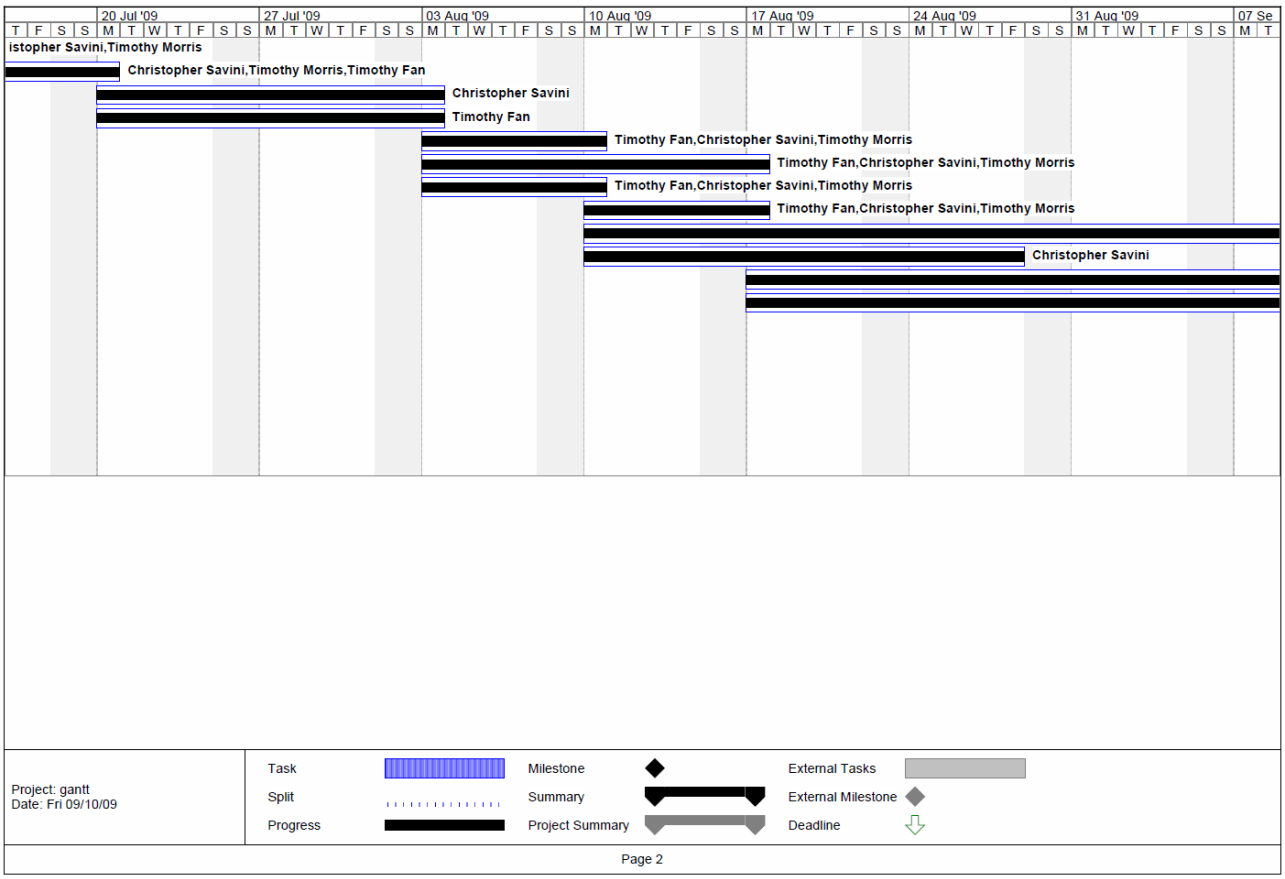
1. Gantt Chart

18.

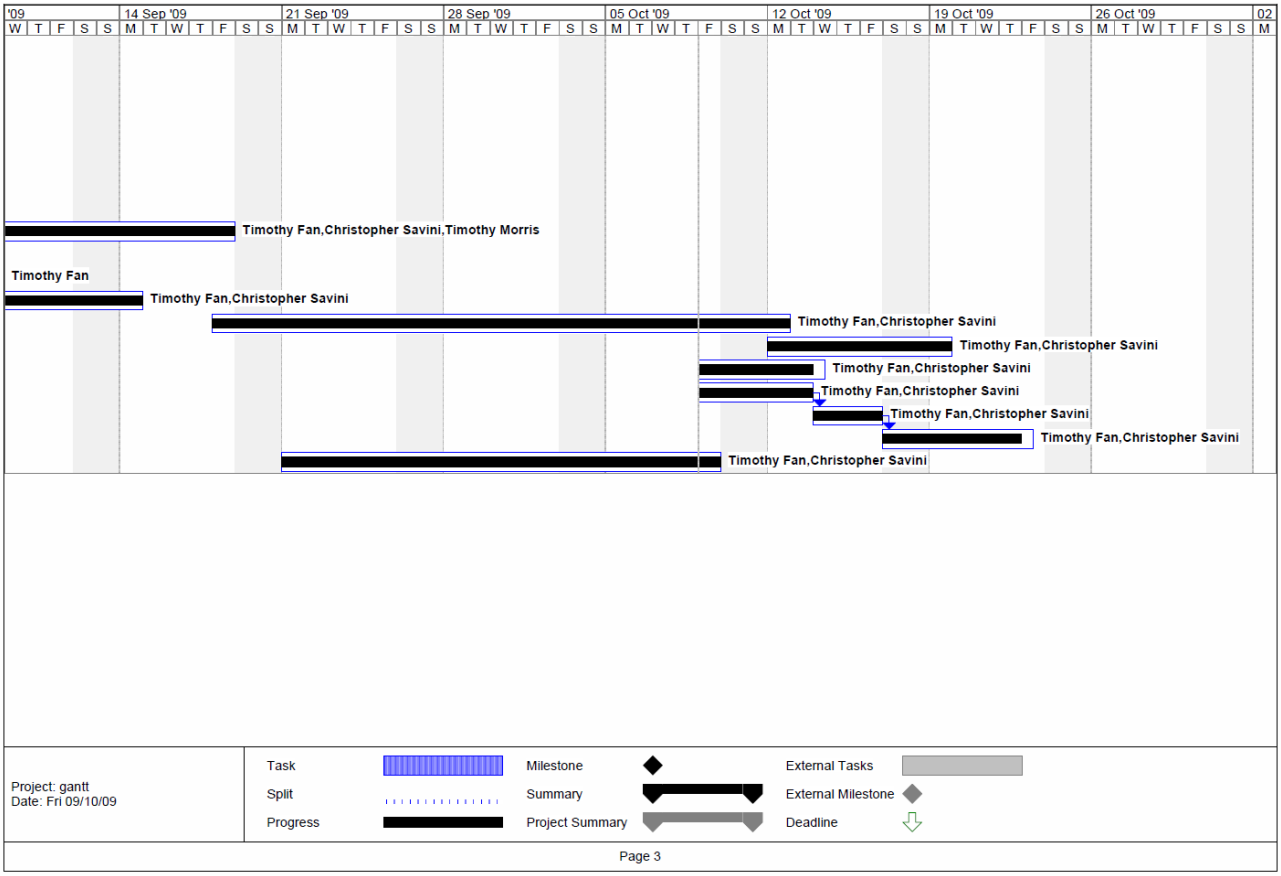
ID	Task Name	Duration	Start	Finish	Predecessors	Resource Names
1	Organise a group	5 days	Mon 06/07/09	Fri 10/07/09		Timothy Fan,Christopher Savini,Timothy Morris
2	Pick up a topic - Stereo Vision	6 days?	Mon 13/07/09	Mon 20/07/09		Christopher Savini,Timothy Morris,Timothy Fan
3	Study Stereo Vision Algorithms	11 days?	Mon 20/07/09	Mon 03/08/09		Christopher Savini
4	Image Research - Stereo Visior	11 days?	Mon 20/07/09	Mon 03/08/09		Timothy Fan
5	Meeting with Joe Y. - Stereo Vis	6 days?	Mon 03/08/09	Mon 10/08/09		Timothy Fan,Christopher Savini,Timothy Morris
6	Reading from Joe Y. - Tensor V	11 days?	Mon 03/08/09	Mon 17/08/09		Timothy Fan,Christopher Savini,Timothy Morris
7	Get Data - IEEE Visualisation C	6 days?	Mon 03/08/09	Mon 10/08/09		Timothy Fan,Christopher Savini,Timothy Morris
8	Proposal Due - Tensor Visusali	6 days?	Mon 10/08/09	Mon 17/08/09		Timothy Fan,Christopher Savini,Timothy Morris
9	Importing Data	30 days?	Mon 10/08/09	Fri 18/09/09		Timothy Fan,Christopher Savini,Timothy Morris
10	Tensor Visusalisation in XNA	15 days?	Mon 10/08/09	Fri 28/08/09		Christopher Savini
11	Impoting Data to Matlab	17 days?	Mon 17/08/09	Tue 08/09/09		Timothy Fan
12	Viewing Data in 3D Slicer	21 days?	Mon 17/08/09	Mon 14/09/09		Timothy Fan,Christopher Savini
13	Drawing Sample Helix on XNA	17 days?	Fri 18/09/09	Mon 12/10/09		Timothy Fan,Christopher Savini
14	Review Patient Data	6 days?	Mon 12/10/09	Mon 19/10/09		Timothy Fan,Christopher Savini
15	Configure Elipsoid glyph and te	3.5 days?	Fri 09/10/09	Wed 14/10/09		Timothy Fan,Christopher Savini
16	Fiber Tracking - Calculate Vect	3 days?	Fri 09/10/09	Tue 13/10/09		Timothy Fan,Christopher Savini
17	Fiber Tracking - Simulate partic	3 days?	Wed 14/10/09	Fri 16/10/09	16	Timothy Fan,Christopher Savini
18	Fiber Tracking - Draw Fibre	4.5 days?	Sat 17/10/09	Fri 23/10/09	17	Timothy Fan,Christopher Savini
19	Progress Report	15 days	Mon 21/09/09	Fri 09/10/09		Timothy Fan,Christopher Savini



19.


20 Jul '09
27 Jul '09
03 Aug '09
10 Aug '09
17 Aug '09
24 Aug '09
31 Aug '09
07 Se
T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T
istopher Savini,Timothy Morris
Christopher Savini,Timothy Morris,Timothy Fan
Christopher Savini
Timothy Fan
Timothy Fan,Christopher Savini,Timothy Morris
Timothy Fan,Christopher Savini,Timothy Morris
Timothy Fan,Christopher Savini,Timothy Morris
Timothy Fan,Christopher Savini,Timothy Morris
Christopher Savini
Project: gantt
Date: Fri 09/10/09
Task
Split
Progress
Milestone
Summary
Project Summary
External Tasks
External Milestone
Deadline
Page 2

20.



Project: gantt
Date: Fri 09/10/09

Task		Milestone		External Tasks	
Split		Summary		External Milestone	
Progress		Project Summary		Deadline	

21. Problems

22. Conclusions

23. References

24. Appendix

1. Interactive Brainviewer

```
using System;

namespace BrainViewer
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        static void Main(string[] args)
        {
            using (Game1 game = new Game1())
            {
                game.Run();
            }
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;


namespace BrainViewer
{
    /// <summary>
    /// An entitiy in the world (model and position and orientation transforms)
    /// </summary>
    public class WorldEntity
    {
        public Model model;
        public Matrix transforms;
        public Vector3 position;
        public Vector3 v1;
        public Vector3 v2;
        public Vector3 v3;
        public int dataIndex;

        public WorldEntity(Model model, Matrix transforms, Vector3 V1, Vector3 V2, Vector3 V3, Vector3
position)
        {
            this.model = model;
            this.transforms = transforms;
            this.v1 = V1;
            this.v2 = V2;
            this.v3 = V3;
            this.position = position;
        }

        public Matrix[] Transforms
        {
            get
            {
                // Copy any parent transforms.
                Matrix[] transforms = new Matrix[model.Bones.Count];
                model.CopyAbsoluteBoneTransformsTo(transforms);
                return transforms;
            }
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections;

using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;


namespace BrainViewer
{
    /// <summary>
    /// A small particle which gets pushed around inside the brain by diffusion
    /// Used to trace fibres for fibre tracking
    /// </summary>
    public class Particle
    {
        static float particleDeltaTime = 5000; //milliseconds

        public Vector3 lastPosition;
        public List<Vector3> pathList;
        int currentLocationIndex;
        float time = 0;

        /// <summary>
        /// Constructor (Particle will move as far away as possible from the last position
        /// </summary>
        /// <param name="lastPos">Initial last position to move away from</param>
        /// <param name="position">Current position</param>
        public Particle(Vector3 lastPos, Vector3 position)
        {
            pathList = new List<Vector3>();
            pathList.Add(position);
            currentLocationIndex = 0;
            this.lastPosition = lastPos;
        }

        /// <summary>
        /// Accesor for position
        /// </summary>
        public Vector3 GetCurrentPosition
        {
            get
            {
                return pathList[currentLocationIndex];
            }
        }

        /// <summary>
        /// Adds a position to the fibre list
        /// </summary>
        /// <param name="position">position to add</param>
        public void AddPosition(Vector3 position)
        {
            pathList.Add(position);
            Next();
        }

        /// <summary>
        /// Cycle through the particle positions
        /// </summary>
        public void Next()
        {
            lastPosition = pathList[currentLocationIndex];
            currentLocationIndex = (currentLocationIndex + 1) % pathList.Count;
        }

        /// <summary>
        /// Moves the particle to the next frame after waiting particleDeltaTime milliseconds
        /// </summary>
```

```
        /// <param name="time">Elapsed time between frames</param>
        public void Next(float time)
        {
            this.time += time;
            if (this.time > particleDeltaTime)
            {
                this.time = 0;
                Next();
            }
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
using Microsoft.Xna.Framework.Design;

namespace BrainViewer
{
    /// <summary>
    /// Camera
    /// </summary>
    public class Camera
    {

        public Vector3 cameraPosition, cameraTarget, cameraUpVector;
        public float fieldOfView, aspectRatio, nearPlaneDistance, farPlaneDistance;

        public Camera(Vector3 cameraPosition, Vector3 cameraTarget, Vector3 cameraUpVector, float
fieldOfView, float aspectRatio, float nearPlaneDistance, float farPlaneDistance)
        {
            this.cameraPosition = cameraPosition;
            this.cameraTarget = cameraTarget;
            this.cameraUpVector = cameraUpVector;
            this.fieldOfView = fieldOfView;
            this.aspectRatio = aspectRatio;
            this.nearPlaneDistance = nearPlaneDistance;
            this.farPlaneDistance = farPlaneDistance;
        }

        /// <summary>
        /// Draw the given worldEntity
        /// </summary>
        /// <param name="worldEntity">entity to draw</param>
        public void Draw(WorldEntity worldEntity)
        {
            Matrix view = Matrix.CreateLookAt(cameraPosition, cameraTarget, cameraUpVector);

            Matrix projection = Matrix.CreatePerspectiveFieldOfView(
                                fieldOfView,
                                aspectRatio,
                                nearPlaneDistance,
                                farPlaneDistance);

            Model model = worldEntity.model;
            Matrix[] transforms = worldEntity.Transforms;

            foreach (ModelMesh mesh in model.Meshes)
            {
                foreach (BasicEffect effect in mesh.Effects)
                {
                    effect.EnableDefaultLighting();
                    effect.World = transforms[mesh.ParentBone.Index] * worldEntity.transforms;
                    effect.View = view;
                    effect.Projection = projection;
                }

                mesh.Draw();
            }
        }
    }

}
```

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
using System.IO;
using System.Collections;
using System.Threading;

using System.Security.Cryptography;

using MathNet;
using MathNet.Numerics;
using MathNet.Numerics.LinearAlgebra;

using MathNetMatrix = MathNet.Numerics.LinearAlgebra.Matrix;
using XNAMatrix = Microsoft.Xna.Framework.Matrix;
using MathNetQuaternion = MathNet.Numerics.Quaternion;
using XNAQuaternion = Microsoft.Xna.Framework.Quaternion;


namespace BrainViewer
{
    /// <summary>
    /// This is the main type for the viewer
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        int X = 256, Y = 256, Z = 38; int length = 256 * 256 * 38; WorldEntity[] entities = new WorldEntity[256 * 256 * 38]; string filePath =
String.Concat(Path.GetFullPath("BrainViewer.exe"), "\\..\\..\\..\\..\\Content\\Patient_A\\A-ten-mask.raw");
        //int X = 256, Y = 256, Z = 53; int length = 256 * 256 * 53; WorldEntity[] entities = new WorldEntity[256 * 256 * 53]; string filePath =
String.Concat(Path.GetFullPath("BrainViewer.exe"), "\\..\\..\\..\\..\\Content\\Patient_B\\B-ten-mask.raw");
        int skip = 1;
        float upperRange = 1.0f; float lowerRange = 0.8f;
        int particleCount = 100; int fibreIterations = 50; float particleDelta = 500.0f; float particleSize = 0.2f;
        float scale = 3;//50; // model scale
        float ZAxisScaleFactor = -2.6f; //Z axis scale and flip as indicated from header file
        float farAway = 1000;

        //region of interest
        int XBoundMin = 100; int XBoundMax = 120;
        int YBoundMin = 120; int YBoundMax = 150;
        int ZBoundMin = 10;  int ZBoundMax = 23;

        string eigenvaluePath = String.Concat(Path.GetFullPath("BrainViewer.exe"), "\\..\\..\\..\\..\\Content\\data\\Eigenvalues_Large1A.txt");
        string eigenvectorPath = String.Concat(Path.GetFullPath("BrainViewer.exe"), "\\..\\..\\..\\..\\Content\\data\\Eigenvectors_Large1A.txt");
        string positionsPath = String.Concat(Path.GetFullPath("BrainViewer.exe"), "\\..\\..\\..\\..\\Content\\data\\Positions_Large1A.txt");
        string rotationsPath = String.Concat(Path.GetFullPath("BrainViewer.exe"), "\\..\\..\\..\\..\\Content\\data\\Rotations_Large1A.txt");
        string particlePath = String.Concat(Path.GetFullPath("BrainViewer.exe"), "\\..\\..\\..\\..\\Content\\data\\Particle.txt");
        string partConectionsPath = String.Concat(Path.GetFullPath("BrainViewer.exe"), "\\..\\..\\..\\..\\Content\\data\\particleConections.txt");

        //string eigenvaluePath = String.Concat(Path.GetFullPath("BrainViewer.exe"), "\\..\\..\\..\\..\\Content\\data\\Eigenvalues_Large1B.txt");
        //string eigenvectorPath = String.Concat(Path.GetFullPath("BrainViewer.exe"), "\\..\\..\\..\\..\\Content\\data\\Eigenvectors_Large1B.txt");
        //string positionsPath = String.Concat(Path.GetFullPath("BrainViewer.exe"), "\\..\\..\\..\\..\\Content\\data\\Positions_Large1B.txt");
        //string rotationsPath = String.Concat(Path.GetFullPath("BrainViewer.exe"), "\\..\\..\\..\\..\\Content\\data\\Rotations_Large1B.txt");
        //string particlePath = String.Concat(Path.GetFullPath("BrainViewer.exe"), "\\..\\..\\..\\..\\Content\\data\\ParticleB.txt");
        //string partConectionsPath = String.Concat(Path.GetFullPath("BrainViewer.exe"), "\\..\\..\\..\\..\\Content\\data\\ParticleConectionsB.txt");

        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        Camera camera;
        Particle[] particle;
        Model particleModel;

        Model model;

        int[] pos = new int[3];
        int confidenceCount = 0;

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            this.graphics.IsFullScreen = false;
            Content.RootDirectory = "Content";
        }

        /// <summary>
        /// Convert an index into the list data to a coordinate is space
        /// The data is stored in a regular grid with regular spacing
        /// and is in a right handed coordinate system. The grid is
        /// traversed in the order X Y Z
        /// </summary>
        /// <param name="index"> index into list</param>
        /// <returns>[X Y Z] coordinate </returns>
        private int[] ConvertIndexToCoordinates(int index)
        {
            //inverts index = X + (xmax * Y) + (xmax * ymax * Z)
            int x = index % X;
            int y = ((index - x)/ X) % Y ;
            int z = ((index - x) - (X * y)) / (X * Y);
            int[] pos = new int[3];
            pos[0] = x; pos[1] = y; pos[2] = z;
            return pos;
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before starting to run.
        /// This is where it can query for any required services and load any non-graphic
        /// related content.  Calling base.Initialize will enumerate through any components
        /// and initialize them as well.
        /// </summary>
        protected override void Initialize()
        {
            // TODO: Add your initialization logic here

            base.Initialize();
        }

        /// <summary>
        /// LoadContent will be called once per game and is the place to load
        /// all of your content.
        /// </summary>
        protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);
            float aspectRatio = (float)GraphicsDevice.Viewport.Width / (float)GraphicsDevice.Viewport.Height;
            camera = new Camera(new Vector3(0, 0, -100), Vector3.Zero, Vector3.Up, MathHelper.PiOver4, aspectRatio, 1f, 40000f);
```

```
            // Create an object
            model = Content.Load<Model>("Models\\Colour_Cube");
            particleModel = Content.Load<Model>("Models\\ball_08");

            //Data = new float[length, 7];
            float confidence, dxx, dxy, dxz, dyy, dyz, dzz;
            MathNetMatrix V1 = new MathNet.Numerics.LinearAlgebra.Matrix(3, 3);
            Vector3 v = new Vector3();
            Vector3 v1 = new Vector3();
            Vector3 v2 = new Vector3();
            Vector3 v3 = new Vector3();
            Vector3 n = new Vector3(0.0f, 0.0f, 0.0f);
            XNAMatrix rotate = new Microsoft.Xna.Framework.Matrix(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
            XNAMatrix position = new Microsoft.Xna.Framework.Matrix(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
            XNAMatrix stretch = new Microsoft.Xna.Framework.Matrix(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
            Vector3 posVec = new Vector3(0, 0, 0);
            XNAMatrix transforms = new Microsoft.Xna.Framework.Matrix(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
            MathNetMatrix tensor = new MathNet.Numerics.LinearAlgebra.Matrix(3, 3);
            EigenvalueDecomposition e;

            Stream stream = File.Open(filePath, FileMode.Open);
            BinaryReader binReader = new BinaryReader(stream);
            int skipCount = 0;
            float l1, l2, l3;

            StreamWriter EigenvaluesWriter = new StreamWriter(eigenvaluePath);
            StreamWriter eigenvectorsWriter = new StreamWriter(eigenvectorPath);
            StreamWriter positionsWriter = new StreamWriter(positionsPath);
            StreamWriter rotationsWriter = new StreamWriter(rotationsPath);

            int[] XYZ = new int[3];

            byte[] bytes = new byte[4];//32 bit -> 4*8 = 32
            for (int i = 0; i < length; i++)
            {
                //read confidence
                confidence = binReader.ReadSingle();

                if ((upperRange >= confidence) && (confidence >= lowerRange))
                {
                    XYZ = ConvertIndexToCoordinates(i);

                    if (((skipCount % skip) == 0) && (XBoundMin < XYZ[0]) && (XYZ[0] < XBoundMax)
                                                  && (YBoundMin < XYZ[1]) && (XYZ[1] < YBoundMax)
                                                  && (ZBoundMin < XYZ[2]) && (XYZ[2] < ZBoundMax))
                    {
                        skipCount++;
                        dxx = binReader.ReadSingle();
                        dxy = binReader.ReadSingle();
                        dxz = binReader.ReadSingle();
                        dyy = binReader.ReadSingle();
                        dyz = binReader.ReadSingle();
                        dzz = binReader.ReadSingle();

                        tensor[0, 0] = dxx; tensor[0, 1] = dxy; tensor[0, 2] = dxz; tensor[1, 0] = dxy; tensor[1, 1] = dyy; tensor[1, 2] = dyz; tensor[2, 0] = dxz; tensor[2,
1] = dyz; tensor[2, 2] = dzz;
                        e = new EigenvalueDecomposition(tensor);
                        l1 = (float)e.EigenValues[0].Real; l2 = (float)e.EigenValues[1].Real; l3 = (float)e.EigenValues[2].Real;

                        v1.X = (float)e.EigenVectors[0, 0]; v1.Y = (float)e.EigenVectors[1, 0]; v1.Z = (float)e.EigenVectors[2, 0];
                        v2.X = (float)e.EigenVectors[0, 1]; v2.Y = (float)e.EigenVectors[1, 1]; v2.Z = (float)e.EigenVectors[2, 1];
                        v3.X = (float)e.EigenVectors[0, 2]; v3.Y = (float)e.EigenVectors[1, 2]; v3.Z = (float)e.EigenVectors[2, 2];
                        v = v1;
                        v.Normalize();

                        n.X = 1.0f; n.Y = 0.0f; n.Z = 0.0f;

                        rotate = XNAMatrix.CreateFromAxisAngle(Vector3.Cross(v, n), (float)MathNet.Numerics.Trig.InverseCosine(Vector3.Dot(v, n)));

                        pos = ConvertIndexToCoordinates(i);

                        posVec.X = pos[0]; posVec.Y = pos[1]; posVec.Z = pos[2] * ZAxisScaleFactor; //header file specified this scaling
                        position = XNAMatrix.CreateTranslation(posVec);

                        stretch = XNAMatrix.CreateScale(l1 * scale, l2 * scale, l3 * scale);
                        transforms = stretch * rotate * position;

                        entities[confidenceCount] = new WorldEntity(model, transforms, v1 * l1, v2 * l2, v3 * l3, posVec);

                        confidenceCount++;

                        #region writeData
                        {
                            EigenvaluesWriter.Write(l1); EigenvaluesWriter.Write(" ");
                            EigenvaluesWriter.Write(l2); EigenvaluesWriter.Write(" ");
                            EigenvaluesWriter.Write(l3); EigenvaluesWriter.Write(" ");
                            EigenvaluesWriter.WriteLine(" ");

                            eigenvectorsWriter.Write(e.EigenVectors[0, 0]); eigenvectorsWriter.Write(" ");
                            eigenvectorsWriter.Write(e.EigenVectors[1, 0]); eigenvectorsWriter.Write(" ");
                            eigenvectorsWriter.Write(e.EigenVectors[2, 0]); eigenvectorsWriter.Write(" ");
                            eigenvectorsWriter.Write(e.EigenVectors[0, 1]); eigenvectorsWriter.Write(" ");
                            eigenvectorsWriter.Write(e.EigenVectors[1, 1]); eigenvectorsWriter.Write(" ");
                            eigenvectorsWriter.Write(e.EigenVectors[2, 1]); eigenvectorsWriter.Write(" ");
                            eigenvectorsWriter.Write(e.EigenVectors[0, 2]); eigenvectorsWriter.Write(" ");
                            eigenvectorsWriter.Write(e.EigenVectors[1, 2]); eigenvectorsWriter.Write(" ");
                            eigenvectorsWriter.Write(e.EigenVectors[2, 2]); eigenvectorsWriter.Write(" ");
                            eigenvectorsWriter.WriteLine("");

                            //m11 m12 m13 m21 m22 m23 m31 m32 m33
                            rotationsWriter.Write(rotate.M11); rotationsWriter.Write(" ");
                            rotationsWriter.Write(rotate.M12); rotationsWriter.Write(" ");
                            rotationsWriter.Write(rotate.M13); rotationsWriter.Write(" ");
                            rotationsWriter.Write(rotate.M21); rotationsWriter.Write(" ");
                            rotationsWriter.Write(rotate.M22); rotationsWriter.Write(" ");
                            rotationsWriter.Write(rotate.M23); rotationsWriter.Write(" ");
                            rotationsWriter.Write(rotate.M31); rotationsWriter.Write(" ");
                            rotationsWriter.Write(rotate.M32); rotationsWriter.Write(" ");
                            rotationsWriter.Write(rotate.M33); rotationsWriter.Write(" ");
                            rotationsWriter.WriteLine("");

                            positionsWriter.Write(posVec.X); positionsWriter.Write(" ");
                            positionsWriter.Write(posVec.Y); positionsWriter.Write(" ");
                            positionsWriter.Write(posVec.Z); positionsWriter.Write(" ");
                            positionsWriter.WriteLine("");

                        }
                        #endregion

                    }
                    else
                    {
                        binReader.ReadSingle();//dxx
```

```
                    binReader.ReadSingle();//dxy
                    binReader.ReadSingle();//dxz
                    binReader.ReadSingle();//dyy
                    binReader.ReadSingle();//dyz
                    binReader.ReadSingle();//dzz
                    skipCount++;
                }
            }
            else //discard the values
            {
                binReader.ReadSingle();//dxx
                binReader.ReadSingle();//dxy
                binReader.ReadSingle();//dxz
                binReader.ReadSingle();//dyy
                binReader.ReadSingle();//dyz
                binReader.ReadSingle();//dzz
            }
        };

        stream.Close();
        rotationsWriter.Close();
        positionsWriter.Close();
        eigenvectorsWriter.Close();
        EigenvaluesWriter.Close();

        //Create Particles

        //random number
        RNGCryptoServiceProvider seeder = new RNGCryptoServiceProvider();
        byte[] randomNumber = new byte[4];
        seeder.GetBytes(randomNumber);
        int randomNum = BitConverter.ToInt32(randomNumber, 0);
        Random randomGenerator = new Random(randomNum);

        particle = new Particle[particleCount];
        Vector3 randomPosition = new Vector3();
        Vector3 center = new Vector3(X/2, Y/2, Z*ZAxisScaleFactor/2);
        for (int j = 0; j < particleCount; j++)
        {
            //scatter some particles at some points
            int ranIndex = randomGenerator.Next(0, confidenceCount);
            randomPosition = entities[ranIndex].position;
            particle[j] = new Particle((randomPosition - center)*farAway + randomPosition, randomPosition);
        }

        //foreach particle find nearest neighbour, move in direction of nearest eigenvector, store new position
        foreach (Particle P in particle)
        {
            int r = 0;
            while((r < fibreIterations) && (0 <= P.GetCurrentPosition.X ) && (P.GetCurrentPosition.X <= X )
                                       && (0 <= P.GetCurrentPosition.Y ) && (P.GetCurrentPosition.Y <= Y )
                                       && (0 >= P.GetCurrentPosition.Z ) && (P.GetCurrentPosition.Z >= Z * ZAxisScaleFactor) //check inside boundaries

                                       && (XBoundMin <= P.GetCurrentPosition.X) && (P.GetCurrentPosition.X <= XBoundMax)
                                       && (YBoundMin <= P.GetCurrentPosition.Y) && (P.GetCurrentPosition.Y <= YBoundMax)
                                       && (ZBoundMin >= P.GetCurrentPosition.Z) && (P.GetCurrentPosition.Z >= ZBoundMax * ZAxisScaleFactor) //check inside region of
interest
                                       )
            {
                Vector3[] p = new Vector3[8];
                int k = 0; int Index = 0;
                float smallest_distance = (entities[0].position - P.GetCurrentPosition).LengthSquared(); //use length^2 for speed
                while (entities[k] != null) //loop over world entities
                {
                    float L = (entities[k].position - P.GetCurrentPosition).LengthSquared();
                    if (L < smallest_distance)
                    {
                        smallest_distance = L; Index = k;
                    }
                    k++;
                }

                //average (add together and divide by 3)
                p[0] = P.GetCurrentPosition + ((( 1 * entities[Index].v1 + entities[Index].v2 + entities[Index].v3) * particleDelta)/3);
                p[1] = P.GetCurrentPosition + ((( 1 * entities[Index].v1 + entities[Index].v2 - entities[Index].v3) * particleDelta)/3);
                p[2] = P.GetCurrentPosition + ((( 1 * entities[Index].v1 - entities[Index].v2 + entities[Index].v3) * particleDelta)/3);
                p[3] = P.GetCurrentPosition + ((( 1 * entities[Index].v1 - entities[Index].v2 - entities[Index].v3) * particleDelta)/3);
                p[4] = P.GetCurrentPosition + (((-1 * entities[Index].v1 + entities[Index].v2 + entities[Index].v3) * particleDelta)/3);
                p[5] = P.GetCurrentPosition + (((-1 * entities[Index].v1 + entities[Index].v2 - entities[Index].v3) * particleDelta)/3);
                p[6] = P.GetCurrentPosition + (((-1 * entities[Index].v1 - entities[Index].v2 + entities[Index].v3) * particleDelta)/3);
                p[7] = P.GetCurrentPosition + (((-1 * entities[Index].v1 - entities[Index].v2 - entities[Index].v3) * particleDelta)/3);

                float[] f = new float[8];
                f[0] = (p[0] - P.lastPosition).LengthSquared();
                f[1] = (p[1] - P.lastPosition).LengthSquared();
                f[2] = (p[2] - P.lastPosition).LengthSquared();
                f[3] = (p[3] - P.lastPosition).LengthSquared();
                f[4] = (p[4] - P.lastPosition).LengthSquared();
                f[5] = (p[5] - P.lastPosition).LengthSquared();
                f[6] = (p[6] - P.lastPosition).LengthSquared();
                f[7] = (p[7] - P.lastPosition).LengthSquared();

                float max = f[0];
                int maxIndex = 0;
                for (int i = 1; i != f.Length; ++i)
                {
                    if (f[i] > max)
                    {
                        max = f[i];
                        maxIndex = i;
                    }
                }

                P.AddPosition(p[maxIndex]);

                r++;
            }
        }

        StreamWriter particleWriter = new StreamWriter(particlePath);
        StreamWriter particleConectionsWriter = new StreamWriter(partConectionsPath);

        //write fibre positions to file
        for (int fibreIndex = 0; fibreIndex < particle.Length; fibreIndex++) //for each fibre
        {
            for (int partIndex = 0; partIndex < particle[fibreIndex].pathList.Count; partIndex++)//for each particle in a fibre
            {
                particleWriter.Write(particle[fibreIndex].pathList[partIndex].X); particleWriter.Write(" ");
                particleWriter.Write(particle[fibreIndex].pathList[partIndex].Y); particleWriter.Write(" ");
                particleWriter.Write(particle[fibreIndex].pathList[partIndex].Z); particleWriter.Write(" ");
                particleWriter.WriteLine("");
            }
        }
```

```
    //write connections
    int connectionCounter = 0;
    int lineCount = 0;
    for (int lineIndex = 0; lineIndex < particle.Length; lineIndex++)
    {
        for (int particleIndex = 0; particleIndex < particle[lineIndex].pathList.Count - 1; particleIndex++)
        {
            lineCount++;
            particleConectionsWriter.Write(connectionCounter); particleConectionsWriter.Write(" ");
            particleConectionsWriter.Write(++connectionCounter); particleConectionsWriter.Write(" ");
            particleConectionsWriter.WriteLine("");
        }
        connectionCounter++;
    }

    particleWriter.Close();
    particleConectionsWriter.Close();
}

/// <summary>
/// UnloadContent will be called once per game and is the place to unload
/// all content.
/// </summary>
protected override void UnloadContent()
{
    // TODO: Unload any non ContentManager content here
}

/// <summary>
/// Allows the game to run logic such as updating the world,
/// checking for collisions, gathering input, and playing audio.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (Keyboard.GetState().IsKeyDown(Keys.Escape))
        this.Exit();

    if (Keyboard.GetState().IsKeyDown(Keys.PageUp))
    {
        Vector3 shift = camera.cameraUpVector;
        camera.cameraPosition += shift ;
        camera.cameraTarget += shift;
    }

    if (Keyboard.GetState().IsKeyDown(Keys.PageDown))
    {
        Vector3 shift = -1*camera.cameraUpVector;
        camera.cameraPosition += shift;
        camera.cameraTarget += shift;
    }

    if (Keyboard.GetState().IsKeyDown(Keys.D))
    {
        Vector3 to = camera.cameraTarget - camera.cameraPosition;
        to.Normalize();
        Vector3 shift = Vector3.Cross(to, camera.cameraUpVector);
        camera.cameraPosition += shift;
        camera.cameraTarget += shift;
    }

    if (Keyboard.GetState().IsKeyDown(Keys.A))
    {
        Vector3 to = camera.cameraTarget - camera.cameraPosition;
        to.Normalize();
        Vector3 shift = Vector3.Cross(camera.cameraUpVector, to);
        camera.cameraPosition += shift;
        camera.cameraTarget += shift;
    }

    //boom
    if (Keyboard.GetState().IsKeyDown(Keys.W))
    {
        Vector3 shift = camera.cameraTarget - camera.cameraPosition;
        shift.Normalize();
        camera.cameraPosition += shift;
        camera.cameraTarget += shift;
    }

    if (Keyboard.GetState().IsKeyDown(Keys.S))
    {
        Vector3 shift = camera.cameraPosition - camera.cameraTarget;
        shift.Normalize();
        camera.cameraPosition += shift;
        camera.cameraTarget += shift;
    }

    //Rotate
    if (Keyboard.GetState().IsKeyDown(Keys.Left))
    {
                Vector3 toVector = camera.cameraTarget - camera.cameraPosition;
                XNAMatrix rotate = XNAMatrix.CreateFromAxisAngle(camera.cameraUpVector, 3.141592653589f/500);
        XNAQuaternion q = XNAQuaternion.CreateFromRotationMatrix(rotate);
        Vector3 newTo = Vector3.Transform(toVector, q);
        camera.cameraTarget = camera.cameraPosition + newTo;
    }

    if (Keyboard.GetState().IsKeyDown(Keys.Right))
    {
        Vector3 toVector = camera.cameraTarget - camera.cameraPosition;
        XNAMatrix rotate = XNAMatrix.CreateFromAxisAngle(camera.cameraUpVector * -1, 3.141592653589f / 500);
        XNAQuaternion q = XNAQuaternion.CreateFromRotationMatrix(rotate);
        Vector3 newTo = Vector3.Transform(toVector, q);
        camera.cameraTarget = camera.cameraPosition + newTo;
    }

    //Roll
    if (Keyboard.GetState().IsKeyDown(Keys.RightShift))
    {
        Vector3 toVector = camera.cameraTarget - camera.cameraPosition;
        XNAMatrix rotate = XNAMatrix.CreateFromAxisAngle(toVector, 3.141592653589f / 10000);
        XNAQuaternion q = XNAQuaternion.CreateFromRotationMatrix(rotate);
        Vector3 newUp = Vector3.Transform(camera.cameraUpVector, q);
        camera.cameraUpVector = newUp;
    }

    if (Keyboard.GetState().IsKeyDown(Keys.LeftShift))
    {
        Vector3 toVector = camera.cameraTarget - camera.cameraPosition;
        XNAMatrix rotate = XNAMatrix.CreateFromAxisAngle(toVector * -1 , 3.141592653589f / 10000);
        XNAQuaternion q = XNAQuaternion.CreateFromRotationMatrix(rotate);
        Vector3 newUp = Vector3.Transform(camera.cameraUpVector, q);
        camera.cameraUpVector = newUp;
```

```
            }

            //pitch
            if (Keyboard.GetState().IsKeyDown(Keys.Up))
            {
                Vector3 toVector = camera.cameraTarget - camera.cameraPosition;
                Vector3 right = Vector3.Cross(toVector, camera.cameraUpVector);
                right.Normalize();
                XNAMatrix rotate = XNAMatrix.CreateFromAxisAngle(right, 3.141592653589f / 500);
                XNAQuaternion q = XNAQuaternion.CreateFromRotationMatrix(rotate);

                Vector3 newUp = Vector3.Transform(camera.cameraUpVector, q);
                Vector3 newTo = Vector3.Transform(toVector, q);

                camera.cameraUpVector = newUp;
                camera.cameraTarget = camera.cameraPosition + newTo;
            }

            //pitch
            if (Keyboard.GetState().IsKeyDown(Keys.Down))
            {
                Vector3 toVector = camera.cameraTarget - camera.cameraPosition;
                Vector3 left = Vector3.Cross(camera.cameraUpVector, toVector);
                left.Normalize();
                XNAMatrix rotate = XNAMatrix.CreateFromAxisAngle(left, 3.141592653589f / 500);
                XNAQuaternion q = XNAQuaternion.CreateFromRotationMatrix(rotate);

                Vector3 newUp = Vector3.Transform(camera.cameraUpVector, q);
                Vector3 newTo = Vector3.Transform(toVector, q);

                camera.cameraUpVector = newUp;
                camera.cameraTarget = camera.cameraPosition + newTo;
            }

            //move particles
            foreach (Particle part in particle)
            {
                part.Next(gameTime.ElapsedRealTime.Milliseconds);
            }

            base.Update(gameTime);
        }

        /// <summary>
        /// This is called when the game should draw itself.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing values.</param>
        protected override void Draw(GameTime gameTime)
        {
            graphics.GraphicsDevice.Clear(Color.CornflowerBlue);

            for (int i = 0; i < confidenceCount; i++)
            {
                camera.Draw(entities[i]);
            }

            //draw particles
            //step particles gameTime
            foreach (Particle p in particle)
            {
                camera.Draw(new WorldEntity
                (
                    particleModel,
                    XNAMatrix.CreateScale(particleSize,particleSize,particleSize)*XNAMatrix.CreateTranslation(p.GetCurrentPosition),
                    new Vector3(0, 0, 0),
                    new Vector3(0, 0, 0),
                    new Vector3(0, 0, 0),
                    new Vector3(0, 0, 0))
                );
                p.Next();
            }


            base.Draw(gameTime);
        }

    }

}
```

2. MATLAB scripts

```
function WriteToVTKFile(Points, Mesh)


numberOfPoints = size(Points, 1);
numberOfCells = size(Mesh, 1);

fid = fopen('OptimisedVTK.vtk','w');

fprintf(fid, '# vtk DataFile Version 3.0\n');
fprintf(fid, 'vtk output\n');
fprintf(fid, 'ASCII\n');
fprintf(fid, 'DATASET UNSTRUCTURED_GRID\n');


%fprintf(fid, 'POINTS %d float\n', numberOfPoints);
%for i=1:numberOfPoints
%    fprintf(fid, '%f %f %f\n', Points(i,1), Points(i,2), Points(i,3));
%end
fprintf(fid, 'POINTS %d float\n', numberOfPoints);
for i=1:numberOfPoints
    fprintf(fid, '%f %f %f\n', Points(i,1), Points(i,2), Points(i,3));
end



fprintf(fid, '\nCELLS %d %d\n', numberOfCells, numberOfCells*5);
for k=1:numberOfCells
    fprintf(fid, '4 ');
    for i = 1:4
        fprintf(fid, '%d ', Mesh(k,i));
    end
    fprintf(fid, '\n');
end
fprintf(fid, '\n');

fprintf(fid, 'CELL_TYPES %d\n', numberOfCells);
for k=1:numberOfCells
    fprintf(fid, '9\n');
end
fprintf(fid, '\n');

fprintf(fid, '\n\nPOINT_DATA %d\n', numberOfPoints);
fprintf(fid, 'SCALARS scalars float 1\n');
fprintf(fid, 'LOOKUP_TABLE default\n');
for k=1:numberOfPoints
    if (mod(k,6) == 0)
        fprintf(fid, '\n');
    end
    fprintf(fid, '%f ', 0.0 );
end
fprintf(fid, '\n');



%fprintf(fid, '\nCELLS %d %d\n', numberOfCells, numberOfCells*5);
%for k=1:numberOfCells
%    fprintf(fid, '4 ');
%    for i = 1:4
%        fprintf(fid, '%d ', Mesh(k,i)-1);
%    end
%    fprintf(fid, '\n');
%end
%fprintf(fid, '\n');

%fprintf(fid, 'CELL_TYPES %d\n', numberOfCells);
%for k=1:numberOfCells
%    fprintf(fid, '9\n');
%end
%fprintf(fid, '\n');

fclose(fid);
```

```
function Write2DX(AllPoints, QuadBase, numberOfEllipsoids, AllPointData, RGB)

numberOfPoints = size(AllPoints, 1);
numberOfQuadBase = size(QuadBase, 1);
numberOfColour = size(RGB, 1);


disp(numberOfEllipsoids);
temp = numberOfPoints/numberOfEllipsoids;

fid = fopen('PatientA1.dx', 'w');
fprintf(fid, 'object 1 class array type float rank 1 shape 3 items %d data follows\n', numberOfPoints);
for i=1:numberOfPoints
    fprintf(fid, '%f %f %f\n', AllPoints(i,1), AllPoints(i,2), AllPoints(i,3));
end
fprintf(fid, 'object 2 class array type int rank 1 shape 4 items %d data follows\n',
numberOfEllipsoids*numberOfQuadBase);
for i = 1:numberOfEllipsoids
    for j = 1:numberOfQuadBase
        fprintf(fid, '%d %d %d %d\n', QuadBase(j,1)+(i-1)*temp, QuadBase(j,2)+(i-1)*temp, QuadBase(j,3)+(i-
1)*temp, QuadBase(j,4)+(i-1)*temp);
    end
end
fprintf(fid, 'attribute "element type" string "quads"\n');
fprintf(fid, 'attribute "ref" string "positions"\n\n');
fprintf(fid, 'object 3 class array type float rank 1 shape 3 items %d data follows\n', numberOfColour);
for i=1:numberOfPoints
    fprintf(fid, '%f %f %f\n', RGB(i,1), RGB(i,2), RGB(i,3));
end
fprintf(fid, 'attribute "dep" string "positions"\n\n');

fprintf(fid, 'object "stuff" class field\n');
fprintf(fid, 'component "positions" value 1\n');
fprintf(fid, 'component "connections" value 2\n');
fprintf(fid, 'component "colors" value 3\n');
fprintf(fid, 'end\n');
fclose(fid);
```

```
function [Data] = readRotationFile(filename)


% Count the Data
fid=fopen(filename,'r');

if fid == -1,
    error('Unable to open specified file');
end

counter = 0;
while 1
    tline = fgetl(fid);
    if ~ischar(tline)
        break; % For end of file recognition
    end
    temp = fscanf(fid, '%f %f %f %f %f %f %f %f %f', [1,9]);
    counter = counter +1;
end

% Create the dataset
Data = zeros(counter, 9);
fclose(fid);

% Read the Data
fid=fopen(filename,'r');
if fid == -1,
    error('Unable to open specified file');
end

counter = 0;


while 1
    tline = fgetl(fid);
    if ~ischar(tline)
        break; % For end of file recognition
    end
    temp = textscan(tline, '%f %f %f %f %f %f %f %f %f');
    counter = counter +1;
    if (size(temp,1) > 0)

        Data(counter,1) = temp{1};
        Data(counter,2) = temp{2};
        Data(counter,3) = temp{3};
        Data(counter,4) = temp{4};
        Data(counter,5) = temp{5};
        Data(counter,6) = temp{6};
        Data(counter,7) = temp{7};
        Data(counter,8) = temp{8};
        Data(counter,9) = temp{9};
    else
        fprintf(1, 'Problem\n');
    end
end
fclose(fid);
```

```
function [Data] = readDataFile(filename)


% Count the Data
fid=fopen(filename,'r');

if fid == -1,
    error('Unable to open specified file');
end

counter = 0;
while 1
    tline = fgetl(fid);
    if ~ischar(tline)
        break; % For end of file recognition
    end
    temp = fscanf(fid, '%f %f %f', [1,3]);
    counter = counter +1;
end

% Create the dataset
Data = zeros(counter, 3);
fclose(fid);

% Read the Data
fid=fopen(filename,'r');
if fid == -1,
    error('Unable to open specified file');
end

counter = 0;


while 1
    tline = fgetl(fid);
    if ~ischar(tline)
        break; % For end of file recognition
    end
    temp = textscan(tline, '%f %f %f');
    counter = counter +1;
    if (size(temp,1) > 0)

        Data(counter,1) = temp{1};
        Data(counter,2) = temp{2};
        Data(counter,3) = temp{3};
    else
        fprintf(1, 'Problem\n');
    end
end
fclose(fid);
```

```
function GenerateScene()
FibreWrite2DX();

% positionsFile = './data/Positions_Large.txt';
% eigenFile = './data/EigenValue_Large.txt';
% rotationFile = './data/Rotation_Large.txt';
% eigenVector = './data/EigenVector_Large.txt';

positionsFile = './data/Positions_Large1A.txt';
eigenFile = './data/Eigenvalues_Large1A.txt';
rotationFile = './data/Rotations_Large1A.txt';
eigenVector = './data/Eigenvectors_Large1A.txt';


Positions = readDataFile(positionsFile);
EigenData = readDataFile(eigenFile);
RotationData = readRotationFile(rotationFile);
EigenVector = readRotationFile(eigenVector);

numberOfGlyphs = size(Positions,1);

% define the ellipsoid density
nx = 20;
ny = 20;

xMin = -1*pi*0.5;
xMax = pi*0.5;
yMin = -1*pi;
yMax =pi;

xstep = (xMax-xMin)/(nx-1);
ystep = (yMax-yMin)/(ny-1);


%RGB Colour from either EigenValues or EigenVectors
RGB = zeros(numberOfGlyphs*nx*ny,3);
for i = 1:numberOfGlyphs
    for j = 1:nx*ny
        RGB(j+(i-1)*nx*ny,:) = abs(EigenVector(i,1:3));
        %RGB(j+(i-1)*nx*ny,:) = abs(100*EigenData(i,1:3));
    end
end

GridPositions = zeros(nx*ny, 2);
counter = 1;
for j = 1:ny
    for i = 1:nx
        GridPositions(counter, 1) = xMin + (i-1)*xstep;
        GridPositions(counter, 2) = yMin + (j-1)*ystep;
        counter = counter +1;
    end
end

%TRI = delaunay(GridPositions(:,1), GridPositions(:,2));
QUADS = zeros((nx-1)*(ny-1), 4);
numberOfQuads = (nx-1)*(ny-1);

counter = 1;
mx = nx;
for j = 0:(ny-2)
    for i = 0:(nx-2)
        QUADS(counter, 1) = i+j*mx;
        QUADS(counter, 2) = i+j*mx + 1;

        QUADS(counter, 3) = i+(j+1)*mx;
        QUADS(counter, 4) = i+(j+1)*mx + 1;
        counter = counter +1;
    end
end


%PointPlot3D(GridPositions(:,1), GridPositions(:,2), zeros(nx*ny,1));
%quadmesh(QUADS, GridPositions(:,1), GridPositions(:,2), zeros(nx*ny,1));
%numberOfGlyphs = 5;
AllPoints = zeros(nx*ny*numberOfGlyphs, 3);
%AllPointData = zeros(nx*ny*numberOfGlyphs);
TempPoints = zeros(nx*ny, 3);
```

```
RotationMatrix = zeros(3,3);
for i = 1:numberOfGlyphs
    % Create the rotation Matrix
    RotationMatrix = [RotationData(i,1:3); RotationData(i,4:6); RotationData(i,7:9)];

    % For this data point, generate a glyph at this position
    TempPoints = GenerateEllipsoid(GridPositions, Positions(i, :), EigenData(i,:), 1, RotationMatrix);

    % Now to stick the new points into our big structure
    AllPoints((i-1)*nx*ny +1: i*nx*ny, :) = TempPoints(:,:);
    % Make each ellipsoid a single colour
    AllPointData((i-1)*nx*ny +1: i*nx*ny, 1) = i;
    % Make each ellipsoid multi colour
    %AllPointData((i-1)*nx*ny +1: i*nx*ny, 1) = (i-1)*(1:nx*ny);
end
%disp(AllPoints);

%PointPlot3D(AllPoints(:,1), AllPoints(:,2), AllPoints(:,3));

Write2DX(AllPoints, QUADS, numberOfGlyphs, AllPointData,RGB);
%WriteToVTKFile(AllPoints, QUADS);




%
%
%
function [newPoints] = GenerateEllipsoid(Grid, Origin, eigenData, Scale, Rotation)

numberOfPoints = size(Grid, 1);
newPoints = zeros(numberOfPoints, 3);
for i = 1:numberOfPoints
    newPoints(i, 1) = 1*cos(Grid(i,1))*cos(Grid(i,2));%*Scale + Origin(1,1);
    newPoints(i, 2) = 1*cos(Grid(i,1))*sin(Grid(i,2));%*Scale + Origin(1,2);
    newPoints(i, 3) = 1*sin(Grid(i,1));%*Scale + Origin(1,3);

    newPoints(i,:) = Rotation*newPoints(i,:)';

    newPoints(i, 1) = eigenData(1,1)*newPoints(i,1)*Scale + Origin(1,1);
    newPoints(i, 2) = eigenData(1,2)*newPoints(i,2)*Scale + Origin(1,2);
    newPoints(i, 3) = eigenData(1,3)*newPoints(i,3)*Scale + Origin(1,3);
end


return


function PointPlot3D(X1, Y1, Z1)
%CREATEFIGURE(X1,Y1,Z1)
%  X1:  vector of x data
%  Y1:  vector of y data
%  Z1:  vector of z data

%  Auto-generated by MATLAB on 14-Aug-2009 10:16:35

% Create figure
figure1 = figure('XVisual',...
    '0x23 (TrueColor, depth 32, RGB mask 0xff0000 0xff00 0x00ff)');

% Create axes
axes1 = axes('Parent',figure1);
view(axes1,[-19.5 60]);
hold(axes1,'all');

% Create plot3
plot3(X1,Y1,Z1,'Marker','.','LineStyle','none');
```

```
function FibreWrite2DX()

load('./data/Particle.txt')
load('./data/ParticleConections.txt')
numberOfPoints = size(Particle,1);
numberOfPointsCon = size(ParticleConections,1);


fid = fopen('FibrePatientA1.dx', 'w');
fprintf(fid, 'object 1 class array type float rank 1 shape 3 items %d data follows\n', numberOfPoints);
for i=1:numberOfPoints
    fprintf(fid, '%f %f %f\n', Particle(i,1), Particle(i,2), Particle(i,3));
end
fprintf(fid, 'object 2 class array type int rank 1 shape 2 items %d data follows\n', numberOfPointsCon);
for i = 1:numberOfPointsCon
        fprintf(fid, '%d %d\n', ParticleConections(i,1), ParticleConections(i,2));
end
fprintf(fid, 'attribute "element type" string "lines"\n');

fprintf(fid, 'object "stuff" class field\n');
fprintf(fid, 'component "positions" value 1\n');
fprintf(fid, 'component "connections" value 2\n');
fprintf(fid, 'end\n');
fclose(fid);
```