university of groningen

February 22, 2013

THESIS

# Abstraction of Dense Line Data Using Shaped Integration and Field Compensation

*Author:*
D.L. Zittersteyn

*Supervisors:*
Dr. H. Bekker
Dr. M.H.F. Wilkinson

*I didn't have time to write a short letter,*
*so I wrote a long one instead.*

— Mark Twain

# CONTENTS

# 1 STRUCTURE OF THIS PAPER

## 1.1 INTRODUCTION TO THE FIELD

In the first chapters, specifically Chapter 2 through 5, an introduction is given to dense line data and vector fields, and their sources and applications.

**Chapter 2** presents a short introduction to the field of dense line data and vector fields, and methods of abstraction.

**Chapter 3** shows different sources of dense line data and introduces the term *coherent* dense line data.

**Chapter 4** goes into the main application of vector field data, i. e. flow.

**Chapter 5** presents the development of diffusion tensor imaging, and the dense line data that results from it.

## 1.2 VISUALIZATION AND ITS ISSUES

**Chapter 6** gives a descriptions of the issues currently associated with visualizing dense line and vector field datasets.

## 1.3 PROPOSED METHOD

In Chapter 7 trough 10 we set ourselves a number of goals and propose methods to reach them.

**Chapter 7** sets the goals and gives an outline of the proposed solution. The chapter also contains an overview of the terms used in this paper.

**Chapter 8** introduces the data structure that will be used in the following chapters: the segment set.

**Chapter 9** formalizes the way we generate the abstraction, by showing the method of integration.

**Chapter 10** goes into the details of how a segment set is represented by abstracted paths.

In the last chapters, Chapter 11 through 14, We will show the results of the proposed method, and present some of the details that were not shown in earlier chapters.

**Chapter 11** shows a number of applications of the proposed method, on both dense line as vector field data.

**Chapter 12** addresses some of the implementation details that significantly speed up the method, and shows some of the prototype versions.

**Chapter 13** introduces a number of possible improvements or alterations to the method that warrant further research.

**Chapter 14** gives a short summary of the contribution the proposed method makes to the field of dense line data visualization.

# 2 INTRODUCTION

Art and science have long tried to visualize the things that are most important to us. From early cave paintings depicting paleolithic scenes of stick figures hunting and gathering, through early anatomical illustrations and the cubist art movement, to modern displays featuring three dimensional visualization and interactivity. Many different methods have been used to convey the essence of a scene.

We live in an era of access to previously unavailable data. Tracking software in cell phones allows us to see the movement of a large number of individuals. New medical technologies give us the ability to look into the human body in great detail. Progress in computational technologies allow us to simulate immensely complex systems. All these advances pave the way for never before possible insight in data.

With great detail however, comes great complexity. While previously data could be manually processed and visualized with simple techniques, automated systems are needed when handling complexity of this magnitude.

## 2.1 DENSE LINE DATA

A category of particular interest is *dense line data*. This kind of data consists of a large number of paths, all with some significance, in a confined 3D space. Visualizing data this complex means dealing with some inherent problems. Firstly there is the fact that as the amount of data grows, the size and resolution of the visualization has to grow with it in order to show the data in a way suitable for human observation.

But even with high-resolution visualization, occlusion will hide parts of the data behind other parts. Along with occlusion, limits to human perception impede the full understanding of the data. A very detailed visualization can simply be too much information to take in.

In Figure 2.1 a naive 3D visualization of white matter nerve fibers in the human brain is shown, illustrating the problems with dense line data. Each fiber is shown as a black line. While with 50 lines (Figure 2.1a) the visualization gives a reasonably good representation of the data, clearly this simple visualization is not suited for the case of 20.000 fibers (Figure 2.1b), suffering from resolution, occlusion and perception issues.

Many methods have been proposed to overcome the problems of visualizing dense line data. The new method proposed and

discussed in this paper is intended as a contribution to visualizing this type of data and aims to introduce a new method of automated processing.

**(a)** 50 fibers **(b)** 20.000 fibers

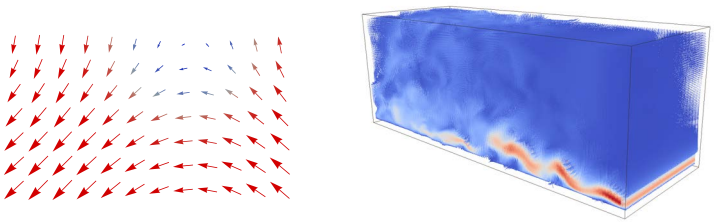
**Figure 2.1:** Simple 3D visualization of white matter fibers in the brain.

## 2.2 VECTOR FIELDS

Another category of data that is known for being hard to visualize is *vector fields*. Vector fields consist of a large number of points, each with an associated vector. This kind of data is often the result of a simulation or measurement of a continuous system, such as air or water flow.

Vector field visualization suffers from many of the same issues as dense line data, as can be seen in Figure 2.2. In this figure a simple visualization is shown, in which each point and its associated vector is shown as an arrow originating from a point, with a length and direction corresponding to the direction and magnitude of the vector.

With the $7\times12$ field of Figure 2.2a this simple visualization is enough to give a good impression of the flow, but with the $130\times260\times240$ field of Figure 2.2b any detail that lies deeper than the first few layers is invisible. As with dense line data, the full dataset is simply too large and complex to visualize and grasp in its entirety, so some form of abstraction is needed.



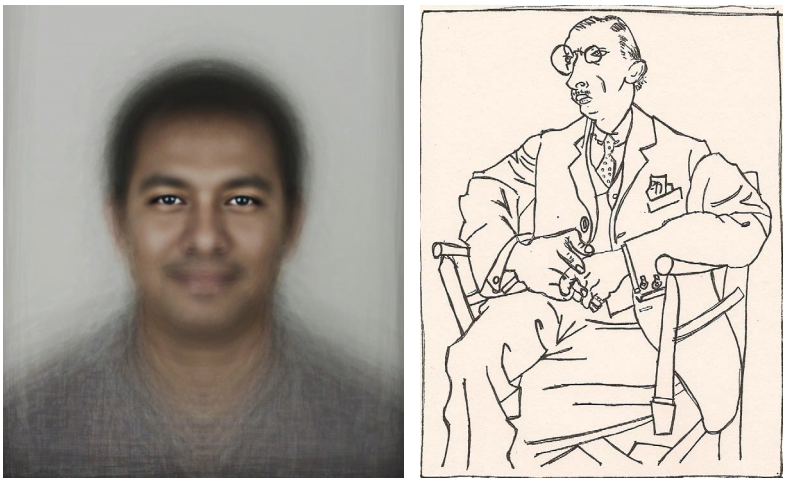
**(a)** $7\times12$ points **(b)** $130\times260\times240$ points

**Figure 2.2:** Simple visualization of flow. The color of the arrow corresponds to the magnitude of the vector (small magnitude is shown as blue, large as red). Dataset courtesy of R. Verstappen.

Methods of abstraction can belong in either of two categories. An abstraction either takes the original data and rearranges it to represent the '*average*' underlying data, or tries to create a new image that captures the '*essence*' of the original data. In Figure 2.3 an illustration of the difference between these concepts is shown.

Figure 2.3a shows a composite image of many photos of adult men of all races. These photos are aligned so the main features overlap, and all images are semitransparent. The resulting image is an 'average' of all the faces in the original images.

In Figure 2.3b the drawing "Portrait of Igor Stavinsky", by Pablo Picasso is shown. While it is clearly not a photo-realistic drawing of Stavinksy, it does make clear his important features. Round glasses, sharp cheekbones, oversized clothes and a mustache, the 'essence' of Igor Stavinsky, are clearly seen in this portrait.



**(a)** Average **(b)** Essence

**Figure 2.3:** (a): David Trood's "Male Face of Humanity" [50], (b): Pablo Picasso's "Portrait of Igor Stravinsky."

Both methods have their own merits. Where it can be said that the *essence* is a clearer, less cluttered representation, the *average* has the advantage that its result can be easily retraced to the original data, and is therefore more easily shown to be objective. While an average view shows the many aspects of the data, the essence captures the parts that are typical. In this way, the essence is quite similar to a caricature.

The method proposed in this paper can be used to generate a caricature-like visualization of a complex dataset. It does not strive to accurately represent the entire dataset, but rather extracts the important, characterizing properties.

In this paper a method is proposed to take dense line data and extract the abstract properties of this data. This data is then used to give context to smaller sections of the original dense line data. By using a small subset of the original data occlusion is reduced, and the visualization can be less complex. It shows the global essence of the data to provide the user with context, and uses a simple visualization to show small details. A way of applying this method to vector fields is also shown.

# 3

# DENSE LINE DATA

Many fields of science and technology deal with large amounts of data that is, in essence, a dense collection of *paths*. Each of these paths in turn consists of a series of *points*, which are connected by *segments*.

The term *coherency* is less exactly defined, but we use it to describe a dataset that consists of paths that share some structure.

Compare for example cooked and uncooked spaghetti. The uncooked spaghetti exhibits clear coherence, as can be seen in Figure 3.1a. In cooked spaghetti (Figure 3.1b) on the other hand, coherence can not be found. The term dense line data will in this paper be used to refer to coherent dense line data. Abstracting incoherent dense line data will in general be very hard, if not impossible, because for abstraction some sort of coherency is needed.

**(a)** Coherent **(b)** Incoherent

**Figure 3.1:** Coherent and non-coherent "dense line data".

This chapter will briefly explore a number of dense line datasets, as well as their origins.

## 3.1 HUMAN TRAVEL

Humans generate a large amount of path data every day, simply by traveling. With modern methods these journeys can be precisely mapped.

From travel registration such as a flight route database (shown in Figure 3.2) or GPS-data from mobile phones, human migration data has never before been avaliable in such quantity and resolution.

With human travel we run into the dense line data limitations quite quickly. The project "Painting Reality" by Iepe Rubingh (shown in Figure 3.3) shows just a few minutes worth of commuters driving over a crossing. This was accomplished by pouring paint at the entrances, so cars crossing would trace their path on the asphalt.

**Figure 3.2:** 59036 airplane routes between 3209 airports, visualized as resp. green lines and red dots [3].

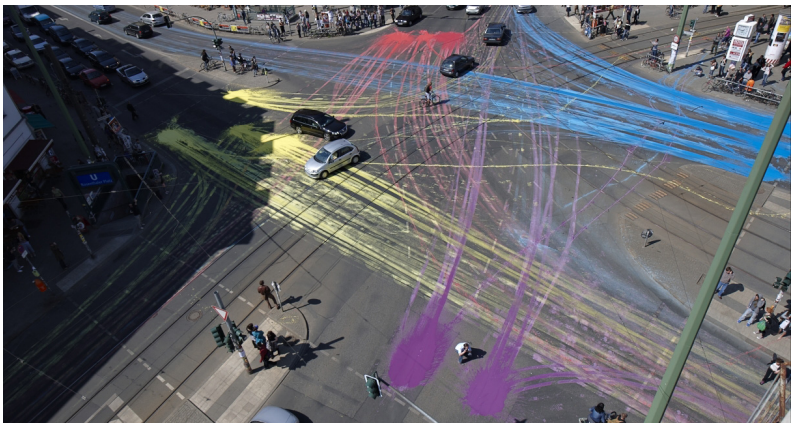


**Figure 3.3:** "Painting Reality" by Iepe Rubingh [45].

If a few minutes worth of traffic over one intersection already yields such a rich dataset, imagine human travel taken over a city, or an even larger scale. Advanced visualization techniques are clearly needed to gain insight into this kind of data.

## 3.2 INTERACTION

Dense lines also make an appearance when visualizing the interaction between many elements. When used in a geographical context, these datasets are quite similar to the human travel data. Compare for example the interaction between friends on Facebook in Figure 3.4 with the airline data from Figure 3.2.

The Facebook friend visualization is a large simplification and generalisation of the original dataset. Facebook has, as of self-reported numbers from November 2011, 721 million users, with 69 billion friendships among them [8]. The data from the visualization takes just 10 million friend pairs, reducing the dataset by a factor of over 7000. The dataset is then further reduced by not drawing the friendships, but drawing a line from city to city, with an opacity relative to the number of friendships between users in those cities [10]. Even then, much of the data is obscured.


**Figure 3.4:** Interaction between people on Facebook [10]. Each line has an opacity relative to the number of friend connections between two cities.

Interaction data can also be more abstract. See for example Figure 3.5, where connections were drawn between websites at the top, and the tracking companies they partner with at the bottom.

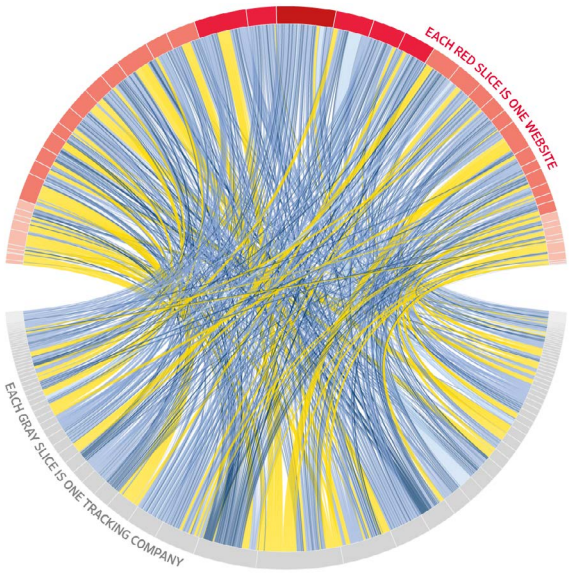



**Figure 3.5:** Tracking methods used by popular websites [29]. Connections between a website at the top, and a tracker at the bottom are colored by type of tracking, ranging from yellow (beneficial) to dark blue (intrusive).

To be able to quickly glean information from any of this data, a new method of visualization is needed.

## 3.3 WHITE BRAIN MATTER

A different source of dense line data is the human brain. The human brain is roughly speaking divisible in three kinds of material: *cerebrospinal fluid* (CSF), *grey matter* and *white matter*. In Figure 3.6 these elements are clearly visible.

While grey matter and CSF are fairly homogenous, white matter shows clear paths. White matter functions as the communication channels between gray matter areas [21]. Take for example the

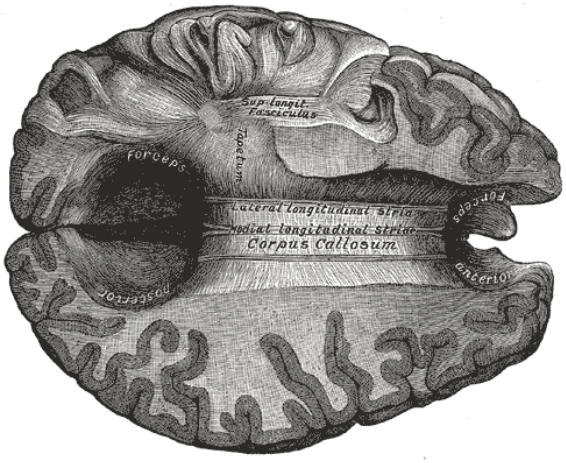



**Figure 3.6:** The corpus callosum from above. In this image the dark edges consist of grey matter, while the lighter areas represent white matter. The voids are filled with cerebrospinal fluid [56].

human *Corpus Callosum*, the bridge between the two halves of the brain. It consists of $\pm$ 160 million nerve fibers [4], facilitating communication between the left and right side of the brain.

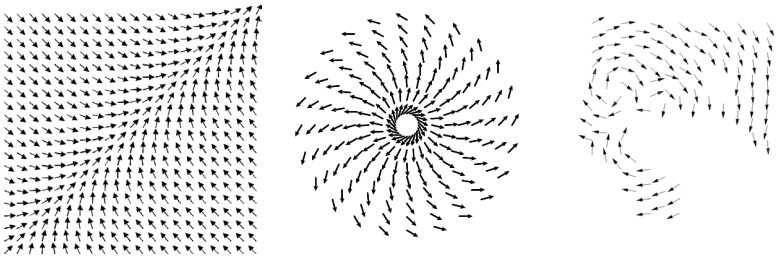
In this paper we will use white matter tracts as the dense line dataset. A simple visualization of this data was shown in Figure 2.1. We will take a closer look at how this data was generated in Chapter 5.

# 4

# VECTOR FIELD DATA

The main goal of the method we propose is to abstract dense line data. The concepts however are also applicable to vector field data.

Many kinds of real-life data can best be described with a field of vectors. In a number of points, direction and magnitude are described with a vector. The arrangement of the points is dependent on the overall structure of the data, as typically in areas of interest density is high. In the case of measured data, arrangement also depends on physically available measuring points.



**(a)** Regular square **(b)** Regular circular **(c)** Irregular

**Figure 4.1:** Vector fields with different arrangements of points.

Basic visualization of vector fields is often done with *hedgehog* visualization [47]. This visualization draws an arrow from each data point, with a direction and length corresponding to direction and magnitude of the respective vector.

Figure 4.1 shows several differently structured 2D vector fields, visualized with hedgehog visualization.

One of the fields that uses *vector field data* extensively is flow analysis, of which this chapter gives a short introduction.

## 4.1 FLOW

A large number of fields of science and technology are interested in the measurement or simulation of 2D or 3D *flow*. Liquid flow is a subject that has long interested scientists. Da Vinci for example conducted extensive studies of water flow around obstacles (shown in Figure 4.2).

In the artwork of Charles Sowers at the Randall Museum in San Fransisco, a direct visualization of a 2D plane of flow can be seen, quite similar to the hedgehog visualization. A wall filled with 612 wind vanes placed in a regular grid shows the air current along it. Figure 4.3 shows this installation, as well as a close up of the vanes.

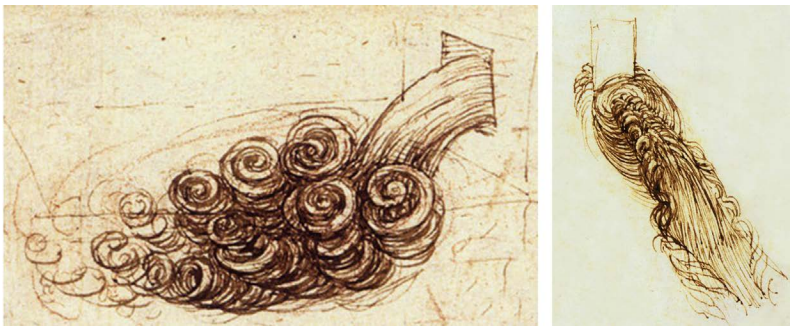


**Figure 4.2:** Leonardo da Vinci's studies of water flow around obstacles.

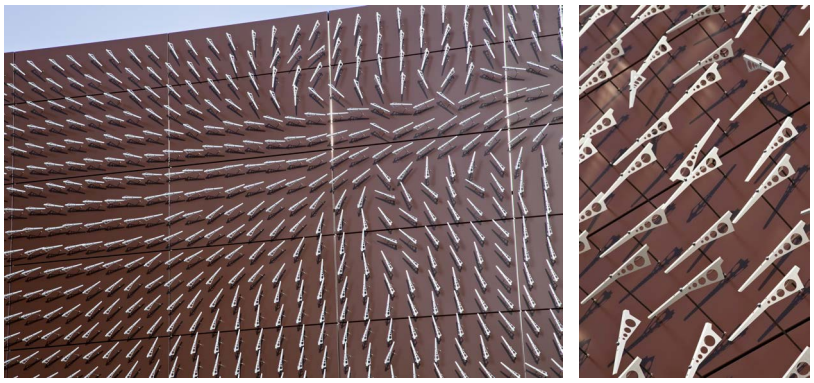
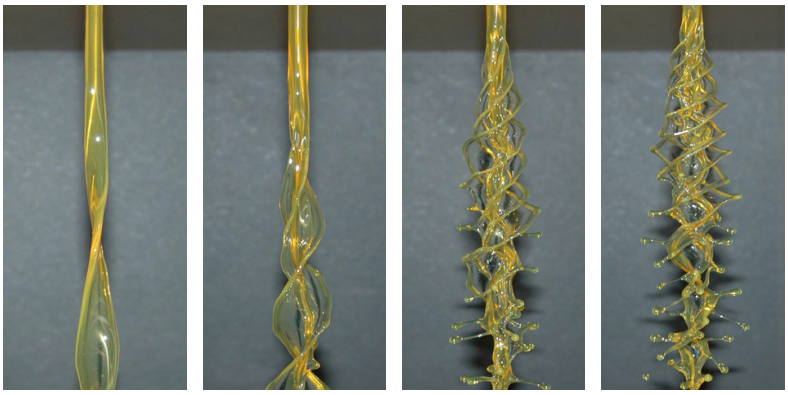


**Figure 4.3:** Charles Sowers' "Windswept" at the Randall Museum.

More complex flows, especially in 3D, are harder to measure directly without disturbing the flow. See for example the laminar and turbulent water flow in Figure 4.4. Inserting probes for measurement would disturb the flow, defeating the purpose of the experiment.

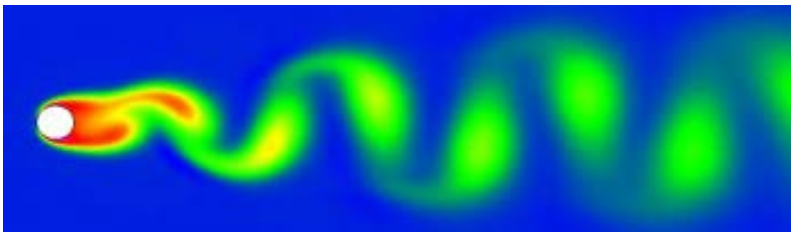
Since directly measuring disturbs the very thing we are measuring, a different approach is needed. Simulations using the *Navier-Stokes equations*, an application of Newton's second law to viscous flow, can be used to approximate the velocity, pressure and other values of interest in a flowing liquid [7].



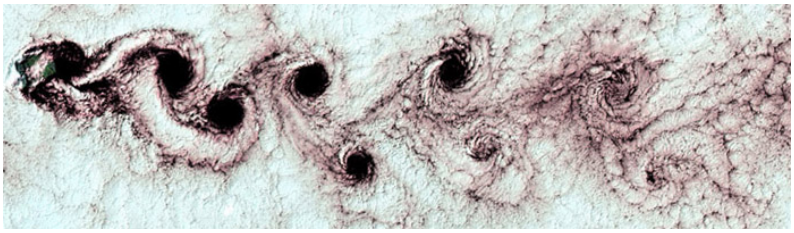
**(a)** **(b)** **(c)** **(d)**

**Figure 4.4:** Rotating fluid illustrating flow that is laminar (a) and turbulent (others) [32].

The Navier-Stokes equations enable us to make an accurate simulation of fluid flow without introducing disturbance through measuring. These simulations can be used in varying scales, from being used to model weather (see Figure 4.5) and ocean currents on large scale, to simulating the flow of blood in blood vessels.



**(a)** Kármán vortices, an effect caused by unsteady separation of flow around a surface mounted cylinder, approximated with the Navier-Stokes method [37].



**(b)** Kármán vortices over the Alexander Selkirk Island (South Pacific Ocean). The image shows an area of approx. 70 by 250 kilometers. Image by LandSat [40].

**Figure 4.5:** The Navier-Stokes method can be used to model large scale weather patterns.

As with dense line data, visualization of large vector fields is often difficult, due to the size and complexity of the data. The method we propose aims to simplify this process.

# 5 WHITE MATTER PATH TRACKING

As Isaac Newton remarked [41]:

*"If I have seen further it is by standing on the shoulders of giants."*

The development of *white matter path tracking* is built upon a multitude of techniques. While a full description of the history of the methods is beyond the scope of this writing, a short explanation of the methods directly involved is needed to understand the properties of the data.

Obviously direct visual in-vivo analysis of white matter fibers is not possible. To be able to gain insight in this structure, a combination of *diffusion tensor imaging* (DTI) to acquire data, and *fiber tracking* to show the data is used.

In this chapter we give a description of the physics and techniques needed to generate dense line data representing the white matter structure in the brain.

## 5.1 DATA ACQUISITION

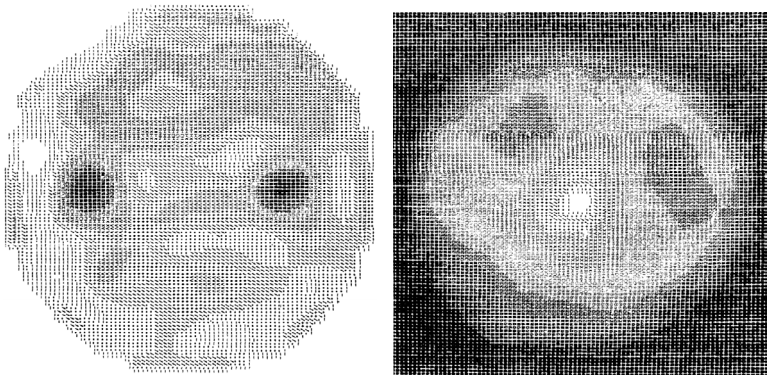
Using DTI, a technique based on *magnetic resonance imaging*, a map can be made of the direction of diffusion of water in the brain. This map is an indication of the direction that white matter fibers have in each region. This section will discuss the development of DTI.

### 5.1.1 *Magnetic Resonance Imaging*

First proposed in 1952 by Carr and Purcell [12], magnetic resonance imaging (MRI) is a technique that measures the local density of hydrogen (specifically common hydrogen, $^1$H or protium, in contrast to $^2$H, or deuterium) in solid bodies, such as biological tissue. Their proposed method however does not include a method of producing an image.

In 1973 Lauterbur et al. published a method of producing an image using MRI [33]. In 1974 they produced the first MRI image of a *living* organism [34], a mouse. These images can be seen in Figure 5.1a and 5.1b, respectively.

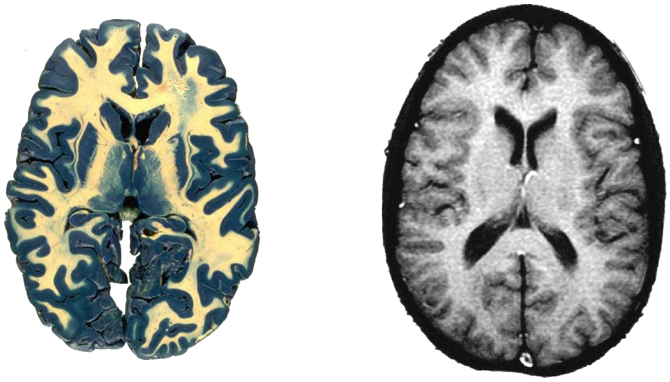
A field in which MRI is especially useful is exploring the brain. Previous techniques such as X-rays or CT scans are ill suited for this task. The brain consists of materials that differ very little in the amount of radiation they absorb, so very little contrast is seen on a scan. Furthermore, these techniques use ionizing Röntgen radiation, which can damage DNA.



(a) (b)

**Figure 5.1:** Two of the first images captured using MRI:
(a): two cylinders containing $^1H_2O$ (dark), enclosed by a cylinder filled with $^1H_2O$ and $^2H_2O$ (light),
(b): thoracic cavity of a mouse (transverse).

An MRI scan does not use ionizing radiation and can make a distinction between very similar materials, making it extremely suitable for imaging the brain. The results of such a scan is shown in Figure 5.2b.



**(a)** A photograph. **(b)** An MRI scan.

**Figure 5.2:** Two views of a slice of a brain [43].

In this figure the different components of the brain are discernible, with the grey matter shown in a dark shade (resp. blue and dark gray), and the white matter visible in a light shade (resp. white and light gray). The CSF is shown in black. The MRI image was made first, whereafter the head was dissected, sliced and photographed.

While MRI allows us to do in-vivo analysis of the composition of various parts of the brain, there is no information about the actual fibrous structure of the white matter. This information can be discerned from the diffusion of water in the brain.

### 5.1.2 *Diffusion*

Particles in a homogeneous fluid environment will randomly move around (diffuse) due to *Brownian motion* [19], with for each

molecule typically $10^{20}$ random steps per second. White matter however is not homogeneous, but consists of a large number of aligned axons, the connections between neurons. A detailed illustration of a neuron is shown in Figure 5.3.

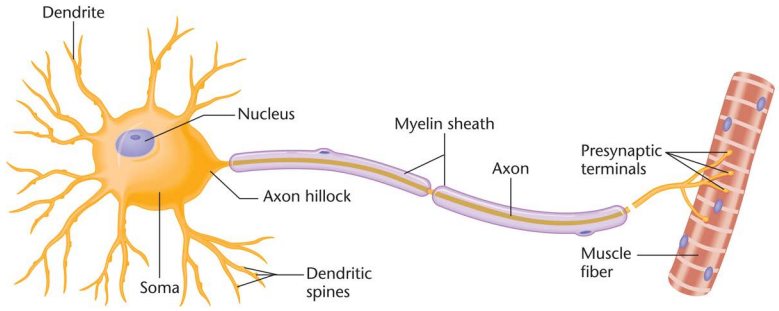



**Figure 5.3:** A neuron whose soma is connected to a muscle fiber via an axon [30].

Axons are basically fatty tubes filled with and surrounded by water. The *myelin sheath* runs along most of the surface of the axon, and functions as electrical insulation. It has a beneficial effect on the transmission speed of signals. The sheath consists mostly of fats, which hinder water diffusion both inside and in the fluid around the axons.

In the brain white matter runs in bundles of often thousands of roughly parallel axons over some distance, as can be seen in Figure 5.4. The brain in this figure has been dissected in order to show the structure of the corpus callosum.

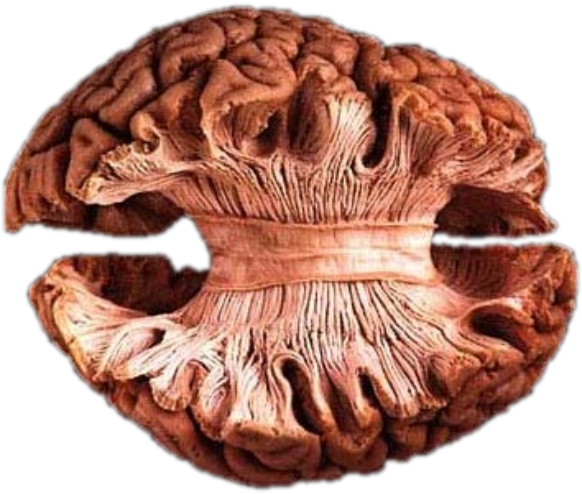
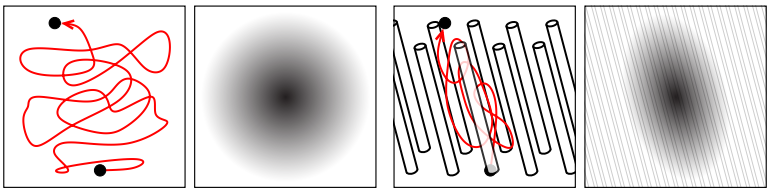


**Figure 5.4:** A post-mortem brain prepared to show the white matter structure of the corpus callosum.

White matter in the brain thus consists of bundles of reasonably parallel fibers, which obstruct the diffusion of water in any non-longtitudinal direction. Figure 5.5 shows the effects these fibers have on the diffusion of water.

*Isotropic* or directionally independent diffusion, shown in Figure 5.5a, occurs when no obstructions are present. Brownian motion occurs equally in all directions, so a drop of ink in water will eventually become a circular stain.



**(a)** Isotropic **(b)** Anisotropic

**Figure 5.5:** Movement of a particle (left) and spread of a drop of ink (right) in water. (a) unhindred, (b) hindred.

*Anisotropic* or directionally dependent diffusion, as shown in Figure 5.5b, will occur when some structure is present in the fluid that restricts diffusion based on direction, such as a large number of densely packed axons. The ink stain will diffuse faster in the longitudinal direction, and slower in the perpendicular direction, resulting in a distorted inkstain.

For brain tissue this means that areas that consist of mostly aligned axons will have a larger diffusion speed along the axons than in other directions [13], and will therefore have diffusion similar to Figure 5.5b.

### 5.1.3 *Diffusion MRI*

Proposed in 1985 by Le Bihan et al. [36], *diffusion* MRI (dMRI) was initially developed to closer study tumors in the liver. By modulating the magnetic field used in the MRI scan, the scan is made sensitive to the diffusion of water. Initially this method was used to measure the amount of diffusion in any direction, in order to detect areas with limited diffusion, thought to indicate tumorous tissue in the liver.

Disappointing results however quickly led to broader research, eventually yielding favorable results in brain scans. In 1991 it was first used by Douek et al. to measure the direction of diffusion. Figure 5.6 shows the results of their work.

In this figure a clear distinction can be made between fiber paths that have a strong up-down or left-right orientation, and those that do not.

### 5.1.4 *Diffusion Tensor Imaging*

Applying the technique of dMRI in multiple directions yields *diffusion tensor imaging* (DTI). Enough information becomes available to not only detect diffusion in the imaging plane (the plane through which the image is taken), but also to calculate diffusion as a second-order tensor [9]. A tensor is a generalization of a vector (which is a first-order tensor). A tensor describes relations between vectors.

A second-order tensor has three eigenvectors representing the main diffusion directions, and three eigenvalues representing

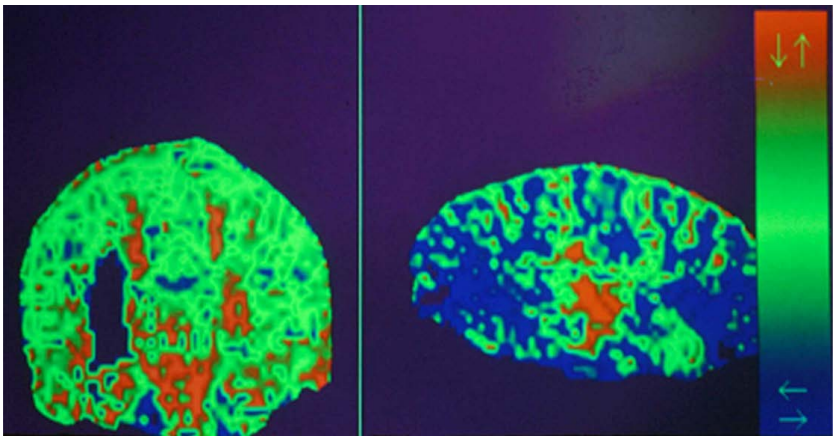


**Figure 5.6:** DMRI image colored according to the direcion of diffusion [17]. Clear left-right or up-down diffusion is shown in blue and red, diagonal or homogenious diffusion is shown in green.

the strength of diffusion in that direction. This can be visualized in an ellipsoid, as shown in Figure 5.7.

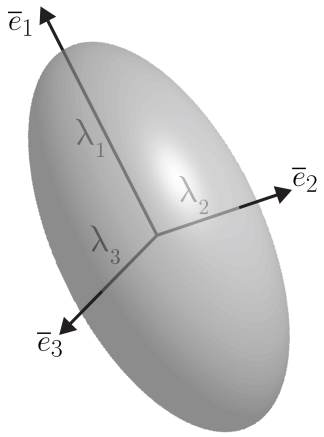
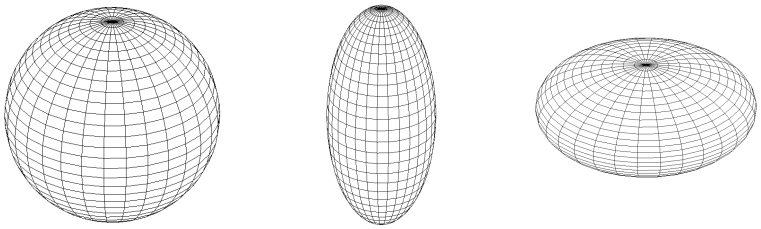



**Figure 5.7:** A tensor visualized as an ellipsoid with eigenvectors $\bar{e}_{1..3}$ and eigenvalues $\lambda_{1..3}$ [18].

This visualization is quite similar to the ink drop mentioned in Figure 5.5. When diffusion in all directions is equal, the ellipsoid will be a sphere. When diffusion is more pronounced in one direction it will become more elongated in that direction. These situations are shown in Figure 5.8.

Diffusion in grey matter would be represented by the sphere of Figure 5.8a, and in white matter by the elongated ellipsoid (or *prolate* ellipsoid) in Figure 5.8b. The disc shaped ellipsoid (or *oblate* ellipsoid) in Figure 5.8c would indicate diffusion in a plane, which could for example result from two crossing fiber bundles.

This way of visualizing diffusion can be used to show structure in the brain. In Figure 5.9 a slice of DTI data is visualized this way. The tensors are represented by ellipsoids, while the brightness of the background is determined by *fractional anisotropy*. Fractional Anisotropy (FA) [27] is a measure of how anisotropic a tensor is, defined as:



**(a)** $\lambda_1 \approx \lambda_2 \approx \lambda_3$ (Sphere) **(b)** $\lambda_1 > \lambda_2 \approx \lambda_3$ (Prolate) **(c)** $\lambda_1 < \lambda_2 \approx \lambda_3$ (Oblate)

**Figure 5.8:** Visualisation of tensors with ellipsoids.

$$FA = \sqrt{\frac{3}{2}} \cdot \frac{\sqrt{(\lambda_1 - \overline{\lambda})^2 + (\lambda_2 - \overline{\lambda})^2 + (\lambda_3 - \overline{\lambda})^2}}{\sqrt{{\lambda_1}^2 + {\lambda_2}^2 + {\lambda_3}^2}},$$

where

$$\overline{\lambda} = \frac{\lambda_1 + \lambda_2 + \lambda_3}{3}.$$

The result is a number from 0.0 to 1.0, where an isotropic tensor yields 0.0, and a completely anisotropic tensor yields 1.0.

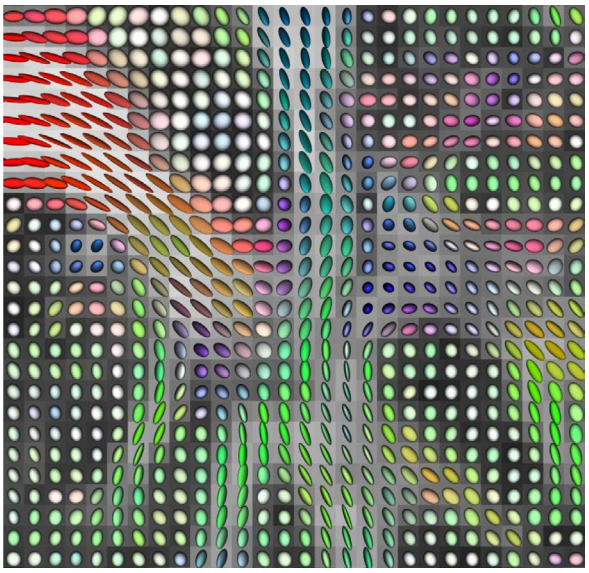


**Figure 5.9:** Ellipsoids representing a slice of DTI data. Ellipsoids are colored according to main direction of anisotropy. Background brightness is determined by FA [57].

While this way of visualizing gives a good sense of the data, it is too fine-grained to be suitable for large datasets, especially since a large amount of occlusion occurs when a 3D volume is visualized. Furthermore, it doesn't show the fibrous structure.

An in-depth history and validation of DTI and its interpretation can be found in "Validation methods for diffusion weighted magnetic resonance imaging in brain white matter" by Fieremans, E. [22].

While visualizing the entire brain volume is conceptually interesting, it soon runs into the problem of being cluttered and uninformative. Usually the information of interest is not the entire field, but the structure of white matter, the bundles of fibers.

It is therefore better for the structural insight to not display the entire field, but to create curves that represent the tracts. The technique that is used to calculate these tracts is called *fiber tracking* or *tractography*. Tractography uses the tensors produced by DTI to approximate the actual nerve fibers in the brain.

This section describes several methods used in tractography. In the next section we use the following definitions:

- The tract starts in seed point $\mathbf{p}_0$ with a direction $\overline{v}_0$.
- $\mathbf{p}_i$ has a current tract direction $\overline{v}_{\text{in}}$, and a next tract direction $\overline{v}_{\text{out}}$
- From $\mathbf{p}_0$ a number of points $\mathbf{p}_1, \ldots, \mathbf{p}_n$ are calculated to create a path. Point $\mathbf{p}_n$ lies a predetermined distance from $\mathbf{p}_{n-1}$, in the direction $\overline{v}_{\text{out}}$ of $\mathbf{p}_{n-1}$.
- The current area (*voxel*) the tract is running in is described by a tensor $\underline{\mathbf{D}}$, with eigenvectors $\overline{e_1}, \overline{e_2}, \overline{e_3}$ with corresponding eigenvalues $\lambda_1, \lambda_2, \lambda_3$, where $\lambda_1 \geq \lambda_2 \geq \lambda_3$.
- The value of $\overline{v}_{\text{out}}$ of point $\mathbf{p}_{n-1}$ is the same as $\overline{v}_{\text{in}}$ of $\mathbf{p}_n$.

The seed point and direction are assumed to be known. These can be calculated using for example the methods described in "DTI Visualization with Streamsurfaces and Evenly-Spaced Volume Seeding" by A. Vilanova et al. [52].

### 5.2.1 *Streamline Tracking*

In its most basic form fiber tracking only uses the eigenvector of the tensor that has the largest eigenvalue: $\overline{e}_1$ (the *major eigenvector*). This reduces a field of tensors to a field of vectors [16, 39]. *Streamline tracking* (ST) can then be used. For ST, each step of tracing $\overline{v}_{out}$ is determined as:

$$\overline{v}_{out} = \overline{e}_1$$

This approach however has the drawback that a nearly isotropic tensor will have a very unstable direction. This issue is illustrated in Figure 5.10, where two similar tensors have very different largest eigenvectors.

This issue can be partially addressed by taking the FA into account as a measure of reliability. Many integration methods stop integrating when the FA falls below a threshold value [24]. While this resolves the problem in 2D, the problem is still present in 3D. A tensor with $\lambda_1 = 0$, $\lambda_2 = 1.00$ and $\lambda_3 = 0.99$ has an FA $> 0.7$, which is higher than a reasonable threshold, while this tensor is

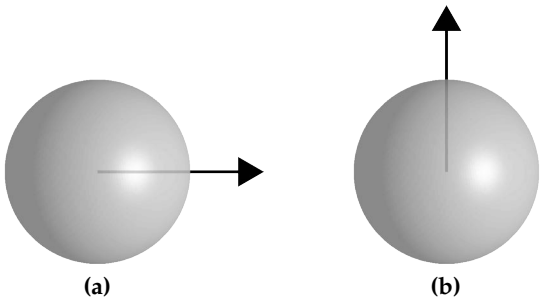


**Figure 5.10:** Isotropic tensors with noise, yielding very different results when only taking the largest eigenvector.
(a): $\lambda_1 = 1.00, \lambda_2 = 0.99, \lambda_3 = 0.99, \bar{e}_1 = \langle 1, 0, 0 \rangle$
(b): $\lambda_1 = 0.99, \lambda_2 = 1.00, \lambda_3 = 0.99, \bar{e}_1 = \langle 0, 1, 0 \rangle$

very sensitive to noise. Decreasing $\lambda_2$ by just 0.02 results in a value for $\bar{e}_1$ that is orthogonal to its original value.

This situation is quite common in white matter, as it occurs whenever two fiber bundles cross. Therefore this method is ill-suited for white matter tractography.

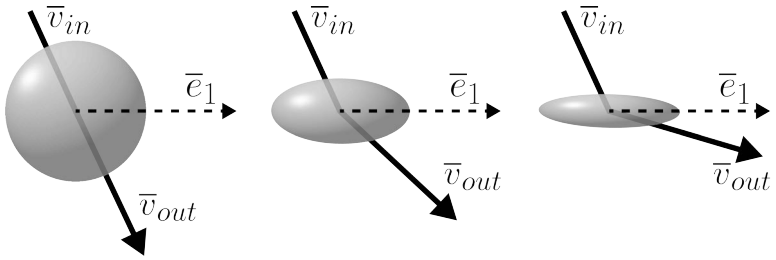
### 5.2.2 *Tensor Deflection*

A method was proposed to improve this behavior, termed *tensor deflection* (TEND) [55]. This method aims to eliminate the noise sensitivity of streamline tracking. To achieve this, tensorlines take into account not just the major eigenvector, but the entire tensor.

$\overline{v}_{out}$ is therefore specified as:

$$\overline{v}_{out} = \underline{\mathbf{D}} \cdot \overline{v}_{in}$$

How $\overline{v}_{out}$ is affected by varying degrees of (an)isotropy is shown in Figure 5.11. Here all tensors are cylindrically anisotropic, meaning at least two eigenvalues are equal.




**(a)** $\lambda_1 \approx \lambda_2 = \lambda_3$ **(b)** $\lambda_1 > \lambda_2 = \lambda_3$ **(c)** $\lambda_1 \gg \lambda_2 = \lambda_3$

**Figure 5.11:** TEND with different amount of anisotropy.

The improvement of TEND over ST can clearly be seen. In all the shown situations, ST would have followed the direction of $\bar{e}_1$, where TEND shows a more intuitive trajectory.

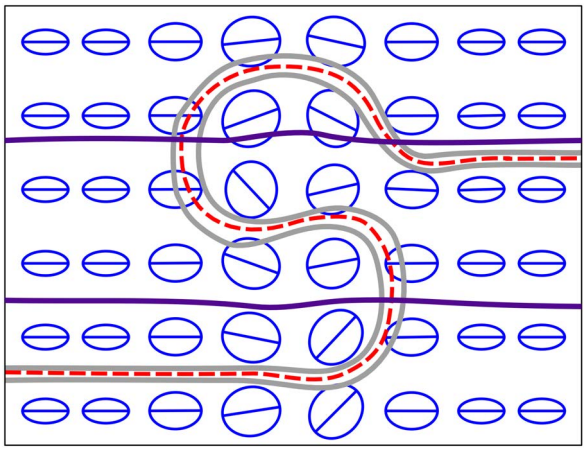


**Figure 5.12:** ST (red) and TEND (purple) results on a 2D tensor set (blue).

When comparing ST to TEND, it can be observed that ST has greater ability to run across crossing fiber bundles without its direction being influenced [35].

A comparison of the ST and TEND methods can be seen in Figure 5.12. The dataset used has a band of more isotropic tensors in the center. TEND clearly proves to be more robust than ST to small sections of isotropy.

### 5.2.3 *Tensorlines*

Tensorlines (TL), as proposed by Weinstein et al. [55] are a combination of ST and TEND, with an added linear component. Each component is weighed by two user-specified parameters:

$$\overline{v}_{out} = f \cdot \overline{e}_1 + (1-f) \cdot (g \cdot \underline{\mathbf{D}} \cdot \overline{v}_{in} + (1-g) \cdot \overline{v}_{in})$$

Three different influences can be determined in this function:

| method | | weight |
|---|---|---|
| ST | $(\overline{e}_1)$ | $f$ |
| TEND | $(\underline{\mathbf{D}} \cdot \overline{v}_{in})$ | $(1-f) \cdot g$ |
| linear | $(\overline{v}_{in})$ | $(1-f) \cdot (1-g)$ |

These influences are weighed according to the value of $f$ and $g$. If $f$ = 1.0, TL is equivalent to ST. If $f$ = 0.0 and $g$ = 1.0, TL is equivalent to TEND. If both $f$ and $g$ are 1.0 the tract will be straight, in the direction of $\overline{v}_{\text{out}}$ of $\mathbf{p}_0$. A value of $f$ and $g$ between 0.0 and 1.0 is used to change the properties of TL to suit the dataset.

### 5.2.4 *Beyond DTI*

While DTI based tractography yields interesting insights in the data, there are many situations where this approach runs into problems. Crossing fiber bundles cause lowered FA, causing either false negatives (prematurely terminating tracing) or false positives (continuing tracing along a different tract) depending on setting the stop criterion either too low or too high. With the development of *high angular resolution diffusion imaging* [5], the diffusion data has become of such high quality that new methods have become feasible.

For example, *(constrained) spherical deconvolution* [26] can be used to produce traces that are more resilient to noise and crossing fibers than DTI, as can be seen in Figure 5.13.

These techniques all however result in dense line data, which means our technique is still applicable.

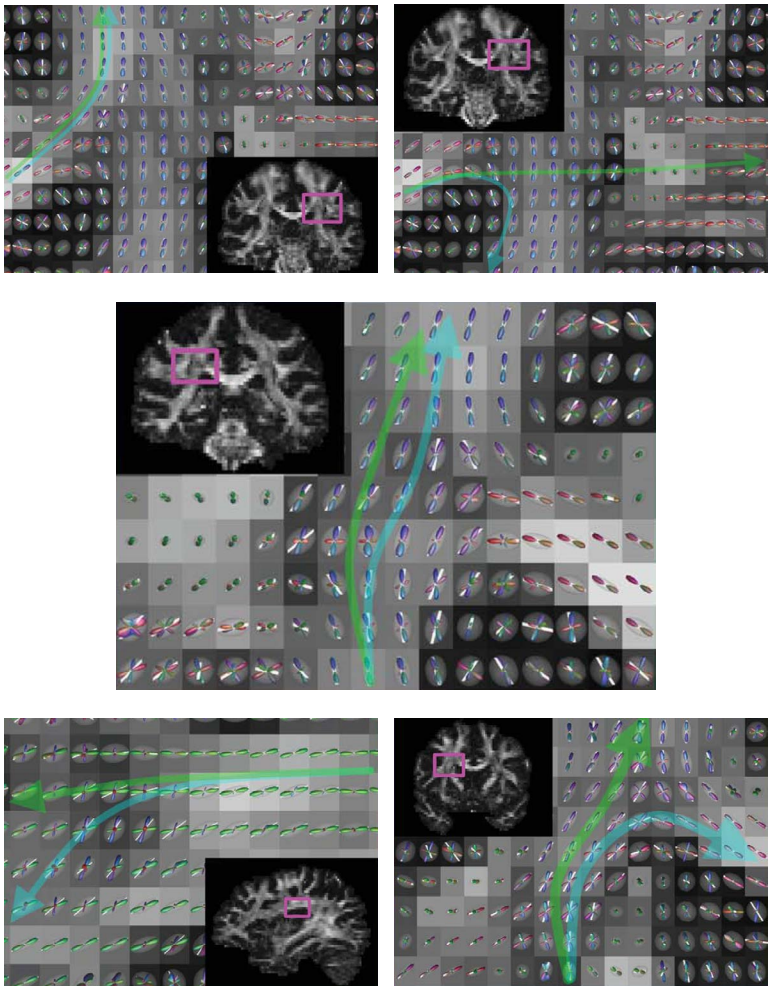


**Figure 5.13:** Results of constrained spherical deconvolution tracing (green) vs. DTI tracing (blue) [26]. Also shown are FA (as the brightness of the background), DTI ellipsoids (semitransparent), $\bar{e}_1$ (white lines) and constrained spherical deconvolution results (multicolored).

# 6 VISUALIZATION

In previous chapters basic visualizations of lines, vectors and tensors have been used. A line was represented by a tube of constant thickness, a vector by an arrow, and tensors by ellipsoids. While these methods have the benefit of being very easy to implement, for larger datasets they quickly become cluttered and uninsightful. They also only show the basic element (line, vector or tensor), while a dataset may contain many more values. This chapter will discuss more advanced visualization techniques for such datasets.

## 6.1 WHOLE DATASET VISUALIZATION

Usually data is regarded as a precious resource, and discarding data is considered undesirable. Therefore many visualization techniques try to show the entirety of the dataset. While this can work in some cases, especially in 2D, with more complex datasets visualization is quite difficult. This section shows some of the more prevalent techniques, as well as issues that arise in using them.

### 6.1.1 *Regular Grid Based*

Some visualization methods display one object per sample point in the data, where the sample points are determined by a regular grid. These objects are usually constrained in size, in order to prevent occlusion between objects in the visualization.

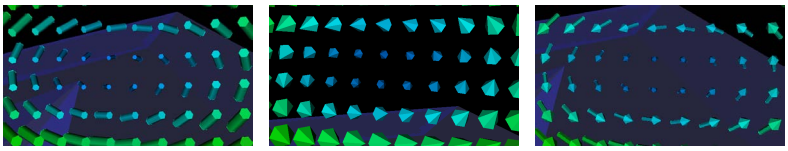
#### *Vector Glyphs*

The hedgehog visualization used earlier falls in the category *vector glyphs*. This method associates each point with a shape, which is then scaled to represent some number of encoded attributes (such as pressure, temperature, etc). Hedgehogs are the result of the most basic glyph that can be used for this: the line.

In a 3D dataset, hedgehogs quickly become unclear. To clearer hint at orientation, the 3D glyphs can be used, which enables us to use shading. Figure 6.1 shows three possible glyph shapes.

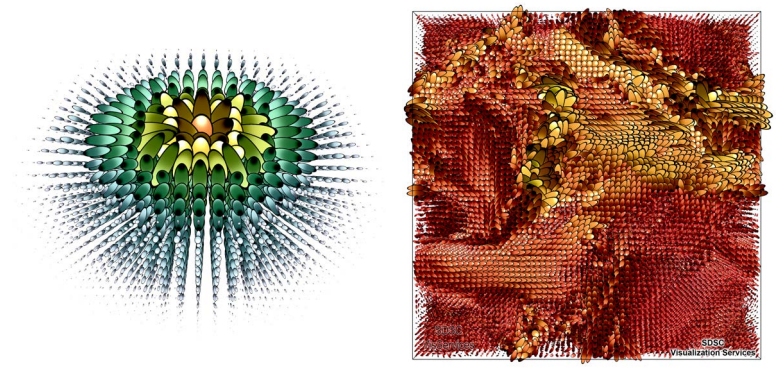
In Figure 6.1a cylindrical glyphs are shown, which are closely related to hedgehog visualization. More expressive glyphs showing the direction of the vector are shown in (b) and (c).

A more complex visualization can be used to more clearly show encoded data, such as in Figure 6.2.



**(a)** Cylindrical **(b)** Conical **(c)** 3D arrow

**Figure 6.1:** 3D glyphs used to visualize a dataset. Images courtesy of R.Rossi [44].




**(a)** Forces in an earthquake **(b)** Magnetic field

**Figure 6.2:** GlyphSea visualization. Images by SDSC Visualization Services [38].

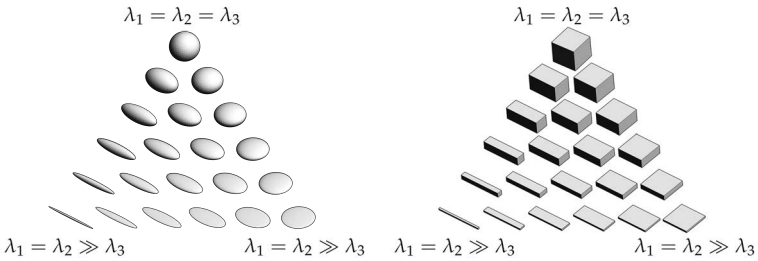
These figures show the *GlyphSea* method [38], where each vector is shown as an ellipse. The hue of the ellipse is determined by the magnitude of the vector, while the lightness is based on the direction of the vector with regard to the ellipse. This has the effect of making the ellipse white in the direction of the vector, and black in the opposite direction.

### *Tensor Glyphs*

Glyphs can not only enhance the visualization of a vector field. As a glyph can easily be made three-dimensional, changing its shape can be used to show data that varies in three dimensions (Note that this does not mean data that is specified in three dimensions, but rather data of which each data point has three dimensions). This makes them suitable to visualize tensors. We simply show the three eigenvectors as the dimensions of the glyph.

We have already seen a basic tensor glyph in earlier chapters, the ellipsoid. The three axes of the ellipsoid were used to encode the three eigenvectors of the tensor. Several tensor glyphs are shown in Figure 6.3, along with the original ellipsoid.

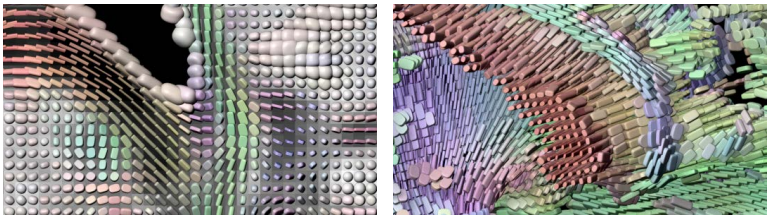
Tensor glyphs can be used to visualize tensor fields in both 2D and 3D, as can be seen in Figure 6.4.




**(a)** Ellipsoid **(b)** Cuboid

**(c)** Cylindric **(d)** Superquadratic

**Figure 6.3:** Various tensor glyphs for various tensors [31].



**(a)** 2D **(b)** 3D

**Figure 6.4:** Visualization of tensor data using superquadratic tensor glyphs [31].

*Streamlines*

In a vector field, a curve that is locally tangential to the field can be drawn. This line is called the *streamline*. The trajectory of a streamline corresponds with an integration path over the field. This method is shown in Figure 6.5.

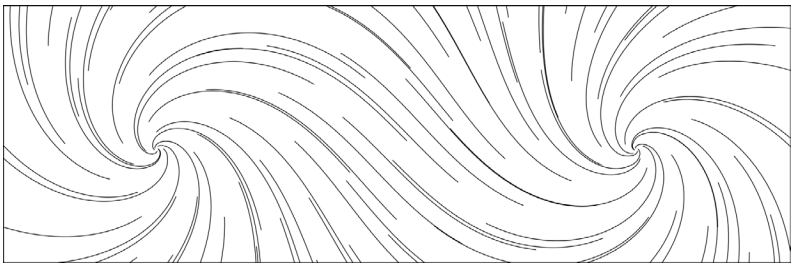


**Figure 6.5:** Regular streamlines [28].

The length of streamlines is usually limited. If the goal of the visualization is showing the direction of the vector field, streamlines are usually given a constant length. If on the other hand the magnitude of the vectors is of greater interest, streamline length can be varied.

Methods that use *texture* or *image based flow visualization* (IBFV) to visualize a vector field take an image and deform it based on the direction and magnitude of the field. Implementations differ in two main ways:

Firstly on what sort of texture is used. Figure 6.6a and b show the difference between using a regular texture and a noise texture.

Secondly on whether the texture is continuously injected and blended, or only introduced once. Figure 6.6c and d illustrate these two methods. When using continuous injection it is necessary to apply a decay function on the texture, as otherwise the entire image would eventually be covered. With one-time injection this is not an issue.

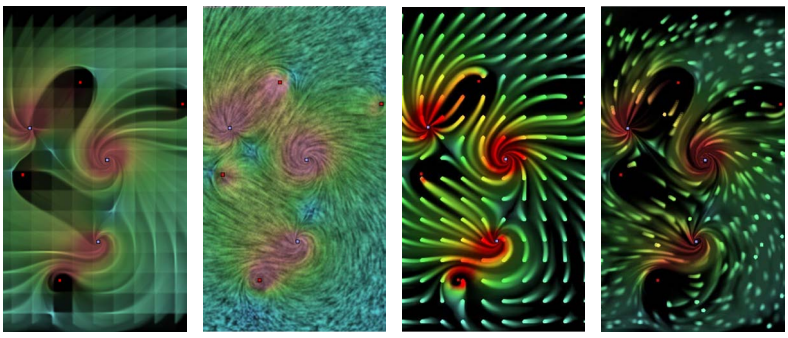



**Figure 6.6:** Texture based flow visualization [51]. (a) Regular texture, (b) Noise texture, (c) Continuous injection, (d) One-time injection

.

An important difference between IBFV and the methods previously discussed, is the addition of an extra dimension, i.e. the number of warping steps applied. With a small number of steps the visualization only shows very large magnitude areas, whereas small magnitude areas become visible with a large number of steps. Visualization using IBFV can easily be animated by mapping the number of steps to the elapsed time.

For 3D data an opacity-based version of this method can be used, as shown in Figure 6.7.

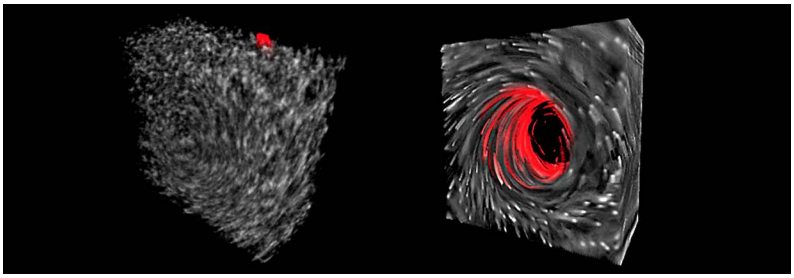


**Figure 6.7:** 3D IBFV.

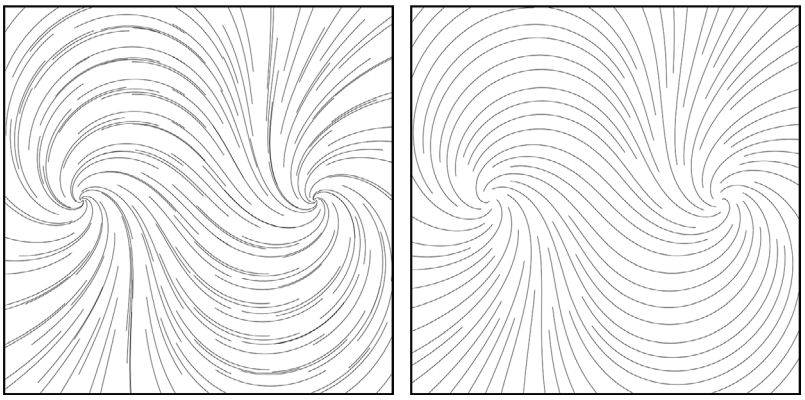
### 6.1.3 *Seed Based*

Some methods, while visualizing the entire dataset, choose to let the sample points depend on the data instead of a predefined grid. While these methods are therefore more flexible, care must be taken that the seed-selection method selects sample points that are representative.

#### *Streamlines*

Similar to the grid based version, seed based streamlines show the direction of the field. Due to the dynamic selection of seed points however, artifacts caused by seed point location are much less likely. When path length is made dependent on other paths instead of the magnitude of the field, a clear depiction can be made of the direction of the field with the method described by Jobart et al. [28].

Figure 6.8a shows a larger view of the same visualization as in Figure 6.5. In Figure 6.8b the Jobart method is applied to the same dataset. The reduction of visual artifacts can clearly be seen.



**(a)** Regular grid **(b)** Seed based

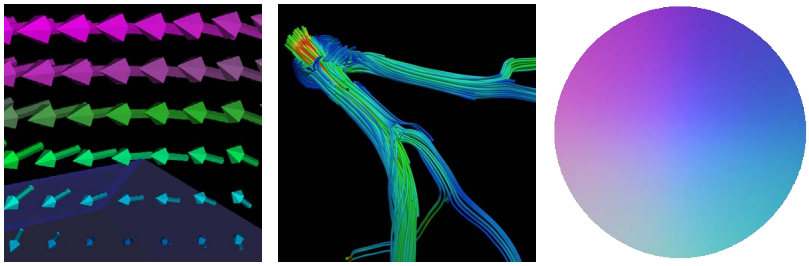
**Figure 6.8:** Streamlines drawn from sample points determined by two different methods [28].

## 6.2 ENCODED INFORMATION

The visualizations shown in the previous chapters often show more than just the field. Some images for example use coloring to indicate the magnitude of the field. Properties like this are called *encoded information*. This information can be something that is derivable from the data, such as magnitude or direction (*implicit*), or something such as temperature or pressure, which is supplied as extra data (*explicit*).

### 6.2.1 *Color*

As shown in Figure 6.4 and 6.6, color can be used to represent various types of encoded information (major eigenvector direction and magnitude of flow, repsectively). Color is usually used to represent either a 1D scalar mapped to a color map, or to represent a 2 or 3D vector, representing X, Y and Z. Figure 6.9 shows some applications of color mapping.



(a) (b) (c)

**Figure 6.9:** Color maps of various types of encoded information.
(a) Mapping the magnitude of a vector to blue-red
(b) Mapping FA along a streamline to blue-red
(c) Mapping the X, Y and Z component of the normal vector of a sphere to red, green and blue.

### 6.2.2 *Streamtubes*

Another way 1D encoded information can be visualized along a streamline is by converting the line to a tube. The width of the tube can then be used to display one dimension of encoded information. In Figure 6.10 this is used to show vector magnitude along a streamline [48].

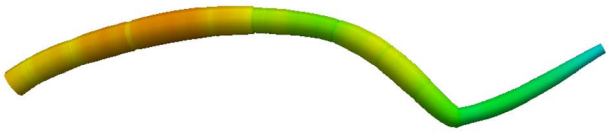


**Figure 6.10:** Streamtube with width and color by vector magnitude.

### 6.2.3 *Hyperstreamlines*

Streamtubes can be combined with the concept underlying tensor glyphs to form *hyperstreamlines* [48]. Assuming the streamline coincides with the major eigenvector, the two minor eigenvectors can be displayed using the shape of the tube, essentially turning the cross-section of the tube into an ellipse with axes $e_2$ and $e_3$. This method is shown in Figure 6.11.

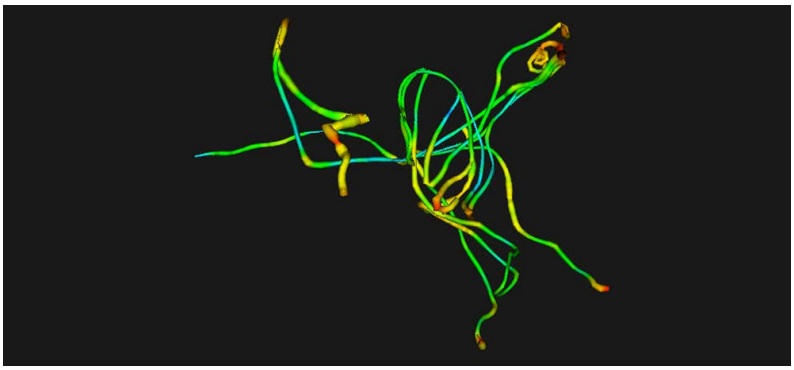


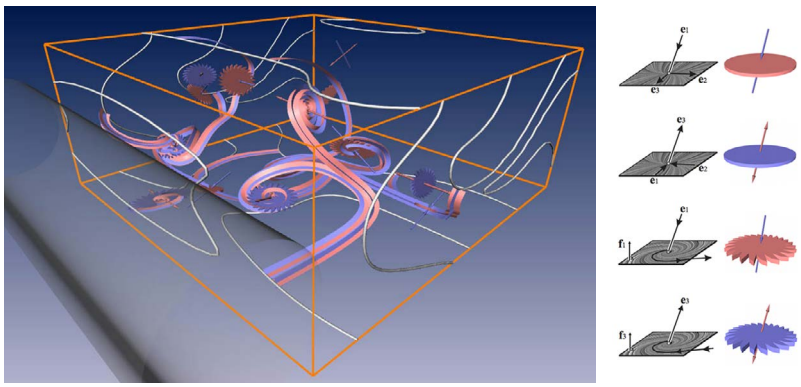
**Figure 6.11:** Hyperstreamline with width by minor eigenvectors, and color by FA [6].

## 6.3 ABSTRACTION

While visualizing the entire dataset can be useful, these visualizations quickly run into severe issues as the dataset grows. To resolve these problems, various methods have been developed that represent the *essence* of the information in the dataset.

### 6.3.1 *Topological Visualization*

The aim of a *topological visualization* is to show areas that are 'interesting'. In these areas some condition is true, such as strong turbulence, or the magnitude having a certain value. An example of this for vector field data can be found in the *boundary switch connector* method. This method (a result of which is shown in Figure 6.12) extracts boundaries of flow behavior, and displays them as ribbons. Saddle points are shown with disks and flowers, shown in Figure 6.12b.



**(a)** **(b)**

**Figure 6.12:** Boundary switch connector method applied to a 3D flow. Flow boundaries are shown as ribbons, saddle points with glyphs as shown in (b) [54].

A different topological visualization uses *isosurfaces* to show areas where a certain value is constant. Figure 6.13 shows a tractography of the brain. To provide context the visualization includes an isosurface, showing where FA is constant.

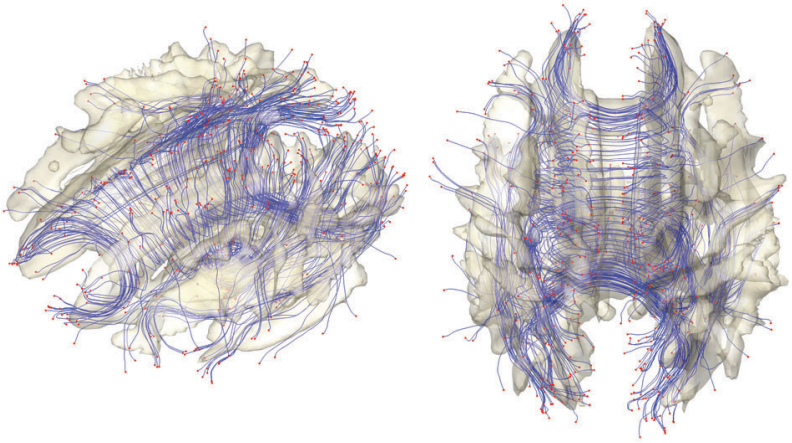
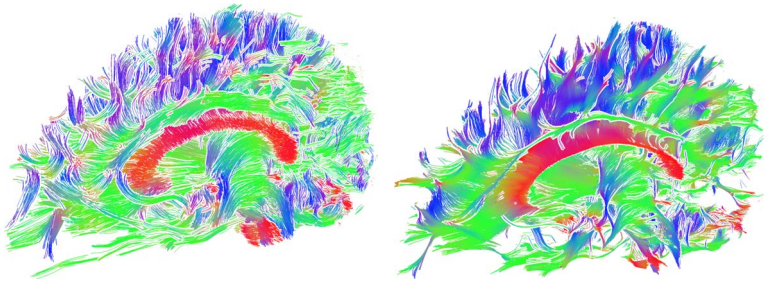


**Figure 6.13:** DTI fiber tracts (blue, with red endpoints) and FA isosurface [14].

### 6.3.2 *Bundling and Clustering*

As an hybrid between abstraction and whole dataset visualization, *bundling* takes a dense line dataset, and searches for lines that are locally similar. It then moves these lines closer together, creating bundles. The bundling of a large number of lines reduces occlusion. This method, as applied to DTI tractography data, is shown in Figure 6.14.



**(a)** Unprocessed **(b)** Bundled

**Figure 6.14:** DTI fiber tracts shown with and without bundling [20].

When we take bundling to extremes, we reach the *clustering* method. Clustering takes the locally similar lines, and replaces them with one line, effectively bundling them so tight all similar lines coincide.

## 6.4 ISSUES

While the methods described in this chapter achieve their goal of visualizing the datasets quite well, they suffer from various drawbacks, which will be discussed in this section.

### 6.4.1 *Occlusion*

In many cases a 3D use of the methods results in heavy occlusion. See for example the tensor glyph visualization in Figure 6.15. Any data beyond the surface is occluded by the ellipsoids at the surface, and at the right side intersecting ellipsoids even obscure the surface data.

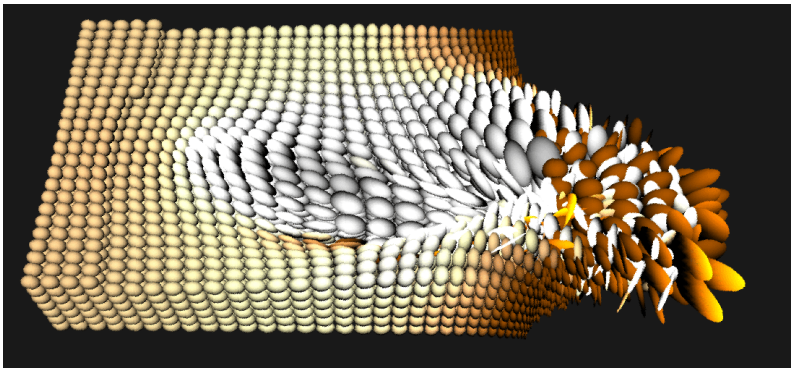


**Figure 6.15:** Tensor glyph visualization of a brain tumor [25].

This issue can be partially addressed by bundling, but excessive bundling can distort the dataset too much to be of any use, while too little bundling does not resolve the occlusion problem.

### 6.4.2 *Artifacts*

When a visualization method is used that seems to show data that is not in the dataset, it is said to contain artifacts. In 3D this problem can easily occur in whole dataset visualization with a regular grid. Depending on the view angle, long diagonal lines, artifacts of the regular grid, streak the image, seemingly showing a flow along that diagonal. This effect (along with occlusion) can be seen in the vector field visualization in Figure 6.16

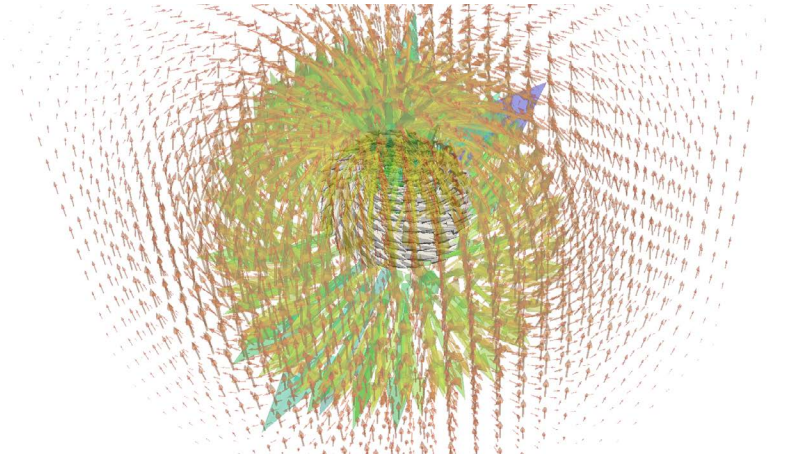


**Figure 6.16:** Vector field showing the magnetic field around a rotating charge. Note occlusion in the center and artifacts on the left and right side of the image [49].

### 6.4.3 *Contradiction*

Some visualizations contain hard to interpret or contradicting elements. The visualization of flow using boundary switch connectors (Figure 6.12) requires a large amount of interpretation by trained eyes to extract useful data. While the visualization may be very useful to a professional, it is useless for a novice.

A more direct instance of contradiction can be seen in tensor glyph visualization. With this method (shown in Figure 6.4) a tensor with a low FA value will be shown as a larger glyph than one with a high FA value, while isotropic areas are usually less interesting than anisotropic areas.

Any difference between the common sense interpretation of a visualization and the desired interpretation can cause confusion, and increases the amount of training needed to interpret the visualization. As this difference increases, the visualization becomes less useful.

## 6.5 CONCLUSION

Many methods can be used to visualize dense line or vector field data, but each comes with specific drawbacks. As has been shown, room for improvement exists in the field of abstraction. In the following chapters, we propose a hybrid visualization inspired by seeded streamlines and bundling that addresses some of the mentioned issues.

# 7

# ABSTRACTING

In this chapter we will outline the goal of our method, as well as introduce a number of terms and concepts that help achieve this goal. The following chapters will go into further detail, formalizing and expanding on the ideas introduced here.

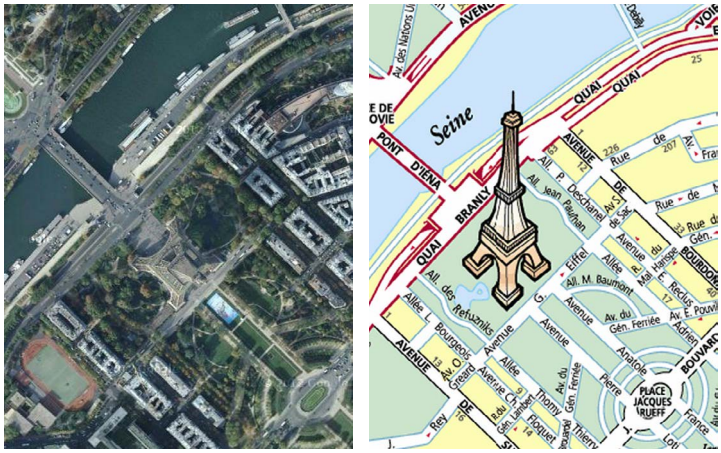
As seen in the previous chapter, the visualization of DTI data suffers from various hard to solve problems.

DTI data is usually visualized using one of two methods. Either the raw DTI tensors are shown using tensor glyphs, or the data is fiber tracked and the resulting fibers are shown. Both of these methods suffer from occlusion and high visual complexity.

The method proposed in this paper aims to address these problems by providing a very local view of the data (using e. g. tensor glyphs or fiber tracts), while presenting context using an abstracted view.

Our goal is to show an abstracted visualization of the dense line data produced by fiber tracking, and to use this abstracted visualization as overview or as context. The aim is not to create a visualization that replaces the fiber tracked data, but rather one that acts as a map for smaller subsets of data.

See for example the images in Figure 7.1. Here the Eiffel tower is shown, using two different methods. Figure 7.1a shows the tower and its surroundings at a constant level of detail, similar to the whole dataset visualization methods we have seen earlier.




**(a)** **(b)**

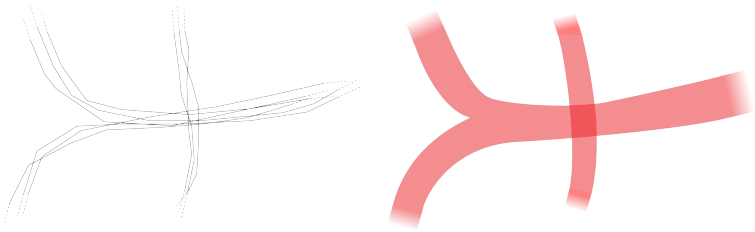
**Figure 7.1:** Looking at the Eiffel tower using (a): a fixed level of detail, or (b): variable level of detail.

In Figure 7.1b however, the Eiffel tower has been selected as a point of interest, and the visualization has been adapted as such. The tower is highlighted, and shown at a larger scale and in more detail than the rest of the map.

Our method is intended as the abstracted view of the data. The high level of detail view can be very simple (e. g. plain lines, or a hedgehog visualization), as it only shows a small dataset.

This abstracted view is constructed using the dense line data resulting from fiber tracking a DTI dataset. A fiber tracked dataset consists of a large number of *paths*. Each path consists of a number of *points*, which are connected by *segments*.

Figure 7.2 illustrates the goal. The line dataset is abstracted to represent the overall structure.

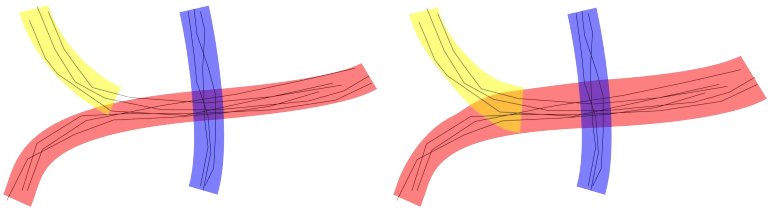


**(a)** Fiber tracked data **(b)** Abstract representation

**Figure 7.2:** Abstracting dense line data.

To achieve this we adapt the streamline method shown in Chapter 6. The integration will result in an *abstracted path*. Figure 7.3 shows how the abstracted paths approximate the optimal abstraction shown in Figure 7.2. In this figure the integration paths have been given either a constant width, or a width which depends on how much of the dataset the abstracted path represents. While not a perfect reconstruction of the optimal abstraction, it comes quite close.

The adapted streamline method is called *shaped integration*.



**(a)** Constant width **(b)** Width dependent on number of segments

**Figure 7.3:** Abstraction of the dense line data using abstracted paths.

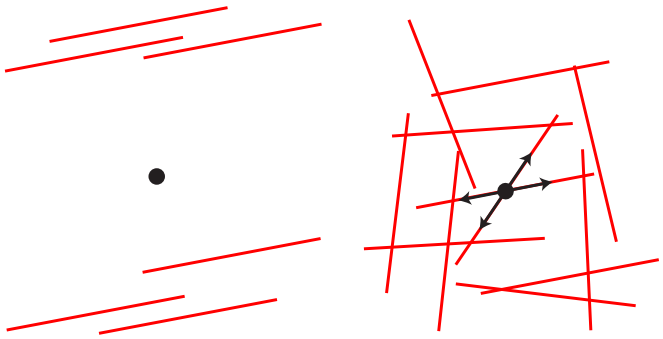
In summary: shaped integration is used to show the coarse structure of a dense line dataset. It achieves this by generating a number of structures that represent a larger number of segments. To prevent occlusion and reduce visual complexity, these structures should not be close to each other.

Using the streamline method on dense line data presents a challenge in finding the direction of the next integration step.

A dense line dataset may contain areas without any paths. The streamline method usually takes the approach of using the data that is nearest. If we would use the same approach with dense line data, that would mean the empty areas would not actually be seen as empty, but represented by the segment closest to it. This is however not representative of the data.

In addition to empty areas, dense line data may also include intersecting segments. These are areas where, for example, two paths cross. In the intersection point, two different directions both apply.

These two situations are illustrated in Figure 7.4.



**(a)** None **(b)** Multiple

**Figure 7.4:** Which segment represents this point?

To resolve these issues, we introduce the *shaped kernel*. The shaped kernel determines, based on a certain criterion, how much a segment is to be taken into account. A shaped kernel consists of two parts. Firstly a *difference metric*, which defines what the shaped kernel is sensitive to. Secondly a *kernel function*, which maps the difference metric to a weight.

The issue of empty spaces can be addressed by using a shaped kernel with a difference metric based on distance, which checks what segments are within a certain range. Segments that are too far away are not taken into account.

To address the issue caused by intersecting segments, a shaped kernel can be used whose difference metric is based on the angle between the previous integration step and a segment. If the angle is too large, the segment isn't taken into account.

A combination of these two shaped kernels give us the tools we need to integrate over the dense line data.

### 7.1.1 *Seed Point*

The streamline algorithm needs a point to start. This point, the seed point, can either be manually selected or determined by a

seeding algorithm. While manual selection can be useful if we already know the structure of the dataset, and are interested in a specific region, it is a time-intensive task which is prone to error.

Our method is intended to provide insight in the dataset, so requiring *a priori* knowledge of the dataset greatly reduces its usefulness. Automated seed selection can select seed points with much less knowledge of the dataset.

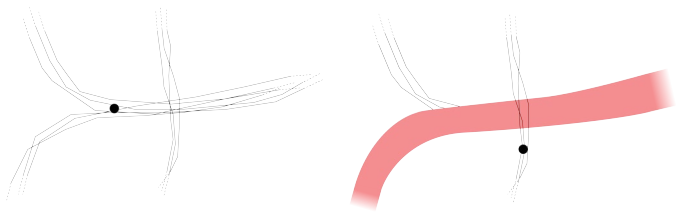
The goal of the visualization is primarily to show the overall *structure* of the data. The most prominent elements of the structure are the areas of the dense line dataset where the largest concentration of paths lie. It is therefore logical to start integration in an area that is densely populated by coherent segments.

As the densest areas usually occur halfway through a bundle, rather than at its endpoints, using one integration path would yield only half of the bundle. Therefore we need to start two integration paths per seed point, in two opposing directions.

## 7.2 FIELD COMPENSATION

Once we have calculated the integration path based on a certain seed point, the segments that were used in that integration have had a certain influence on that path. It can therefore be said that these segments have been represented by the path. As they have already been represented, further paths should not take them into account again. The logical step therefore is to remove the segments as *compensation* for making the integration path.

Figure 7.5 shows a single integration path using this method. Note that the segments in the crossing paths are unaffected, as the kernel function discarded them based on the angle between them and the integration. They have thus not been represented, and should not be removed. The segments of the path at the top left have been discarded by the kernel function based on the distance between them and the integration path, and are not removed either.



**(a)** Before **(b)** After

**Figure 7.5:** The dataset (black) before and after compensation based on an integration path (red). Maximum density is shown as a black dot.

Doing this means that the segments near the seed points will be compensated, as the abstracted path definitely represents them, which results in a different point of maximum density. This way the seed point selection method can simply be repeated to produce multiple abstraction paths.

### 7.2.1 *Connectivity Information*

One piece of data we have until now ignored is the connectivity information included in a path. A path is not just a collection of segments, it is a sequence.

In the abstracted view however, there is no way to accurately show this information.

As we use a streamline method, we have a single path of integration. A bundle however may merge with other bundles or split into multiple other bundles.

If we were to require that an abstracted path corresponds with a path in the dataset, the situation shown in Figure 7.6 would require splitting and merging integration paths.

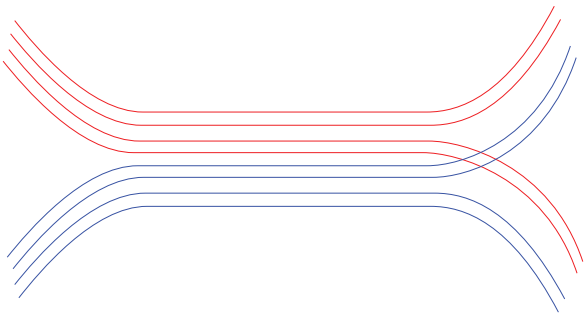


**Figure 7.6:** Paths not assignable to one particular bundle.

While branching off is a possibility, the goal of the visualization is not to give a bundling of the dataset, but rather illustrate its main features. Following a specific path is not necessary to accomplish this. Furthermore the resulting abstraction is likely to look very similar to the result without the use of connectivity information. Therefore we choose to discard it, and simply consider a path as a set of unrelated segments.

## 7.3 A CLOSER LOOK

In this chapter we have given a short introduction into the methods we will be using. The following chapters will provide a more in-depth view, providing the details of each of the methods.

In Chapter 8 we will go into the details behind the dataset we use to represent the dense line data: the *segment set*. We also show the methods that can be used to facilitate the compensation after integration.

Chapter 9 expands on the various options we have with integration. It takes a closer look at, and formalizes the shaped kernel, including the kernel function and distance metric, and the method of integration. It will also describe a method for determining the seed point and starting direction.

In Chapter 10 the method for compensation is detailed.

After that, we will go into the results and details of the method, in Chapter 11 through 14.

In this chapter we have introduced a number of terms regarding the proposed method. This section gives a short recap of these terms.

**Dense line data** consists of a number of **paths**, paths consist of a number of **points**, these points are connected by **segments**.

Our method uses **shaped integration**, which is a method of integration that can be applied to a segment set. It uses a number of **shaped kernels**, which consist of a **difference metric** and a **kernel function**, to determine what segments are relevant. Shaped integration generates a number of **abstracted paths**.

The integration starts at a **seed point**, which is defined as the point with the highest coherent segment density. After integration the segments that have been represented by the abstracted path are removed from the dataset, using **field compensation**. This results in a new point with maximum density, yielding a new seed point.

The method can be described as a function taking as input a dense line dataset `DLD` and the number of tracts that should be generated `N`:

```
function abstract(DLD, N)
SS := convert_to_segment_set(DLD)

paths = []
for i := 1 to N-1 do:
    seed    := pick_seedpoint(SS)
    path[i] := shaped_kernel_int(SS, seed)
    SS      := compensate(SS, path[i])
end
return paths
```

**Listing 7.1:** Method to generate an abstracted representation of dense line data.

The functions used here are explained in further detail in the following sections.

- `convert_to_segment_set()` in Section 8.1
- `pick_seedpoints()` in Section 9.4
- `shaped_kernel_int()` in Section 9.5
- `compensate()` in Section 10.1

# 8 SEGMENT SETS

As has been stated in the previous chapter, we will use the dense line data to create a *segment set*. We do this by adding each step of the path as a segment to the set. The formal definition is as follows:

A dense line dataset $DLD$ can be defined as a set of $n$ paths $D_0, \ldots, D_{n-1}$, where $D_i$ consists of a sequence of points $\mathbf{p}$. If $D_i$ has $m$ points, $D_i = \{\mathbf{p}_0, \ldots, \mathbf{p}_{m-1}\}$.

We define a segment set $S$ as: $n$ tuples $S_0, \ldots, S_{n-1}$, where $S_i = (\mathbf{a}_i, \mathbf{b}_i)$, where $\mathbf{a}$ and $\mathbf{b}$ are two coordinates.

## 8.1 CONVERSION

In order to convert a dense line dataset to a segment set, we follow the method in Listing 8.1.

```
function convert_to_segment_set(DLD)
SS := ∅
n := number of paths in DLD
for i := 0 to n-1 do:
    m := number of points in D_i
    for k := 1 to m-1 do:
        add (p_{k-1}, p_k) to SS
    end
end
return SS
```

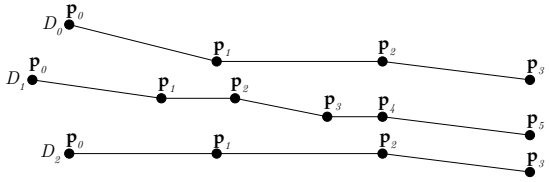
**Listing 8.1:** Converting dense line data to a segment set.

This turns each path into a set of segments. Figure 8.1 shows the result of this method when applied to a small set of paths.

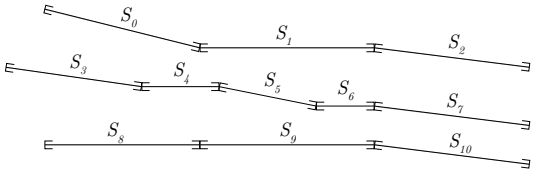
## 8.2 COMPENSATION

When previously discussing the notion of compensation after an abstracted path was created, we used the term 'remove'. While actually removing segments from the segment set is possible, we used a more nuanced approach.

If we assign to each segment in the segment set a certain *segment weight*, we can use this weight to lower the significance of a segment. Reducing the weight to zero has the same effect as removing it. This way a segment that was taken into account during the generation of an abstracted path, but contributed only partially, can be reduced in weight correspondingly, i. e. can be compensated.




**(a)** Dense line dataset

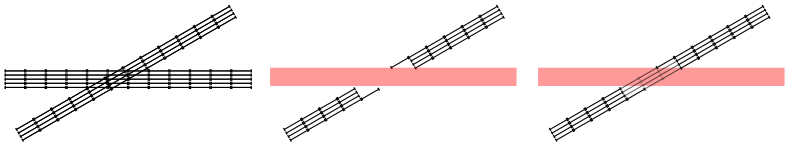



**(b)** Segment set

**Figure 8.1:** Converting the dense line dataset to a segment set.

Figure 8.2 shows the benefit this approach has. In Figure 8.2a a segment set with assigned weight is shown. An abstracted path is then calculated, shown in red. Figure 8.2b shows the result of removing the segments that were taken into account during integration. The diagonal segments were partially used, but did not produce a large influence on the abstraction path.

Compared to complete removal, the compensation method in Figure 8.2c yields more favorable results, as while the weight of the diagonal segments is decreased, a clear path still exists.



**(a)** Original **(b)** Removal **(c)** Compensation

**Figure 8.2:** Compensation of segment set (black) after calculation of abstraction path (red), using (b): segment removal, (c): weight reduction. Weight of a segment is shown as thickness. Abstraction path is shown in red.

The method to determine how much a segment contributed to an abstracted path is discussed in Chapter 10.

# 9 SHAPED KERNEL INTEGRATION

The goal of our method is to generate a set of paths that represent an abstraction of the dataset. This abstraction should contain the most prominent features of the dataset.

To achieve this, we will use shaped kernel integration, as shown in Chapter 7. Shaped kernel integration will, given a segment set, generate a path that corresponds with the most prominent bundle of coherent segments in the set. In this chapter we will formalize the method of integration, including the shaped kernel and seed selection.

## 9.1 INTEGRATION METHOD

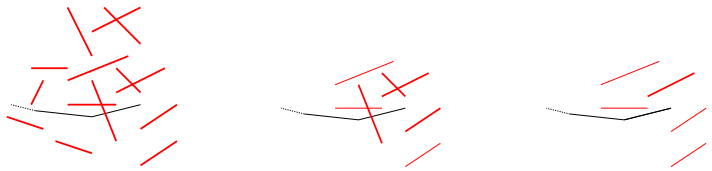
Shaped kernel integration is inspired by the streamline method. Starting from a certain point, it aims to locally follow the densest segment bundle.

Given a certain segment set $SS$, the current integration point $p$ and the previous step of integration $d$, it will determine the next integration step. This step depends on the segments in its neighborhood. Each of these segments has an associated weight and direction, which are taken into account during integration. As the integration path represents a variable number of segments, we also assign a representing weight to each integration step. An integration step that represents many segments is given a higher weight than one that represents only a few.

During integration we need to compute the direction of the next step at a certain point in a segment field. This point may have no segments nearby (*locality*), or may be on crossing segments, providing two distinct directions for a single point (*ambiguity*).

These issues can be solved using a shaped kernel. A shaped kernel is a combination of a difference metric and a kernel function, and is used to decide what segments should be taken into account, and to what extent.

Figure 9.1 shows how the segment set is reduced using two shaped kernels. First a shaped kernel with a difference metric based on distance is applied, removing any segment that is too far away, whereafter an angle metric based shaped kernel is used to remove the segments that run in a direction too different from that of the previous integration step.



**(a)** Original **(b)** Distance **(c)** Angle

**Figure 9.1:** The result of first applying a distance based shaped kernel in (b) and then angle based in (c), to the segment set (red) shown in (a). Current integration path is shown in black.

## 9.2 DIFFERENCE METRICS

The two challenges faced in shaped kernel integration are locality and ambiguity of the segment set. We address these by means of two difference metrics.

To solve the locality problem, a metric based on the distance between the segment and the current integration point is introduced.

The ambiguity problem can be solved using a metric based on the angle between the direction of the previous step of integration and a segment.

### 9.2.1 *Distance*

When integrating over a segment set, we want to only consider data that is close to us. To create this effect, a *distance metric* is introduced. Given a segment $S$ and a point $\mathbf{p}$, this metric yields the value $\Delta_d$: the smallest distance between $S$ and $p$.

Figure 9.2 shows the values of $\Delta_d$ in different situations.

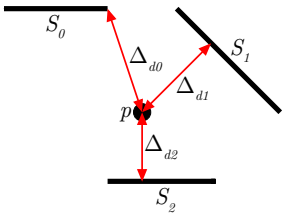


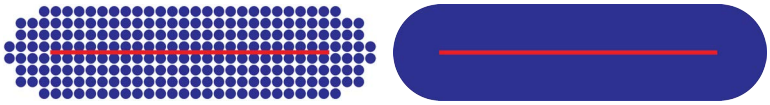

**Figure 9.2:** $\Delta_{di}$ for point $p$ and various segments $S_i$.

*Region of Influence*

Showing the effect a distance based shaped kernel has can be difficult. An intuitive visualization can be achieved by not showing the kernel around a point, but by showing it extending from a segment. This changes it from showing which points take the segment into account, to what region the segment influences.

This duality can be seen in Figure 9.3. If a plane is densely covered by points, and we discard the points that are not influenced by the segment, the pattern seen in Figure 9.3a emerges.

In Figure 9.3b a segment is drawn with a region of influence extending a certain range from the segment. It can clearly be seen that if we increase the number of points in Figure 9.3a, the images become more and more similar.



**(a)** Densely packed points **(b)** Region of Influence

**Figure 9.3:** Duality of a distance kernel and region of influence.

The region of interest interpretation allows us to easily visualize the effect the the shaped kernel has on a set of segments.

*Normalization*

When we use the region of influence visualization on the segments that result from converting a path, an issue comes to light. The summed weight of a path sampled at a certain rate, is larger than that of the same path sampled at a lower rate.

Consider a dense line dataset with two crossing bundles of an equal number of lines. One dataset is sampled at a frequency $f$, while the other is sampled at $2f$. This situation is shown in Figure 9.4. The average weight around the former is lower than around the latter. This results in a preference for following the high-frequency bundle, while both bundles are equal and no such preference should exist. This effect will lead to inaccurate results during integration.

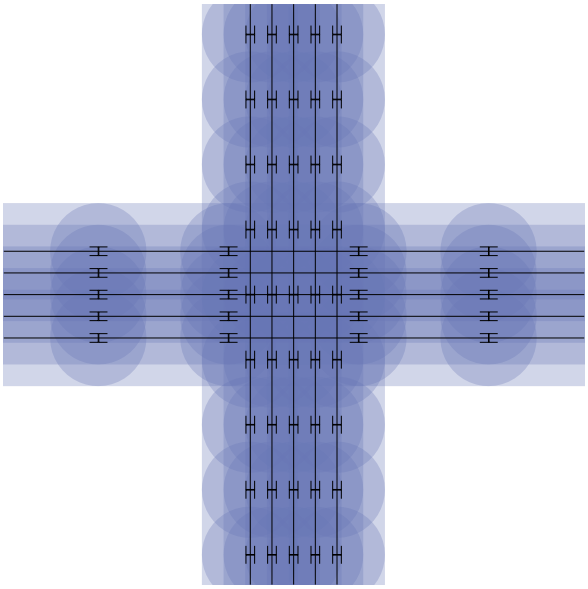


**Figure 9.4:** Two crossing segment bundles with different sample frequencies. The horizontal bundle is sampled at frequency $f$, the vertical at $2f$.

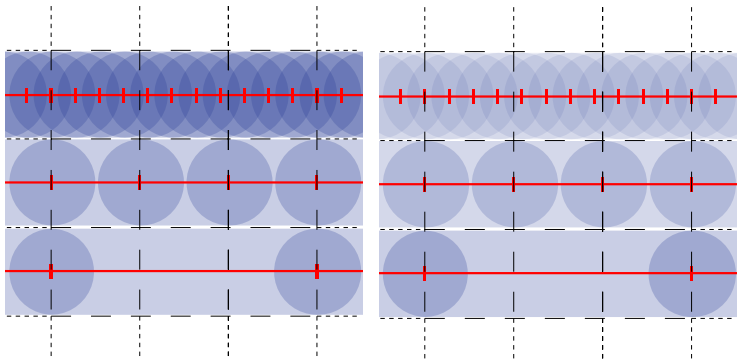
The difference stems from the region of influence a segment of zero length has. Such a segment has a non-zero region of influence. Thus, if we sample at an infinite sample rate, the weight of that line will also be infinite. A comparison of the weight around the same paths with different sampling frequencies can be seen in Figure 9.5a.

To compensate for this the calculated weight $w_p$ that results from the distance based shaped kernel is normalized. To arrive at the normalized value $w_n$, $w_p$ is multiplied by a value proportional to the ratio of the weight of a zero length segment $w_0$, and the weight of the total segment $w_s$:

$$w_n = w_p \cdot (1 - \frac{w_0}{w_s}).$$

Note that when $w_s \gg w_0$, $\frac{w_0}{w_s} \approx 0$, thus $w_n \approx w_p$ for long segments.

In Figure 9.5b the result of this length dependent weight can be seen. While still not uniform, it is clear that the average weight is less dependent on the sample rate.



(a) (b)

**Figure 9.5:** Region of influence (blue) around a segment (red). Sample frequency decreases from top to bottom. (a): constant weight, (b): weight by length.

### 9.2.2 *Angle*

If we define the difference metric to not be sensitive to distance, but angle, we can begin solving the problem of ambiguous direction.

We take the last integration step, which lies between current and previous integration points, $\mathbf{p}_i$ and $\mathbf{p}_{i-1}$. We then define $\Delta_a$ as the smallest absolute angle between that and a segment. Figure 9.6 shows the values of $\Delta_a$ for different segments.

This creates a preference for segments that run in the same direction as the integration.

As segments do not have a direction *per se*, a method is needed to extract a direction from a segment set. In order to do this, we

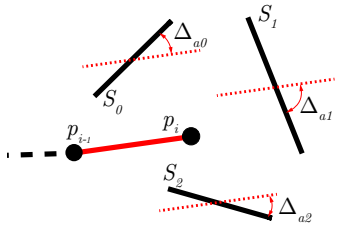



**Figure 9.6:** $\Delta_{ai}$ for point $p$ and various segments $S_i$.

convert the segment set $S_{max}$ to a vector set $V_{max}$. The method (as shown in Listing 9.1) takes as argument a segment set `SS`, which in this case is $S_{max}$, and the previous step direction `dir`, which is $\mathbf{p}_i - \mathbf{p}_{i-1}$.

```
function convert_ss_to_v(SS, dir):

V := []
for each segment in SS do:
    v̄ := segment.b - segment.a
    if v̄ · dir < 0 then:
        v̄ := −v̄
    end
    v̄ := v̄ * segment.weight
    add v̄ to V
end

return V
```

**Listing 9.1:** Converting a segment set to a vector.

The direction can then be determined by taking the average of the direction of the segments that were not discarded. Note that, since segments do not have a direction, we assign the direction to the segment that results in the smallest angle between the segment and the current integration direction.

Care must be taken not to discard too eagerly or reluctantly, since this can result in either missing relevant segments, or a non-significant, noise-sensitive result. Generally $\Delta_a < \frac{\pi}{4}$ yields results that are flexible while still yielding significant results. Figure 9.7 shows the influence the cutoff angle has. Note in this figure that increasing the cutoff angle decreases the magnitude of the average, thus yielding less certain results.

## 9.3 KERNEL FUNCTIONS

A kernel function is a function $w : \mathbb{R}_{\geq 0} \to \mathbb{R}$. It maps a certain difference metric ($\Delta$) to a weight ($w$). This weight is then multiplied by the weight of the segment, to get the weight that is added to the current integration step.

The distance metric can be any quantifiable property, but in the following sections we will focus on the two previously mentioned metrics: *distance*: a metric produced using a segment and the current integration position, and *angle*: a metric determined

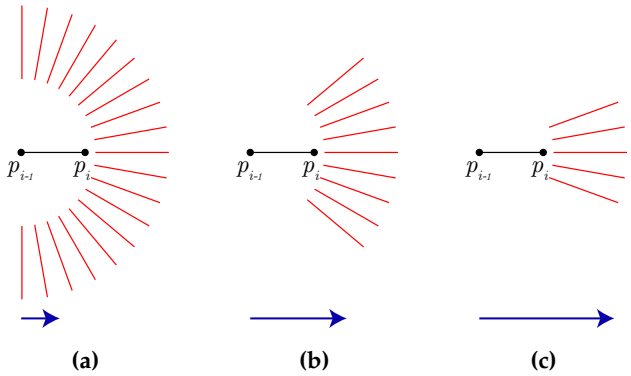



**Figure 9.7:** The effect of different cutoff values on the selected segments. Cutoff value of: (a): $\frac{\pi}{2}$ (b): $\frac{\pi}{4}$ (c): $\frac{\pi}{8}$. The average of the segments is shown as a blue arrow.

using a segment and the direction of the previous integration step.

We apply the following restrictions to the kernel function:

**Restriction 1:** $w(x) \geq 0$,
i. e. $w$ is positive.

A negative weight would result in one segment influencing other segments, as one segment can negate the weight of another. While this can be used to create kernel functions with results similar to a Canny edge detector [11], we will not use it in this paper.

**Restriction 2:** $w(0) = 1$,
i. e. When $\Delta$ is zero, $w$ is one.

When the difference metric is at its lowest possible value a segment should be completely taken into account. If $w(0) = x \neq 1$, a replacement function $w$ that satisfies this restriction can be defined as $w' = \frac{w}{x}$.

**Restriction 3:** $a \geq b \Rightarrow w(a) \geq w(b)$,
i. e. $w$ is monotonically descending.

Similar to restriction 1, this restriction can be lifted to obtain certain specific effects, but for our method we will use a function that ensures that a segment that has a higher $\Delta$ will be assigned a lower weight.

**Restriction 4:** $\exists a, \forall x > a : w(x) \approx 0$,
i. e. $w$ eventually approaches zero.

There should be a certain point $\Delta = a$, after which the weight is approximately zero, and remains there. If this is not the case, a segment that is infinitely different may still be taken into account.

**Restriction 5:** $w(x) \approx w(x + \epsilon)$,
i. e. $w$ is insensitive to noise.

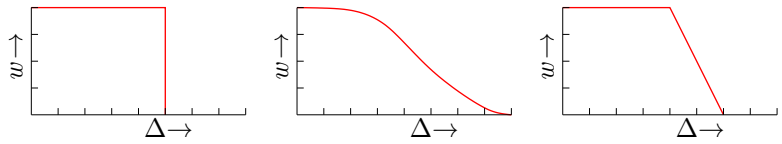
This restriction makes sure that no large changes in weight assignment occur as a result from a minor change to $\Delta$.

These restrictions ensure the kernel behaves in a manner consistent with the metrics we define. Note that the combination of

restrictions 1, 2 and 4 restricts $w$ to the range $[0,1]$. Also note that we don't require $\int w(x)dx = 1$. This would have no reason, as $w(x)$ is only used relative to itself. For each kernel any number of equivalent kernels exist $c \cdot w(x) \equiv w(x)$ with $c \in \mathbb{R}_{>0}$.

Furthermore some $\Delta$ are inherently restricted. Take for example the minimal angle, where $\Delta_a \leq \pi$. In these cases restriction 4 should be adapted to: $\exists a < max(\Delta), \forall x > a : w(x) \approx 0$.

Three different kernel functions will be presented in the following sections. These functions are shown in Figure 9.8.




**(a)** Step **(b)** Gaussian **(c)** Trapezium

**Figure 9.8:** Three possible kernel shapes.

### 9.3.1 *Step Function*

The step function, shown in Figure 9.8a, is defined as:

$$w(x) = \begin{cases} 1 & \textbf{if } x \leq \alpha \\ 0 & \textbf{if } x > \alpha \end{cases}$$

This is the function we have seen in the previous sections. Using the step function as kernel function only allows for segments that satisfy a certain criterion, all others are discarded. While computationally and conceptually very simple, this kernel breaks restriction 5, as it suffers from severe noise sensitivity: $w(\alpha) = 1,\ w(\alpha + \epsilon) = 0$. This makes it less suitable for use as kernel function.

### 9.3.2 *Gaussian Function*

The Gaussian function is defined as follows:

$$w(x) = ae^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

It can be used as a kernel function in a number of different ways. Taking $\mu = 0$ and $a = 1$ yields a kernel much more smooth than the step kernel, as shown in Figure 9.8b. Its defining function is

$$w(x) = e^{-\frac{x^2}{2\sigma^2}}.$$

Note that any value of $\mu \leq 0$ can be used without breaking restrictions, as long as we take:

$$a = e^{\frac{\mu^2}{2\sigma^2}}.$$

Using a $\mu < 0$ results in a steeper descent from $x = 0$. Using $\mu > 0$ would violate restriction 4.

Two issues however make for poor usability of this function: Firstly it is computationally intensive, calling for the use of two power functions and a division ($2\sigma^2$ and $a$ are constants).

The second, less major issue is the high configuration complexity of the kernel. If for testing a steeper descent is needed, but the area under the function needs to remain constant, several complex changes need to be made to $\mu$, $a$ and $\sigma$, involving either integration or approximation.

### 9.3.3 *Trapezium Function*

In order to address the noise issues of the step function and the complexity issues of the Gaussian function, a compromise is proposed. The trapezium function, shown in Figure 9.8c, is defined as follows:

$$w(x) = \begin{cases} 1 & \textbf{if } x \leq \alpha \\ 1 - \frac{x-\alpha}{\beta} & \textbf{if } x > \alpha \textbf{ and } x \leq (\alpha + \beta) \\ 0 & \textbf{if } x > (\alpha + \beta) \end{cases}$$

The function has a section between zero and $\alpha$ where $w$ is one, similar to the step function. Instead of directly becoming zero however, the trapezium function has a range $\beta \geq 0$ in which $w$ linearly decreases. The case $\beta = 0$ can trivially be shown to be equivalent to the step function.

This function can easily be configured to maintain constant area, by keeping the value of $2\alpha + \beta$ constant.

Using a value of $\beta$ that is a reasonable factor larger than the amount of noise in the data sufficiently reduces noise sensitivity. Given noise $\epsilon$, the maximum error can be computed as $\frac{\epsilon}{\beta}$.

## 9.4 SEED POINT

In previous sections we have assumed a point and direction were known, as we were at a point other than the starting point of integration. We do however need a way of determining these start or *seed points*. Manual seed selection is a possibility, and while this makes sure that integration starts at the location we want it to, it is an arduous method which is prone to error and which requires a priori knowledge about the structure of the dataset. Automated seed point generation would alleviate this problem.

Manual seed selection is an appropriate method if we are interested in a very specific region. For our method however we

need paths that represent the *important* bundles in the data. To achieve this, an automated method should select seed points that lie along important paths. These points are likely to consist of dense bundles.

### 9.4.1 *Seed Position*

To determine where to start integration, a regular grid with edge size $x$ is placed over the dataset. At each grid point a shaped kernel with $\alpha = \frac{x}{2}, \beta = \alpha$ is applied, selecting only the segments close to the grid point. Note that some overlap is introduced, as a square grid is used, while the distance metric results in a circular area.

Using the trapezium function with a reasonably large value for $\beta$ yields quite uniform coverage, as can be seen in Figure 9.9. A different regular space filling grid can be used to increase the accuracy of the seed position selection method, such as hexagons (2D) or truncated octahedrons (3D).

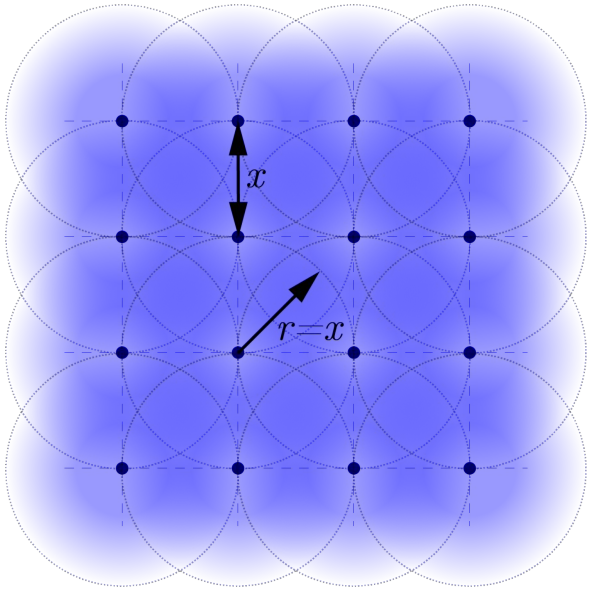



**Figure 9.9:** Coverage of the shaped kernel used in the seed selection method. Grid spacing = $x$, $\alpha = \beta = \frac{x}{2}$. Grid points are shown as black dots, a circle with radius $x$ is shown around each grid point

At this point we could calculate the summed weight for each grid point, and use the grid point with the highest weight as the seed point. This would however mean that an area with a large number of random segments would be preferred over an area with a slightly smaller, but still large, number of collinear segments. To address this issue, we also determine a seed direction.

### 9.4.2 *Seed Direction*

We apply a number of shaped kernels using an angle metric to the result of the distance based kernel. For the input direction $\bar{r}$

of the angle metric, a number of vectors are chosen that equally subdivide the space around the grid point.

In 2D this is trivial, as we can take $n$ vectors $\frac{2\pi}{n}$ apart, as in Figure 9.10.

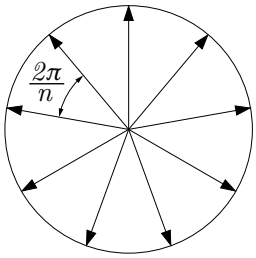
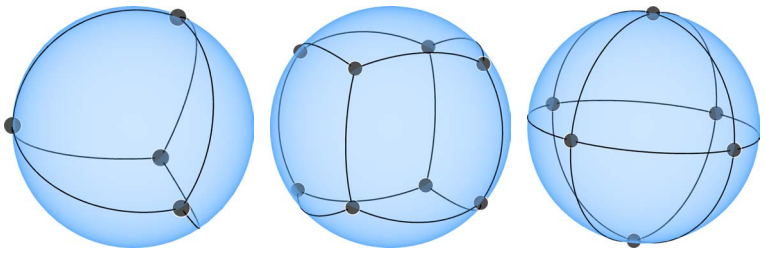



**Figure 9.10:** $\bar{r}$ used as input for shaped kernel directions in the case $n = 9$.

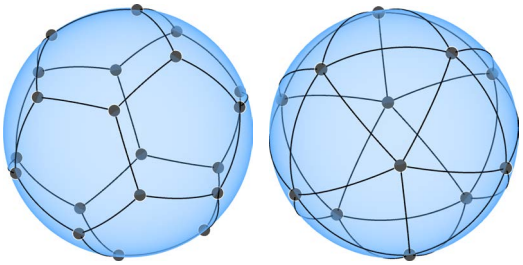
The 3D case is less trivial. We are looking for an equal subdivision of the surface of a sphere. In many cases an approximation of an equal subdivision can be made [46].

If the number of desired subdivisions is however the same as the number of vertices of any *platonic solid*, an exact solution can be found. Projecting a platonic solid to a sphere (shown in Figure 9.11) results in an equal subdivision of the surface of a sphere. If we take the vectors from the center of the platonic solid to the center of each of its surfaces as $\bar{r}$, we have achieved our goal.

While the number of platonic solids is limited, they provide us with a solution for 4, 6, 8, 12 and 20 directions. These cases have proven to provide sufficient accuracy for the datasets used in this paper. A higher number would be needed for a dataset in which bundles are defined with higher accuracy. In such a case the difference between bundles can be very small, while still being significant, requiring a smaller value for $\alpha$.



**(a)** Tetrahedron **(b)** Hexahedron **(c)** Octahedron



**(d)** Dodecahedron **(e)** Icosahedron

**Figure 9.11:** Projection of the platonic solids on a sphere [42].

The $\alpha$ of the angle based shaped kernel is set to the smallest angle between two different $\overline{r}$ (for example, for a tetrahedron $\alpha \approx 109.5°$, for an hexahedron $\alpha = 90°$, for an octahedron $\alpha \approx 70.53°$, etc). $\beta$ is set to the same value, following the same reasoning as with the location based kernel.

Each of the applications of these shaped kernels results in a weighed set of segments. The grid location which yielded the set with the highest `weight`($S_{max}$) is selected as starting point. $S_{max}$ is is then used to determine the starting direction.

In this method we use `convert_ss_to_v()`, in which each segment $s = (\mathbf{a}, \mathbf{b})$ is converted to a vector $\overline{v}_s = \mathbf{b} - \mathbf{a}$. It is then compared to the input vector. If the angle is larger than $\frac{\pi}{2}$, i. e. $\overline{v}_s \cdot \overline{r} < 0$, the vector is the wrong way around, and is flipped. Note that this method is very sensitive to noise if segments with an angle close to the maximum angle ($\frac{\pi}{2}$) are taken into consideration. Therefore it is advised to select the number of vectors so that the maximum angle that is taken into consideration is significantly smaller than that.

As segments do not have an explicit direction, the direction we have just calculated can be mirrored to yield a vector that has the exact same properties. It is therefore recommended to start two integration paths from such a generated seed point, one in the direction calculated previously, and one in the exact opposite direction.

This method yields a point and two directions representing the densest collinear bundle of segments in the dataset. Generating multiple seed points however is not possible, as this maximum is very likely to be unique, while other (local) maxima may lie on the same bundle as our current seed point.

The technique discussed in the next chapter adapts the dataset so that after each completed integration path a new global maximum is available.

Now we combine the discussed techniques into a single method, shaped kernel integration. Input is a segment set `SS` and a seed `seed`, consisting of a position and a direction. Also known are: Shaped kernels for angle ($K_a$) and distance ($K_d$) and integration step size $\delta$. The implementation of this method is shown in Listing 9.2.

```
function shaped_kernel_int(SS, seed):

abs_path_part := []
for integration := 0 to 1 do:
    # initialize variables
    step     := 0
    pos      := []

    pos[0]   := seed.point
    cur_dir  := seed.direction[integration]

    while true do:
        # determine the step direction
        SS_pos  := K_d(SS, pos[step])
        SS_used := K_a(SS_pos, dir_cur)
        if weight(S_used) != 0 then:
            # perform an integration step
            step += 1
            cur_dir := avg(convert_s_to_v(SS_used
                ))
            cur_dir := normalize(cur_dir)
            pos[step] += cur_dir * δ
        else:
            # we are in area without
            # segments of interest, so we stop
            abs_path_part[integration] := pos
            break
        end
    end
    # reverse first part, concatenate and return
    abs_path_part[0] := reverse(abs_path_part[0])
    return abs_path_part[0] + abs_path_part[1]
end
```

**Listing 9.2:** Integration of a path using shaped kernel integration.

The methods called in this function have been previously described:

- `convert_s_to_v()`: Section 9.2.2
- $K_a()$ and $K_d()$: Section 9.3

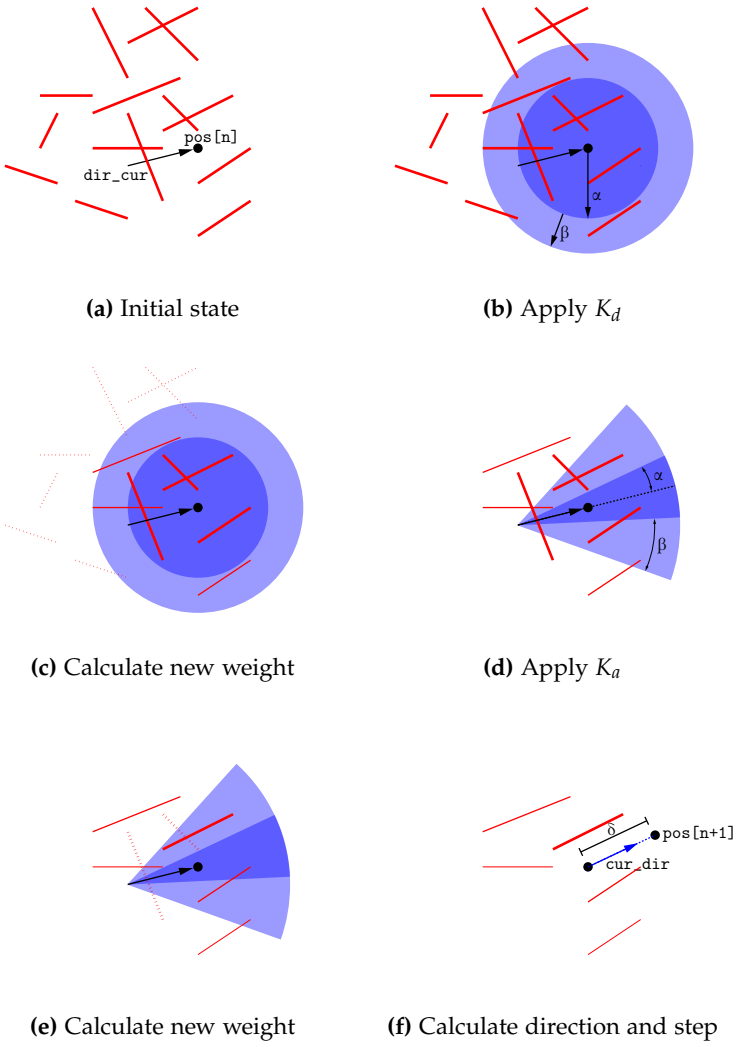
Note that, as we integrate from the center outwards, we need to reverse the first path part before concatenating.

In Figure 9.12 a step-by-step illustration of the method of integration is shown, deriving a direction from a segment field, given current integration position `pos[n]` and direction `cur_dir` (a). First the distance based shaped kernel $K_d$ is applied to the

segment field (b) to weigh the segments according to how close they are to the current integration location (c).

Then the angle based shaped kernel $K_a$ is applied (d) to weigh the segments according to how much their directions differ from current integration direction (e). Once this is done, we have obtained a set of segments that are weighed according to how much they apply to the current integration. A weighed average is then taken of these segments, the result of which is the new integration direction (f).

A new position is then calculated based on the integration direction and step size, and the process starts again.




**(a)** Initial state **(b)** Apply $K_d$

**(c)** Calculate new weight **(d)** Apply $K_a$

**(e)** Calculate new weight **(f)** Calculate direction and step

**Figure 9.12:** Method to obtain an integration direction from a segment set. Thickness of a segment indicates its weight. Dashed segments have a weight of zero, and will not be shown in the next step.

An implementation that uses this method is demonstrated using a real-life dataset. The dataset used is a slice of a DTI data showing the corpus callosum (visualized as semitransparent black lines), shown in Figure 9.13.

**Figure 9.13:** A slice of DTI data.

The method that is used in the following sections is shown in Listing 9.3.

```
function abstracted_paths(SS, seeds):
paths := []
path  := 0
for seed in seeds:
    #seed contains one point and two directions.
    paths[path] := shaped_kernel_int(SS, seed)
    path += 1
end
return paths
```

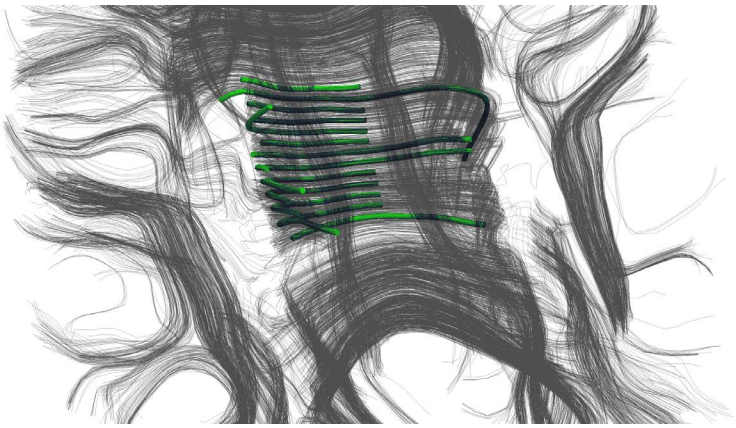
**Listing 9.3:** Abstracted path generation.

### 9.6.1 *Distance Metric*

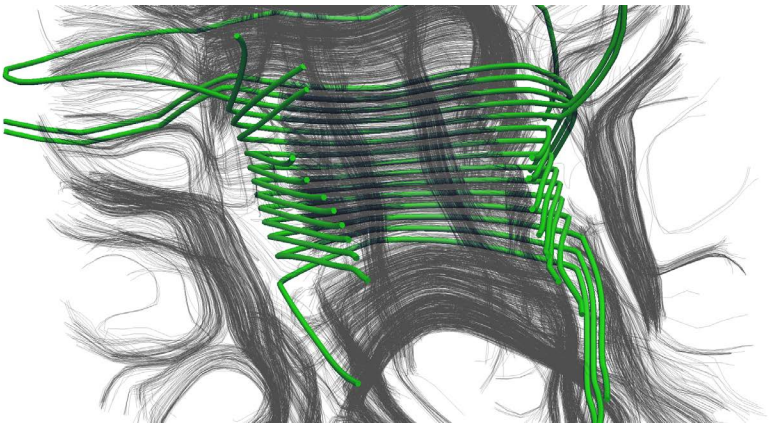
This section shows the influence the choice of distance based shaped kernel has on integration. Figure 9.14 shows integration over the corpus callosum dataset using a trapezium shaped kernel with different values for $\alpha$. $\beta$ is set to $\frac{\alpha}{2}$. Seed points are manually placed at regular intervals along the *medial longitudinal stria*, the center of the corpus callosum. At each point two integration paths are started in the direction of the two different halves of the brain.

The optimal value for $\alpha$ and $\beta$ was determined empirically, and depends on the dataset.

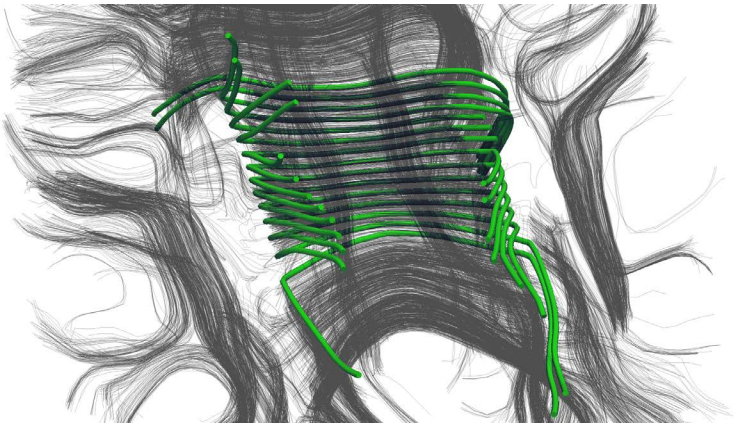
With a too small value of $\alpha$, integration will stop in small empty spaces in the dataset. With a value for $\alpha$ that is too large however, the direction of integration is no longer locally tangential, as segments that are quite far away are taken into account.



**(a)** Too small



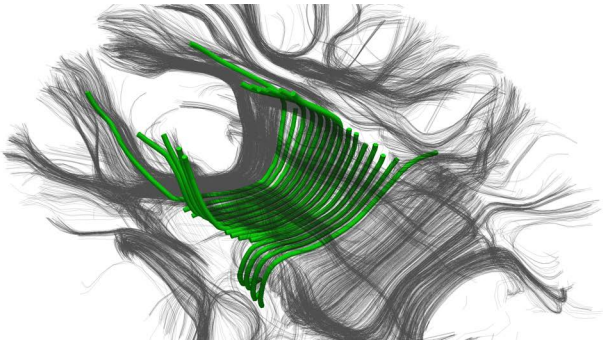
**(b)** Too large



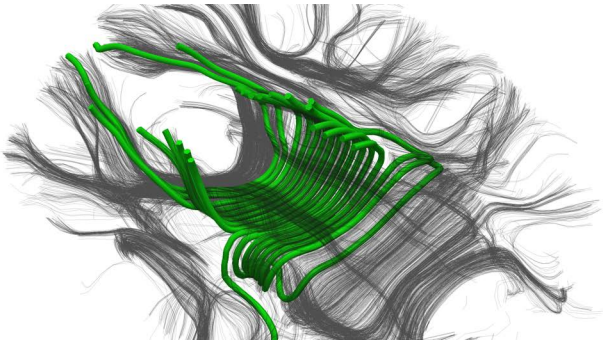
**(c)** Optimal

**Figure 9.14:** Influence of $\alpha$ on a distance based shaped kernel:
(a) premature stop due to a too small $\alpha$,
(b) deformation of the data due to a too large $\alpha$,
(c) optimal choice of $\alpha$.

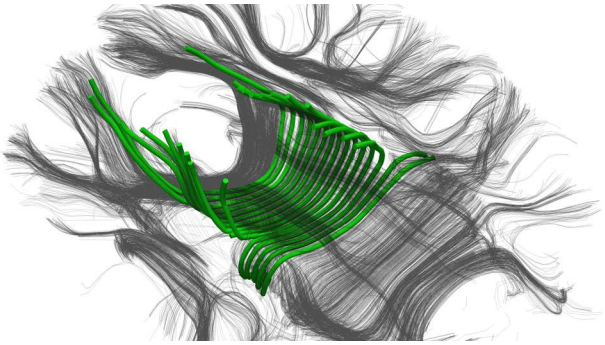
Using the same seed points as before, we now show the influence the angle based shaped kernel has. In Figure 9.15 three different angle kernels are taken, with $\alpha = \frac{\pi}{\{4,16,32\}}$, $\beta = \frac{\alpha}{2}$.



**(a)** Too small



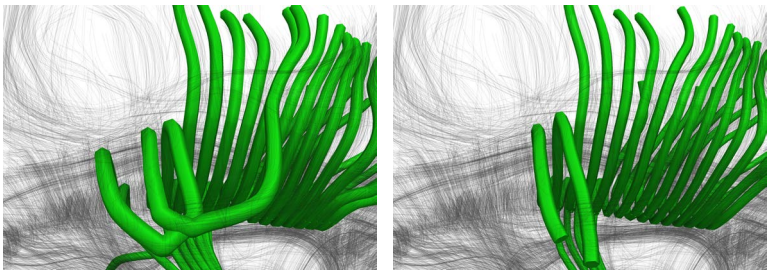
**(b)** Too large



**(c)** Optimal

**Figure 9.15:** Influence of $\alpha$ on an angle based shaped kernel:
(a) running outside a bundle due to a too small $\alpha$,
(b) wild paths due to a too large $\alpha$,
(c) optimal choice of $\alpha$.

As can be seen in Figure 9.15a, with a value of $\alpha = \frac{\pi}{32}$ integration cannot turn fast enough to stay within the bundle, resulting in premature termination of paths. With $\alpha = \frac{\pi}{4}$, as in Figure 9.15b, the integration is so sensitive to crossing paths that there are very sudden changes in the integration direction. A close up of these twists is shown in Figure 9.16a. With a reasonable value for $\alpha$, these twists disappear, as can be seen in Figure 9.16b. This results in the paths shown in Figure 9.15c.



**(a)** Wild twist with $\alpha = \frac{\pi}{4}$ **(b)** Same position with $\alpha = \frac{\pi}{16}$

**Figure 9.16:** High sensitivity leading to sudden changes in integration direction.

## 9.7 APPLICATION TO VECTOR FIELDS

With a small adaptation to the generation of the segment set, we can also make shaped kernel integration applicable to regular vector fields. On the surface this may seem counter-intuitive. We have have first modified the streamline algorithm to use non-vector field data, and are now adapting a vector field to be usable in our adapted method.

However, what data the streamline method used was not the only adaptation we made. A method to regulate the maximum turning rate, as well as a search region were added. This means integration over a vector field can be much more robust when using shaped kernel integration. This effect is shown in Figure 9.17.

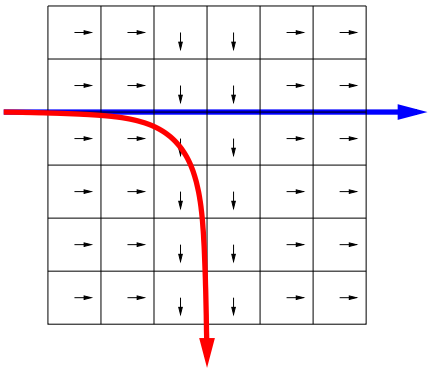
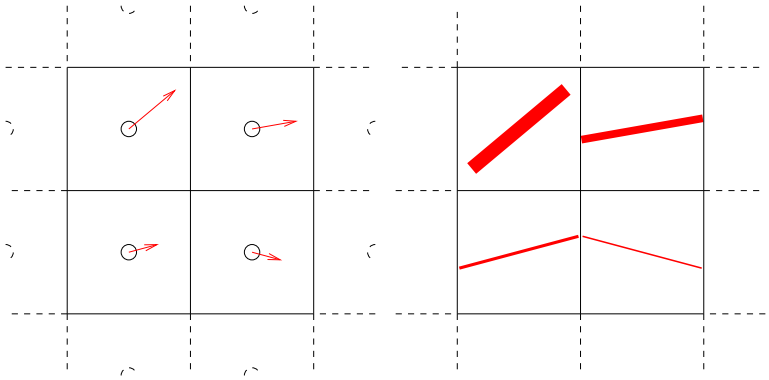


**Figure 9.17:** Comparison between regular streamline (red) and shaped kernel integration (blue) on a vector field.

To convert a grid-based vector field to a segment set we use the following method. For each grid point we create a segment with length equal to the distance between two grid points. The segment is placed with its center on the grid point, and rotated so the vector corresponding with that point and the segment are collinear. The segment weight is then set to the magnitude of the vector.

This method can be seen in Figure 9.18.



**(a)** Vector field **(b)** Segment set

**Figure 9.18:** Converting a grid-based vector field to a segment set.

This method is used on a fluid flow simulation showing turbulence near a flat inflow section visualized as semitransparent lines colored by weight.

The results of shaped kernel integration on this dataset can be seen in Figure 9.19 as green tubes. Seed points were placed along the inflow section in the bottom right of the image.

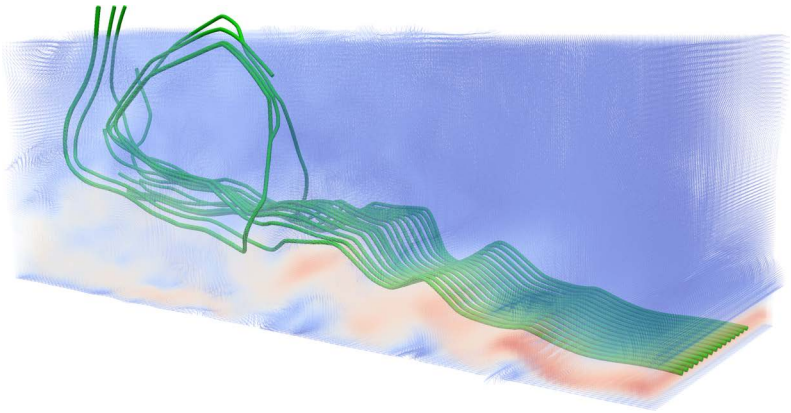


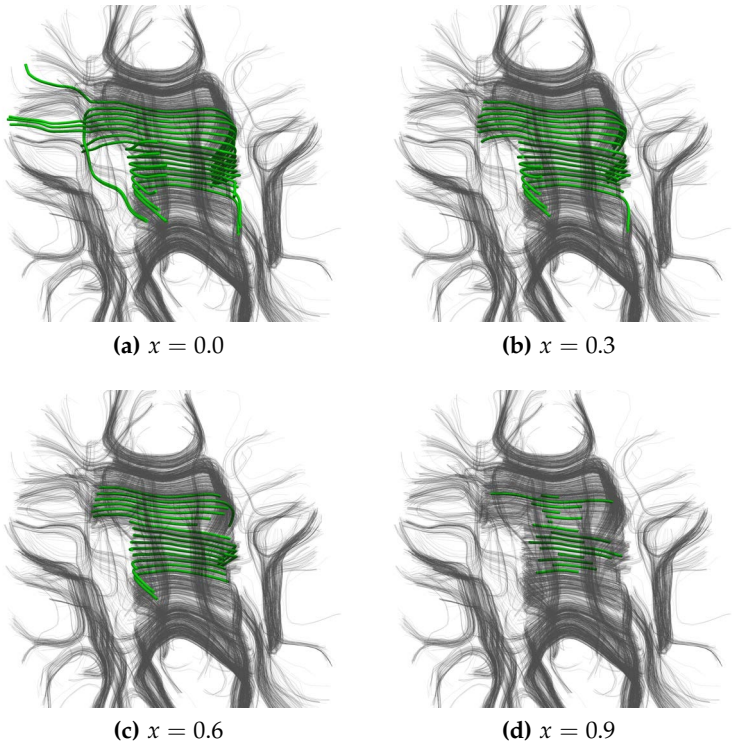
**Figure 9.19:** Integration over a segment set based on a regular vector field.

## 9.8 STOP CONDITION

Until now we have been integrating until we reached the end of the dataset. This stop criterion was implemented in the line `if weight(S_used)!= 0` of Listing 9.1. This means however, that as long as a single segment is still in range, integration will continue. A single segment however is not interesting for an abstraction method. Therefore we will adapt the stop criterion to reflect this.

As we start integration, we record the sum of the weight of the segments that are used in the first step (`weight(S_used)`). We then compare the weight of each subsequent step to that original weight. If this weight drops below a certain fraction of the initial weight, integration is stopped. This means that a certain integration path will only represent bundles that contain more than a

certain fraction of the bundle the seed was placed in. Applying this method with different values for the stop factor yields the results in Figure 9.20.



**(a)** $x = 0.0$ **(b)** $x = 0.3$

**(c)** $x = 0.6$ **(d)** $x = 0.9$

**Figure 9.20:** Integration stopping when weight is less than $x$ times the start weight.

In this figure various values are used for the stop criterion, ranging from 0 (only stop when there are no relevant segments) to 0.9. A value between 0.3 and 0.6 appears to yield the best results, depending on the goal of the visualization.

As using a stop criterion ensures at least a certain amount of weight is represented, it reduces the sensitivity to noise.

# 10 FIELD COMPENSATION

Each time an abstracted path is generated, a part of the original segment set is represented. As it has been represented, it should not influence any further abstracted paths, as this can result in a bundle being represented by an abstracted path more than once. This includes any influence the represented segments would have on the selection of a seed point. The method with which this is implemented is *field compensation*.

Each integration step is preceded by the application of a number of shaped kernels. After these kernels have been applied, every segment has been assigned a weight, based on their original segment weight and how well they fit within the kernels. Many segments have a weight of zero, few have a weight of one, and some segments fall between these two cases.

The property introduced in Chapter 8 called *segment weight* denotes how much influence the segment has on integration. This weight has also been used when converting vector fields to segment sets in Section 9.7. Note that this weight is separate from the weight applied by kernels, as it is preserved across the generation of multiple abstraction paths.

If we reduce the segment weight for the segments that have influenced integration, we *compensate* the segment set for the generation of an abstracted path. This way we can say that during generation of paths the compensated segment set plus the abstracted paths generated so far are an approximation of the original dataset.

The goals for compensation are:

- Segments that have not been used should not have their segment weight changed.
- Segments that have been fully represented by an abstracted path should have their segment weight compensated to zero.
- Segments that have partially been represented by an abstracted path should have their weight partially compensated.
- Segment weight should never become negative.

To implement this we record, for every segment, the maximum weight it has had during the calculation of the integration direction. Recalling the implementation shown in Listing 9.2, this would be the maximum of the weights it had in `S_used`.

After a complete abstraction path has been calculated the segment weight of each segment is decreased by this maximum. For example we take a certain segment $S$ that has an initial weight

of 0.7. During integration it is taken into account several times, with the weights shown in Table 10.1 (Note: for brevity entries where any factor is zero are not shown).

| weight($S$) | · | $K_d(S)$ | · | $K_a(S)$ | = | represented weight |
|---|---|---|---|---|---|---|
| | | | ... | | | |
| 0.7 | | 0.1 | | 0.4 | | 0.028 |
| 0.7 | | 0.3 | | 0.6 | | 0.126 |
| 0.7 | | 0.9 | | 0.8 | | 0.504 |
| 0.7 | | 0.3 | | 0.5 | | 0.105 |
| 0.7 | | 0.1 | | 0.2 | | 0.014 |
| | | | ... | | | |

**Table 10.1:** Representing weights for a certain segment.

After this integration is done, the maximum represented weight, in this case 0.504, is subtracted from the original weight of 0.7. For the next integration $S$ will have a weight of 0.196.

In Figure 10.1 this method is illustrated with two crossing bundles at an angle of $\frac{2}{9}\pi$. In this illustration the angle based kernel has an $\alpha$ and $\beta$ of $\frac{1}{6}\pi$ each. The directional kernel has an $\alpha$ slightly larger than the width of the horizontal path, and a relatively small $\beta$. The weight of all segments initially (in Figure 10.1a) is 1.0.

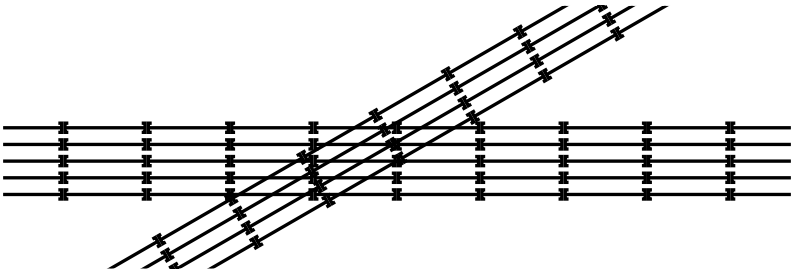
As the abstracted path along the horizontal bundle was computed, it crossed the diagonal bundle.

For all segments $S$ in the horizontal segment bundle we know that the maximum represented weight is 1.0, as every segment in the bundle is at one point within the $\alpha$ range of integration, where $K_d(S) = 1.0$. As integration runs almost parallel to each segment, $K_a(S)$ has a constant value of 1.0.

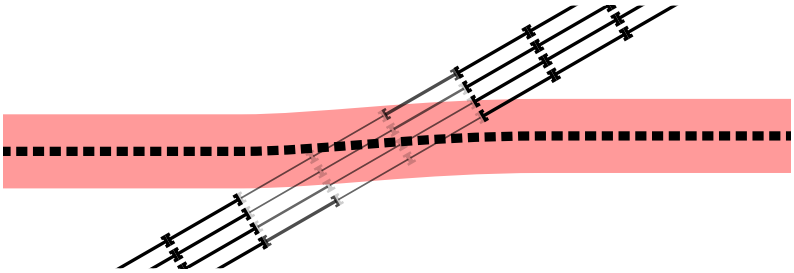
For all segments $S$ in the diagonal bundle, the value of $K_a(S)$ is approximately constant (as the direction of integration does not vary by much). However, the angle it makes with the integration ($\approx \frac{2}{9}\pi$) lies outside the $\alpha$ range of $K_a$. The value of $K_a$, assuming its kernel function is the trapezium, is $\frac{2}{3}$ (in the $\beta$ range, the value of the trapezium function is $1 - \frac{\Delta-\alpha}{\beta}$). The value of $K_d(S)$ will range from 1.0 at the center of the crossing, to 0.0 at the ends. The maximum represented weight will range from $\frac{2}{3}$ to 0.0.

After compensation (in Figure 10.1b) the segment weight of segments in the horizontal bundle are 0.0, as they have all been completely represented. The diagonal segments, having at best only been partially represented, will have a segment weight of $\frac{1}{3}$ near the crossing, gradually increasing to 1.0 outside the range of the abstracted path.

Now that we have formalized the concept of compensation, we can use it in our implementation.



**(a)** Before integration and compensation



**(b)** After integration and compensation

**Figure 10.1:** An integration path (dashed black line, red area) representing a bundle of segments (solid black lines).

## 10.1 APPLIED COMPENSATION

Replacing the manual seed selection method by the method proposed in Section 9.4 we can automate the abstraction of the dataset with the method in Listing 10.1, taking as input a segment set `SS` and a number of paths `n`.

```
function abstract(SS, n):
paths := []
for path := 0 to n-1 do:
    seed := determine_seed(SS)
    #seed contains one point and two directions.
    paths[path] := shaped_kernel_int(SS, seed)
    SS := compensate(SS, paths[path])
end
return paths
```

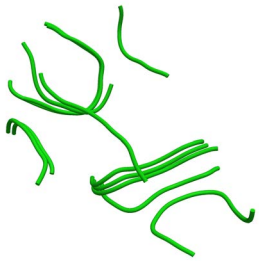
**Listing 10.1:** Integrate and compensate.

This method generates an abstracted path from a determined seed point, and then compensates the segment weights of the set.

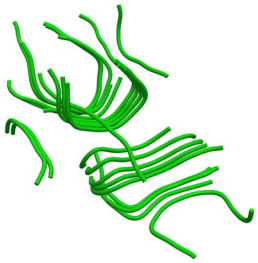
Using this method, several abstracted paths were generated. The generated paths are shown in Figure 10.2. With 50 abstraction paths the structure of the dataset can clearly be seen. The original dataset consists of 6801 paths, so our method shows clear structure at less than 1% of the number of paths.

Also shown, in Figure 10.2e, is a superposition of 50 abstracted paths on the full dataset. It can clearly be seen that the paths represent the dataset.

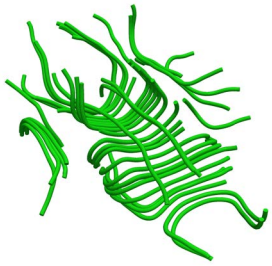
**(a)** Original data



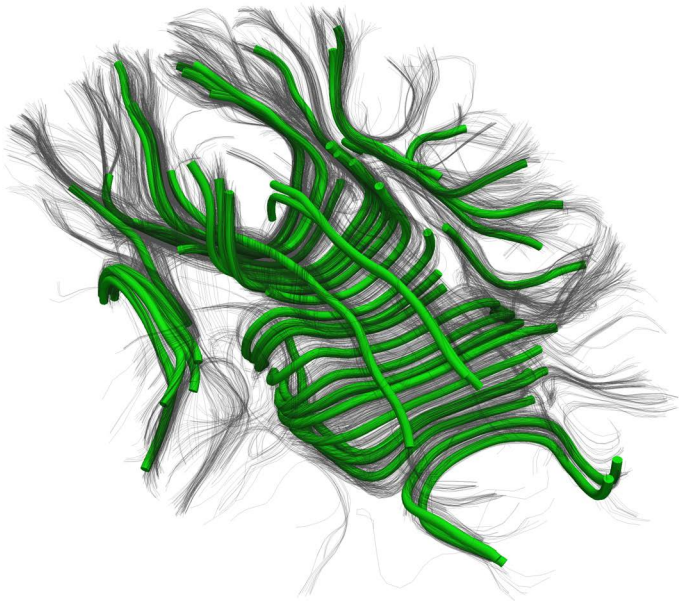
**(b)** 15 paths



**(c)** 30 paths



**(d)** 50 paths

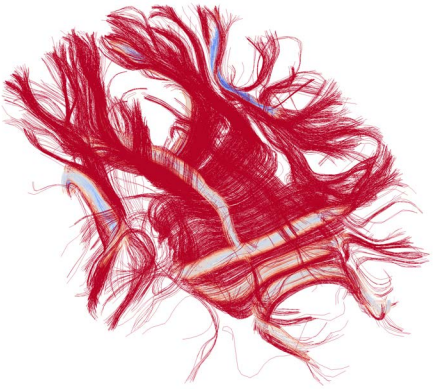


**(e)** Superposition of 50 abstracted paths and dataset

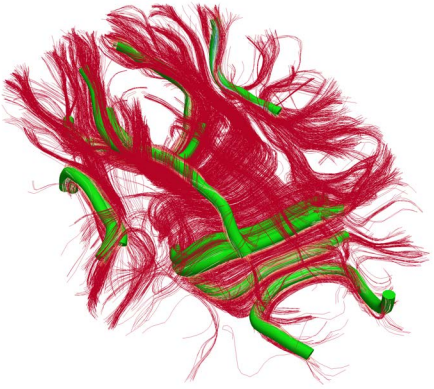
**Figure 10.2:** Abstracted path (green) generation on a slice of DTI tract data(black).

The effect of the compensation method can be tested by coloring a dataset according to the segment weight, which has been done in Figure 10.3. In these images segments with a segment weight of 1.0 are shown in red, going down through white to blue, which denotes 0.0 segment weight. It is clear in this figure that the segments that are compensated are indeed those that participate in the generation of the abstracted paths.

**(a)** Original data



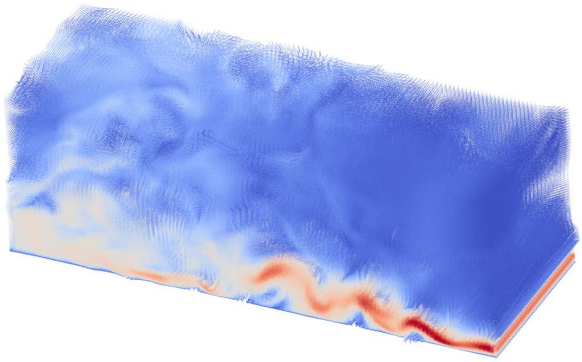
**(b)** Weights after 15 paths



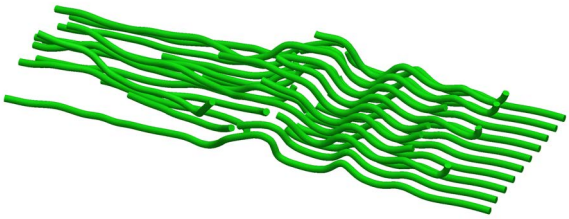
**(c)** The 15 paths transposed on the dataset

**Figure 10.3:** Influence of compensation on a DTI dataset.

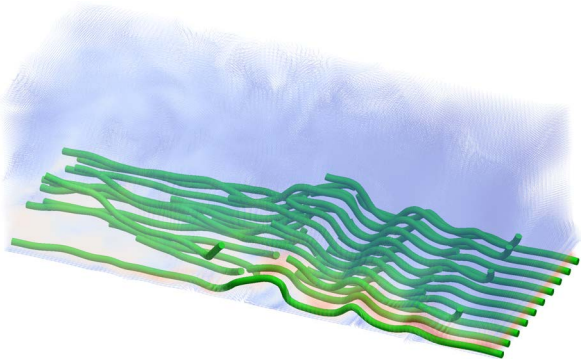
In Figure 10.4 the method is applied to a vector field based segment set: the flow data shown earlier. 30 abstracted paths were generated from a dataset of ~1.000.000 vectors, resulting in coverage of the major flow in the dataset. Note that the equal spacing of the seed points is not the result of a manual restriction, but rather an effect caused by the size of the location based shaped kernel. The distance between the seed points is slightly larger than the value of $\alpha$ of that kernel.



**(a)** Original segment set



**(b)** 30 abstraction paths



**(c)** Original segment set superimposed on 30 abstraction paths

**Figure 10.4:** Compensating integration applied to a vector field based segment set.

# 11

# RESULTS: ABSTRACTED PATH VISUALIZATION

Now that we have a method that can generate abstracted paths from a dataset, we can start exploring the various visualization possibilities this provides.

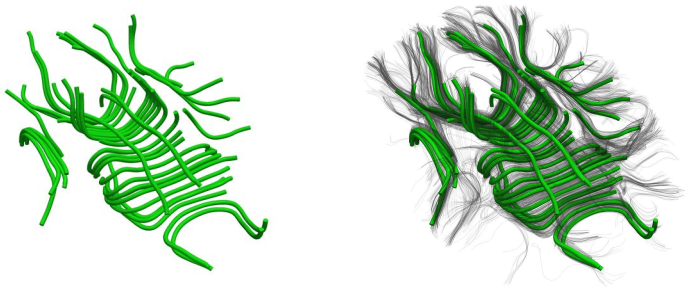
## 11.1 WHOLE DATASET REPRESENTATION

The abstracted paths can be used to give insight in the global structure of a dataset. This is done by either showing just the abstracted paths, or by combining the abstracted paths with a visualization of the dataset. Due to the much smaller number of abstracted paths, they can also be used for navigating a large dataset, so that this dataset only has to be drawn once the orientation is as desired. This makes navigating a large dataset easier and faster.

Due to the distance based kernels, abstracting paths also have the tendency to avoid each other (except near crossing fibers), further decreasing the visual complexity and occlusion in the visualization.

### 11.1.1 *Stream Tubes*

A basic visualization has been used in the previous chapter, showing a tube on the abstracted path (see Figure 11.1a). If the dataset is semi-transparently superimposed, we see that the structure of the dataset becomes clearer, as the abstracted paths provide depth cues through (partial) occlusion (Figure 11.1b).



**(a)** 50 tubes **(b)** 50 tubes plus dataset

**Figure 11.1:** Basic tube visualization.

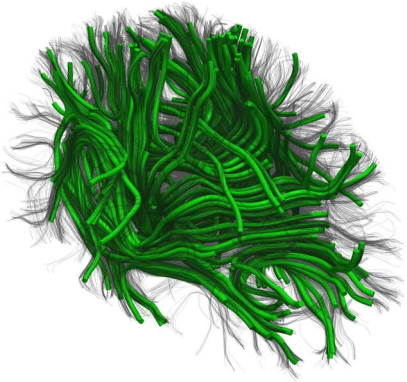
When we apply this method to an entire DTI dataset of 150.000 fibers, the improvement of the abstracted paths visualization over regular whole dataset visualization is clear. We first filter the dataset, so all tracts shorter than 2cm are deleted. This

leaves 20.000 tracts. Then 200 abstracted paths are calculated. The results can be seen in Figure 11.2.

Where the naive visualization is merely a black blob, superimposing the abstracted paths results in an image with visible structure.

**(a)** Full dataset

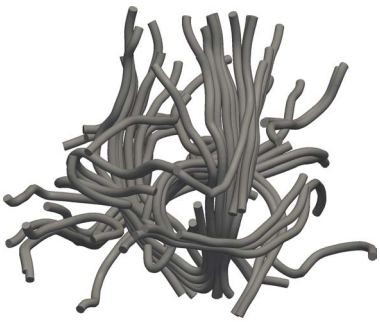
**(b)** Filtered dataset



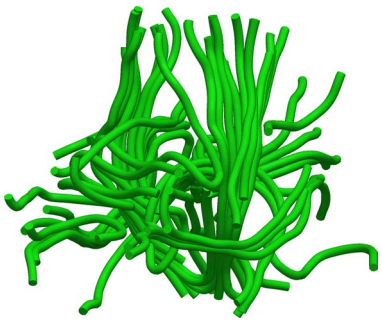
**(c)** 200 abstracted tracts and filtered dataset

**Figure 11.2:** Basic tube visualization on a 20.000 fiber DTI dataset.

The dataset was filtered to decrease the time taken to calculate the abstraction paths. Figure 11.3 shows the difference between the result of a filtered dataset and an unfiltered one. Note that while the results are not identical, the abstraction shows approximately the same structure.



**(a)** Filtered dataset



**(b)** Unfiltered dataset

**Figure 11.3:** Comparison of the result of abstraction using a filtered and unfiltered dataset.

### 11.1.2 *Scaled Stream Tubes*

Stream tubes can be modified to have variable width. If we, for each integration step, register the sum of weights assigned to segments, we get a metric that indicates the amount of segments represented by that integration step. An integration step that represents a high number of segments is likely to be important in the abstraction. Therefore we show this metric in the width of the stream tube, making important parts of an abstraction path wider and thus more visible. By also varying the color, we make the difference even clearer. The color map used in the following sections, from lowest to highest representing weight, is shown in Figure 11.4.

**Figure 11.4:** Blue to Red color map, indicating low (left) to high (right) value.

Applying these two techniques results in the visualization as shown in Figure 11.5 for the DTI dataset, and Figure 11.6 for the flow dataset.

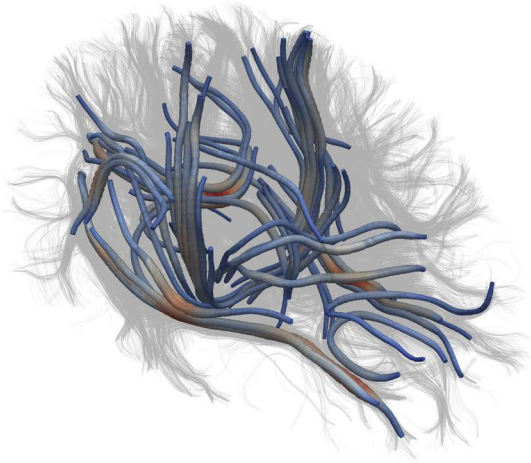
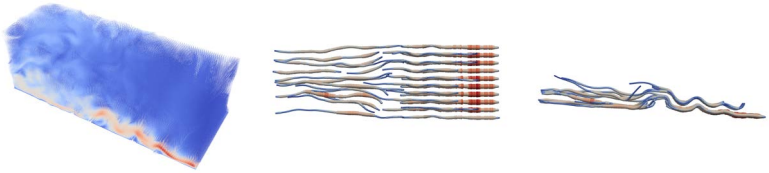


**Figure 11.5:** Visualization of a large DTI dataset, using scaled stream tubes, colored and scaled by representing segment weight.

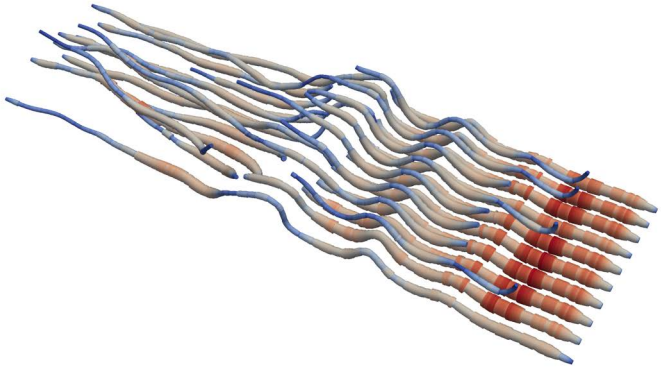
## 11.2 CONTEXT

The abstracted paths can also be used to provide context. If for example a small set of fibers is of interest, visualization of only these fibers can lead to confusion, as it is not clear where they fit in the dataset. This has for example been done by showing an MRI slice behind the tracts, as in Figure 11.7 [53].

While useful, an MRI slice does not give context with regards to the actual tracts. We can use abstracted paths to give this context, as seen in Figure 11.8



**(a)** Original dataset **(b)** Top view **(c)** Side view



**(d)** 30 abstraction paths, scaled and colored by representing segment weight

**Figure 11.6:** Visualization of a large vector field based segment set, using scaled stream tubes, colored and scaled by representing segment weight.

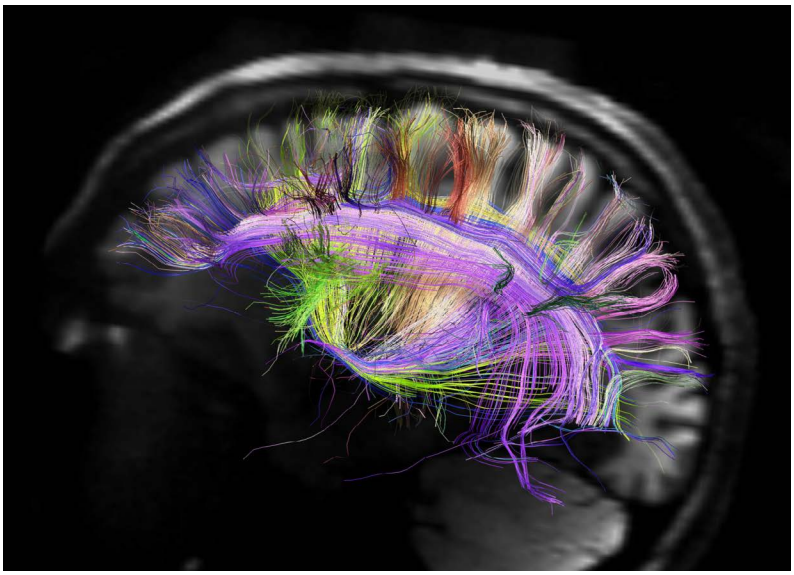


**Figure 11.7:** Giving context to fiber tracts using an MRI slice.

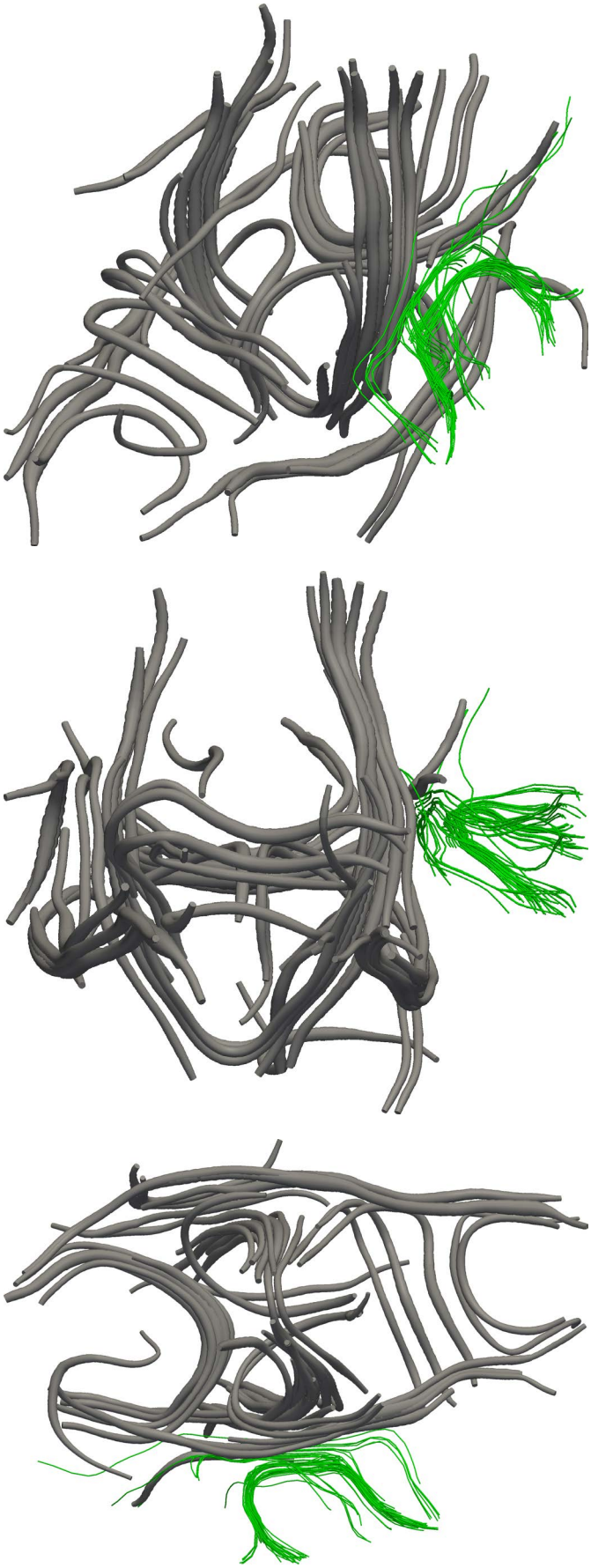


**Figure 11.8:** Giving context to a small selection of data (green) with 50 abstracted paths (gray).

# 12 DEVELOPMENT AND IMPLEMENTATION

The implementation of the method has been presented rather matter-of-factly. It was however preceded by a large number of prototypes. Furthermore, some details of implementation were not discussed, as they do not directly influence the method. They do however influence the implementation, so they will be discussed here.

## 12.1 PROTOTYPES

Firstly a 2D version was implemented. This version did not use the kernel function approach, but a method inspired by integration over vector fields.

Instead of a regular vector field, which is a set of points with one corresponding vector, each point would be assigned a number of vectors.

During integration, an angle based shaped kernel is applied to the four nearest neighbors of the current integration point, and each of the resulting vector sets is summed. This way an angle weighed average of the vectors in a point is calculated.

The value in a certain point between these grid points is then approximated using bilinear interpolation on the resulting vectors.

This method yielded favorable results, with abstraction paths running over important bundles. One such abstraction is shown in Figure 12.1. In this figure the dataset, shown in black, consists of two copies of a flattened slice of DTI data. The second copy was transposed and rotated to create a more complex dataset, with more intersections.

This grid based implementation was then adapted to integrate over 3D datasets. While this yielded some results, it quickly became apparent that the reduction to a grid introduced a large number of grid artifacts.

Figure 12.2 shows the abstraction paths and the weight they represent. The rapid transition between high and low represented weight is caused by interference from the grid. These artifacts reduce the usefulness of the visualization, and make using a stop criterion impossible.

While these issues were already present in the first prototype, they did not influence results as much due to the 2D nature of the dataset.

The integration method that was shown in the previous chapters was the result of the adaptation of this grid based method to one

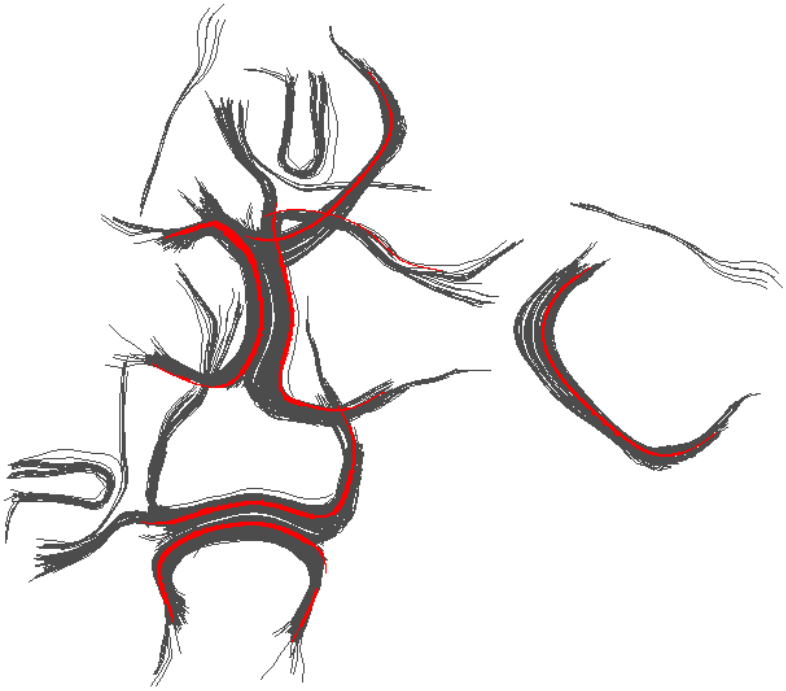
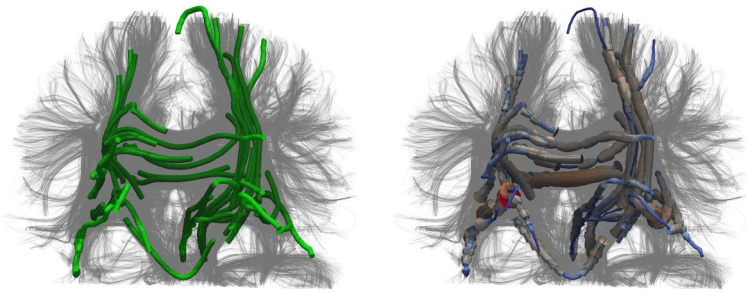


**Figure 12.1:** Results of the first prototype (red), applied to a DTI based flattened dataset (black).



**(a)** Abstraction paths

**(b)** Scaled and colored by representing weight

**Figure 12.2:** Results of the first method, applied to a 3D DTI dataset.

that did not use a grid, in order to remove artifacts and give a more accurate representation.

The same settings were used to generate Figure 12.3, but using the presented method. A clear improvement is seen, as the distribution of representing weight is much more even.

## 12.2 DISTANCE BASED KERNEL OPTIMIZATION

The distance based kernel, when naively implemented, has for a dataset of $n$ segments a time complexity of $\mathcal{O}(n)$. As this is something that has to be calculated for each step of integration, optimization of this method yields quick improvement.

To achieve this, a preprocessing step is introduced. In this step a regular grid is created, for each segment the cell or cells of the

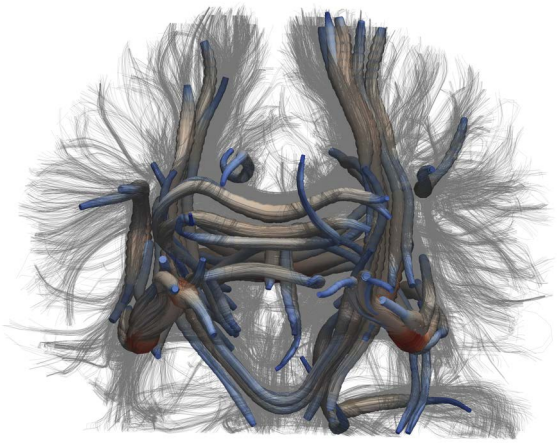


**Figure 12.3:** Results of the non grid-based method.

grid in which they lie is recorded. With this data structure in place, we can speed up the distance based kernel.

We do this by calculating in advance the maximum range of the kernel (in the case of a trapezium kernel: $\alpha + \beta$). When calculating a distance based kernel at a point $\mathbf{p}$, in a grid with spacing $d$, we know that only certain cells can lie within the range of the distance based kernel.

As the grid is regular, we can easily calculate the indices $x$ and $y$ (and, in a 3D case, $z$) of the grid point $g_{x,y}$ nearest to $\mathbf{p}$. We can then calculate $m = \lceil \frac{\alpha+\beta}{d} \rceil$. The maximum range of influence of the kernel can now be calculated as the cells $g_{i,j}$, for $x - m \leq i \leq x + m$ and $y - m \leq j \leq y + m$. This can be done in linear time, and reduces the runtime by a large constant factor (of the order of the ratio between total number of grid points and $m^2$, or $m^3$ in 3D).

All grid cells that fall entirely outside this range will have the weight of their segments set to zero. Other segments are not influenced. This greatly reduces the number of segments that have to be checked against the distance based kernel. Figure 12.4 shows the result of cell elimination. All segments that are not in a blue cell have their weight set to zero for this kernel.

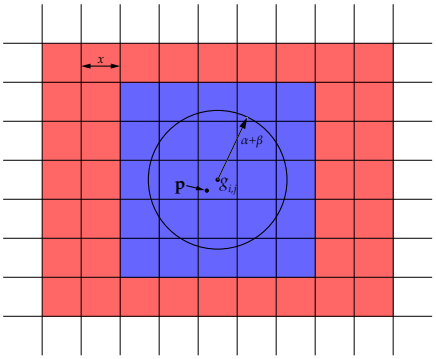



**Figure 12.4:** Results of basic cell elemination. Segments in red cells are eliminated in constant time, while segments in blue cells need to be checked against the kernel function. In this case $m = 2$.

The method was implemented in Java 1.7. the source code can be found on GitHub [23]. The source has been released under Creative Commons Attribution-ShareAlike 3.0 Unported [15].

The previously shown abstracted datasets were generated on a i7-3630QM CPU. The implementation runs mostly on a single core, with the exception of multi-threaded processes in the Java Virtual Machine.

Three datasets were timed: the filtered and unfiltered DTI dataset of respectively 20.000 and 150.000 tracts (consisting of 677.180 and 1.475.120 segments), as well as the fluid flow data of 998.400 segments. The results are shown in Table 12.1.

| Dataset | Preprocessing | Paths | Path generation |
|---|---|---|---|
| Filtered DTI | 9.2 seconds | 25 | 41.8 seconds |
| | | 50 | 82.6 seconds |
| Unfiltered DTI | 19.3 seconds | 25 | 63.8 seconds |
| | | 50 | 121.6 seconds |
| Flow data | 29.2 seconds | 25 | 153.3 seconds |
| | | 50 | 251.8 seconds |

**Table 12.1:** Runtime over various datasets.

Visualization of the abstracted paths was done using the ParaView software package [1], which makes use of the Visualization Toolkit (VTK) [2]. Datasets were exported from a custom integration tool to the VTK file format, and visualized using raw or tube visualizations in ParaView.

# 13 FUTURE WORK

This paper has discussed the basic principle of shaped kernel integration and field compensation. While the method has been thoroughly explored, there still exist some subjects that deserve further attention.

## 13.1 DENSITY PREFERENCE

Currently only the seed selection method has a preference for very dense bundles. While a dense bundle of segments has a large influence on the direction of integration, a small but still significant bundle running at a small angle to the large bundle may result in the integration path leaving the large bundle and being terminated.

To solve this, a preference for dense collinear sets of segments surrounding the integration path could be introduced. During integration we calculate the next integration step. Then, in stead of proceeding along that step, we calculate a number of possible steps, each with a small deviation to the originally calculated step. Then, for each of these possible steps, we calculate the following step. Each of these steps has a certain significance (recall Figure 9.7). The point which has the step with the highest significance is then selected as the actual integration step.

This method would improve the behavior of shaped kernel integration. A large computational overhead is however introduced, as for each step we calculate a number of next steps, instead of just one.

## 13.2 CACHING

As each integration step is highly local, we can speed up the method by introducing caching in the integration method, especially during the application of the distance based shaped kernel. While the grid based data structure introduced in Section 12.2 results in a large speedup, a further improvement can be made by caching the results, as it is likely the next step will do the same, or a similar lookup.

Further caching can be introduced in the selection of a seed point. A large number of kernels are applied each time a new seed point is selected, and while some of these have to be recomputed to account for the changes in the compensated segment set, most will not have changed.

For seed selection a regular square grid is used. While this grid works for a small dataset, it will present a challenge when used with large datasets. The grid will either not have a high enough resolution to provide accurate seed positions, or have such a number of grid points as to be computationally expensive.

To alleviate these problems, a different data structure may be used. Using a quad- or octree-like structure as a search grid allows for increasingly small location based kernels to be applied, first looking for large areas with dense bundles and then refining the search in interesting regions.

# 14

# CONCLUSION

This paper has proposed a method to create an abstracted visualization of a dense line dataset, using only a fraction of the number of lines. This visualization represents the main bundles that occur in the dataset.

The abstracted paths in the visualization do not correspond with any single line of the dataset, but rather follow large bundles of collinear line segments. This results in an abstracted map that can be used in a number of ways.

Firstly it can be used as a stand-alone map, showing the essence of the dataset (see Figure 14.1). Large-scale structure is shown in lieu of detailed data.

This visualization can be used to get a rough impression of the data, giving the user a feel for the structure of the lines. Furthermore it can be used to look for interesting points in the dataset, that can for example be used for manual seed point selection, or the selection of a region of interest.

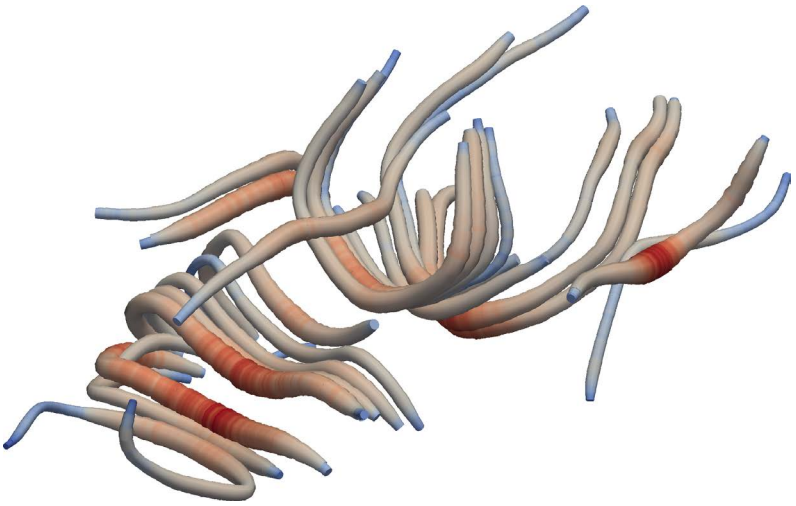


**Figure 14.1:** Raw visualization of the abstracted paths.

The abstracted paths can also be used as a context. This context can then be used to further tune the eventual visualization, by for example showing the location of major structures. It can also be used, as shown in Figure 14.2 to reduce visual complexity outside the region of interest. A number of curves were selected and are shown as stream tubes, while the rest of the dense line dataset is represented by the abstracted paths.

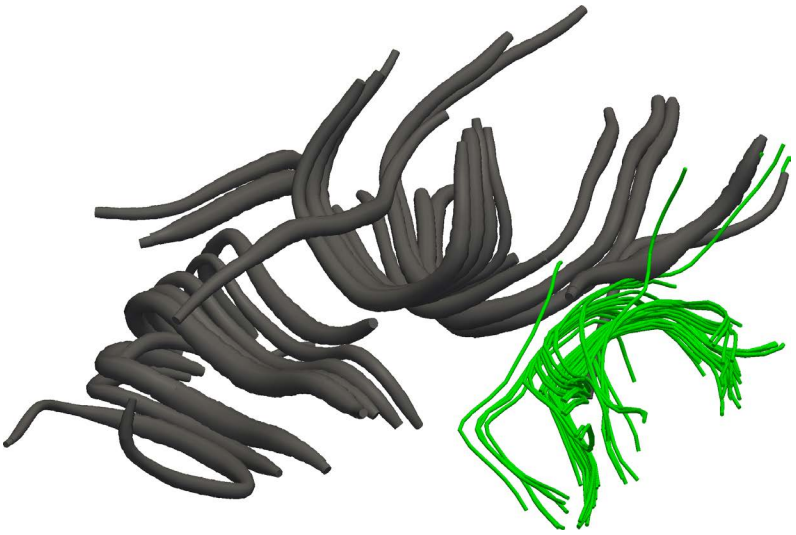


**Figure 14.2:** Using the abstracted paths to provide context to a small number of selected lines.

Finally, due to the occluding properties of the tube visualization, a set of abstracted paths can be used to give depth cues for a dense line dataset. In Figure 14.3 a number of abstracted paths are shown as scaled stream tubes, along with a semitransparent visualization of the dense line data as black lines. The occlusion caused by the scaled stream tube visualization provides depth hinting in important areas. This effect reduces the visual complexity of the visualization.

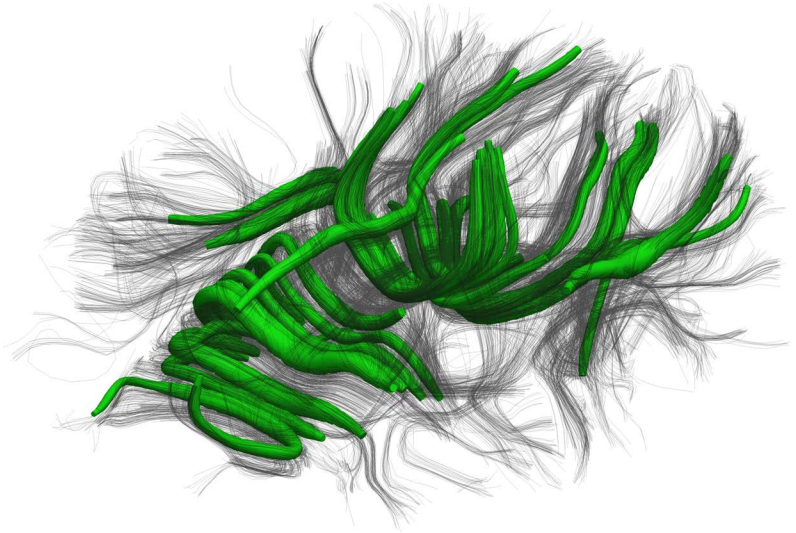


**Figure 14.3:** Using the abstracted paths as depth cue.

[1] Paraview - open source scientific visualization. http://www.paraview.org. [Accessed: 20 November 2012].

[2] Vtk - the visualization toolkit. http://www.vtk.org. [Accessed: 20 November 2012].

[3] OpenFlights Data. http://openflights.org/data.html, january 2012. [Accessed: 20 November 2012].

[4] Francisco Aboitiz, Arnold B. Scheibel, Robin S. Fisher, and Eran Zaidel. Fiber composition of the human corpus callosum. *Brain Research*, 598(1-2):143–153, 1992. ISSN 0006-8993. doi: 10.1016/0006-8993(92)90178-C. URL http://www.sciencedirect.com/science/article/pii/000689939290178C.

[5] A.W. Anderson. Measurement of fiber orientation distributions using high angular resolution diffusion imaging. *Magnetic Resonance in Medicine*, 54(5):1194–1206, 2005.

[6] Deepak Antony. Visualization - diffuse tensor volumes. http://www.deepakantony.com/2011/07/14/visualization-diffuse-tensor-volumes/. [Accessed: 22 January 2013].

[7] R. Aris. *Vectors, tensors and the basic equations of fluid mechanics*. Dover publications, 1990.

[8] Lars Backstrom. Anatomy of facebook. https://www.facebook.com/notes/facebook-data-team/anatomy-of-facebook/10150388519243859, November 2011. [Accessed: 30 November 2012].

[9] PJ Basser, J Mattiello, and D LeBihan. Diagonal and off-diagonal components of the self-diffusion tensor: their relation to and estimation from the nmr spin-echo signal. *Proc. 11th Annu. Meet. SMRM, Berlin*, 1:1222, 1992.

[10] Paul Butler. Visualizing friendships. https://www.facebook.com/notes/facebook-engineering/visualizing-friendships/469716398919, 12 2010. [Accessed: 30 November 2012].

[11] J. Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679–698, 1986.

[12] H.Y. Carr and E.M. Purcell. Effects of diffusion on free precession in nuclear magnetic resonance experiments. *Physical Review*, 94(3):630, 1954.

[13] Cariisa J. Cascia, Guido Gerig, and Joseph Piven. Diffusion tensor imaging: Application to the study of the developing brain. *Journal of the American Academy of Child & Adolescent Psychiatry*, 46(2):213 – 223,

2007. ISSN 0890-8567. doi: 10.1097/01.chi.0000246064.93200.e8. URL http://www.sciencedirect.com/science/article/pii/S0890856709618293.

[14] M.K. Chung, N. Adluru, K.M. Dalton, A.L. Alexander, and R.J. Davidson. Scalable brain network construction on white matter fibers. In *SPIE Medical Imaging*, pages 79624G–79624G. Citeseer, 2011.

[15] Creative Commons. Attribution-sharealike 3.0 unported. http://creativecommons.org/licenses/by-sa/3.0/.

[16] T.E. Conturo, N.F. Lori, T.S. Cull, E. Akbudak, A.Z. Snyder, J.S. Shimony, R.C. McKinstry, H. Burton, and M.E. Raichle. Tracking neuronal fiber pathways in the living human brain. *Proceedings of the National Academy of Sciences*, 96(18):10422–10427, 1999.

[17] P. Douek, R. Turner, J. Pekar, N. Patronas, D. Le Bihan, et al. MR color mapping of myelin fiber orientation. *Journal of computer assisted tomography*, 15(6):923, 1991.

[18] G. Dougherty. Image analysis in medical imaging: recent advances in selected examples. *Biomedical imaging and intervention journal*, 6(3):32, 2010.

[19] A. Einstein. *Investigations on the Theory of the Brownian Movement*. Dover publications, 1956.

[20] M.H. Everts. *Visualization of Dense Line Data*. University Library Groningen, 2011.

[21] R.D. Fields. White matter matters. *Scientific American*, 298 (3):54–61, 2008.

[22] Els Fieremans. *Validation methods for diffusion weighted magnetic resonance imaging in brain white matter*. PhD thesis, Ghent University, 2008.

[23] GitHub. Abstraction of dense line data using shaped integration and field compensation. https://github.com/DZittersteyn/DLD_Abstract.

[24] S. Hofer, J. Frahm, et al. Topography of the human corpus callosum revisited? comprehensive fiber tractography using diffusion tensor magnetic resonance imaging. *Neuroimage*, 32(3):989–994, 2006.

[25] Colin Holmes, Paul Thompson, and Arthur Toga. Mapping growth rates in brain tumors. http://www.loni.ucla.edu/~thompson/NPACI/vis.html. [Accessed: 22 January 2013].

[26] B. Jeurissen, A. Leemans, D.K. Jones, J.D. Tournier, and J. Sijbers. Probabilistic fiber tracking using the residual bootstrap with constrained spherical deconvolution. *Human brain mapping*, 32(3):461–479, 2010.

[27] Hangyi Jiang, Peter C.M. van Zijl, Jinsuh Kim, Godfrey D. Pearlson, and Susumu Mori. DtiStudio: Resource program for diffusion tensor computation and fiber bundle tracking. *Computer Methods and Programs in Biomedicine*, 81(2):106 – 116, 2006. ISSN 0169-2607. doi: 10.1016/j.

cmpb.2005.08.004. URL http://www.sciencedirect.com/science/article/pii/S0169260705002348.

[28] B. Jobard and W. Lefer. Creating evenly-spaced streamlines of arbitrary density. *Visualization in scientific computing*, 97: 43–55, 1997.

[29] Wall Street Journal. What they know. http://blogs.wsj.com/wtk/. [Accessed: 30 November 2012].

[30] J.W. Kalat. *Biological psychology*. Wadsworth Publishing Company, 2012.

[31] G. Kindlmann. Superquadric tensor glyphs. In *Symposium on Visualization*, pages 147–154. The Eurographics Association, 2004.

[32] JP Kubitschek and PD Weidman. Helical instability of a rotating viscous liquid jet. *Physics of Fluids*, 19:114108, 2007.

[33] P.C. Lauterbur et al. Image formation by induced local interactions: examples employing nuclear magnetic resonance. *Nature*, 242(5394):190–191, 1973.

[34] P.C. Lauterbur et al. Magnetic resonance zeugmatography. *Pure Appl. Chem*, 40(1-2):149–157, 1974.

[35] M. Lazar, D.M. Weinstein, J.S. Tsuruda, K.M. Hasan, K. Arfanakis, M.E. Meyerand, B. Badie, H.A. Rowley, V. Haughton, A. Field, et al. White matter tractography using diffusion tensor deflection. *Human brain mapping*, 18 (4):306–321, 2003.

[36] D. Le Bihan, E. Breton, et al. Imagerie de diffusion in-vivo par résonance magnétique nucléaire. *Comptes-Rendus de l'Académie des Sciences*, 93(5):27–34, 1985.

[37] Hong Luo, Hidehiro Segawa, and Miguel R. Visbal. An implicit discontinuous galerkin method for the unsteady compressible navier-stokes equations. *Computers & Fluids*, 53(0):133 – 144, 2012. ISSN 0045-7930. doi: 10.1016/j.compfluid.2011.10.009. URL http://www.sciencedirect.com/science/article/pii/S0045793011003082.

[38] E. McQuinn, A. Chourasia, JH Minster, and J. Schulze. Glyphsea: Interactive exploration of seismic wave fields using shaded glyphs. In *AGU Fall Meeting Abstracts*, volume 1, page 1351, 2010.

[39] S. Mori, B.J. Crain, VP Chacko, and P. Van Zijl. Three-dimensional tracking of axonal projections in the brain by magnetic resonance imaging. *Annals of neurology*, 45(2):265–269, 1999.

[40] NASA. SciJinks cloud library. http://scijinks.nasa.gov/clouds-gallery. [Accessed: 15 January 2013].

[41] I. Newton. *The correspondence of Isaac Newton*, volume 1. Published for the Royal Society at the University Press, 1959.

[42] Fritz H. Obermeyer. Jenn 3d. http://www.math.cmu.edu/~fho/jenn/index.html. [Accessed: 15 Febuary 2013].

[43] PhD Rand S. Swenson, MD. Atlas of the brain: Structure with functional correlates, online version. http://www.dartmouth.edu/~rswenson/Atlas/index.html, 2009. [Accessed: 3 December 2012].

[44] Ryan Rossi. Visualizing flow around a delta wing. http://www.ryanrossi.com/sv4.php. [Accessed: 17 January 2013].

[45] Iepe Rubingh. Painting reality. http://www.painting.iepe.net/, 2010. [Accessed: 30 November 2012].

[46] E.B. Saff and A.B.J. Kuijlaars. Distributing many points on a sphere. *The Mathematical Intelligencer*, 19(1):5–11, 1997.

[47] A. Telea and J.J. Van Wijk. Simplified representation of vector fields. In *Visualization'99. Proceedings*, pages 35–507. IEEE, 1999.

[48] Alexandru C. Telea. *Data Visualization*. A K Peters, Ltd., 2007. ISBN 1568813066.

[49] Chris Tiee. Nested tori. http://nestedtori.blogspot.nl/2012/11/happy-thanksgiving-from-nested-tori.html, 11 2012. [Accessed: 22 January 2013].

[50] David Trood. The face of humanity. http://www.trood.dk/blog/the-face-of-humanity/, march 2011. [Accessed: 14 November 2012].

[51] J.J. Van Wijk. Image based flow visualization. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 745–754. ACM, 2002.

[52] A. Vilanova, G. Berenschot, and C. Van Pul. Dti visualization with streamsurfaces and evenly-spaced volume seeding. In *Symposium on Visualization*, pages 173–182. The Eurographics Association, 2004.

[53] V.J. Wedeen, D.L. Rosene, R. Wang, G. Dai, F. Mortazavi, P. Hagmann, J.H. Kaas, and W.Y.I. Tseng. The geometric structure of the brain fiber pathways. *Science*, 335(6076): 1628–1634, 2012.

[54] T. Weinkauf, H. Theisel, H.C. Hege, and H.P. Seidel. Boundary switch connectors for topological visualization of complex 3d vector fields. In *Data Visualization*, pages 183–192, 2004.

[55] D. Weinstein, G. Kindlmann, and E. Lundberg. Tensorlines: Advection-diffusion based propagation through diffusion tensor fields. In *Proceedings of the conference on Visualization'99: celebrating ten years*, pages 249–253. IEEE Computer Society Press, 1999.

[56] P.L. Williams et al. *Gray's anatomy*, volume 378. Churchill livingstone London, 1995.

[57] S. Zhang, G. Kindlmann, and DH Laidlaw. Diffusion tensor mri visualization. *Visualization handbook*, pages 327–340, 2004.