UNIVERSITÄT LEIPZIG

INSTITUT FÜR INFORMATIK (IfI)

Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik
Abteilung für Bild- und Signalverarbeitung

Image Space Tensor Field Visualization Using a LIC-like Method

Diplomarbeit

Leipzig, 28. Mai 2009

vorgelegt von
Sebastian Eichelbaum
Diplom Informatik

Betreuender Hochschullehrer: Prof. Dr. Gerik Scheuermann
Universität Leipzig
Institut für Informatik
Abteilung für Bild- und Signalverarbeitung

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Leipzig, 28. Mai 2009

Sebastian Eichelbaum

Acknowledgements

At first, I want to thank Mario Hlawitschka for his great help in every situation during this work. As my former student assistant supervisor, he led me to the interesting field of medical visualization and gave me very interesting tasks, while always letting me bring in my ideas.

Another big thanks goes to the Abteilung für Bild- und Signalverarbeitung of the University of Leipzig, for giving me the opportunity to gain first sight into scientific work during my student assistant job. Especially Alexander Wiebel always answered questions, helped during problems and gave useful tips. And Prof. Dr. Gerik Scheuermann, I have to thank for making this possible and for never sending me away if I had a question or a problem. Of course, I do not have to miss Prof. Dr. Bernd Hamann for his help with the Paper also relating to this work.

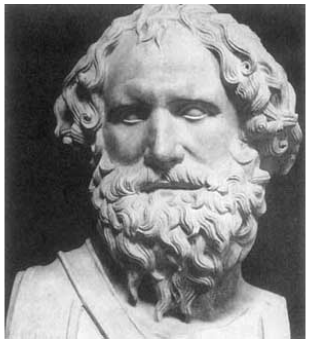
A big thanks belongs to my friends. They never complained if I was busy, depleted and totally unconcencrated, because my thoughts sometimes just rotated around this work. They motivated me whenever I needed it. I can't name all, so I would like to mention especially René and Patrick.

For every time supporting me, being there while everything looked unsolvable and motivating me during the whole work, the biggest thanks goes to my loved girlfriend Beatrice.

Abstract

Tensors are of great interest to many applications in engineering and in medical imaging, but a proper analysis and visualization remains challenging. Physics-based visualization of tensor fields has proven to show the main features of symmetric second-order tensor fields, while still displaying the most important information of the data, namely the main directions in medical diffusion tensor data using texture and additional attributes using color-coding, in a continuous representation. Nevertheless, its application and usability remains limited due to its computational expensive and sensitive nature. We introduce a novel approach to compute a fabric-like texture pattern from tensor fields on arbitrary non-selfintersecting surfaces that is motivated by image space line integral convolution (LIC). Our main focus lies on regaining three-dimensionality of the data under user interaction, such as rotation and scaling. We employ a multi-pass rendering approach to estimate proper modification of the LIC noise input texture to support the three-dimensional perception during user interactions.

Contents



*Archimedes of Syracuse, *287 BC - †212 BC.*
The greatest mathematician of his age.

1 Introduction

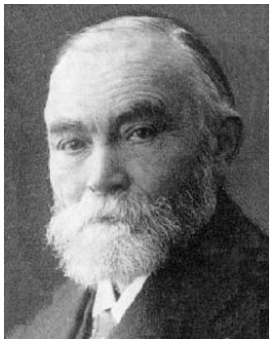
Visualization is not an invention of our time. Far from it! It is as old as science itself. Even Archimedes, one of the most famous Greek mathematicians, is said to be killed by a roman soldier in 212 AC while drawing geometric figures in the sand. Over the centuries scientists, geographers, and engineers improved their methods and began to draw more complex maps, astronomical charts, and data plots. One of the famous images, visualizing multivariate data, is Minard's map from 1861 of Napoleon's invasion of Russia. In the last century, Conrad Zuse opened a totally new way to the scientific world. The Z3 was the first modern computer intended to be used for aerodynamic calculations during World War II. From then, the rapid development of computers took care that the number of data that mathematical models could produce increased tremendously and the demand for better possibilities of information perception led to a rapid development in graphics hardware and software. So, the foundation for visualization, as own discipline, was set in the late 1980's. One of the pioneers was the american Yale professor Edward Rolf Tufte. Edward Rolf Tufte wrote two books [Tuf83, Tuf90], that firstly explained many of the visualization techniques used over centuries.

The push in the development of graphics hardware, last but not least triggered by Hollywood, significantly discovered the potential of visualization in scientific applications. Today, it is possible to do even complex visualizations with standard desktop computers. Engineers design and test cars, planes, or space ships in laboratories with large projection walls, physicians visualize the universe's largest phenomenons like supernova explosions, chemists draw molecular structures, and scientists are able to discover their models and data even in 3D using 3D shutter glasses. Besides this, visualization has many medical applications. Medical re-

searchers use visualization to discover the reasons of deceases like Altzheimer's or multiple sclerosis, moreover, radiologists or surgeons use modern medical imaging methods, like Magnetic Resonance Imaging (MRI), to plan operations.

This diploma thesis contributes to the area of medical visualization by introducing a novel method to render a fabric-like pattern on arbitrary, nonselfintersecting surfaces using line integral convolution in image space. One of the goals is to achieve real time user interaction while still visualizing physical-based features of tensor fields.

First, Chapter 2 introduces the mathematical foundations, needed to get a basic understanding of tensors, tensor fields, and their properties. Then, the magnetic resonance imaging method, as well as diffusion tensor imaging, gets shortly described to show where the data, we are using, comes from. To complete the introductional chapter, the subsequent sections finally describe some very commonly used visualization methods for MRI and Diffusion Tensor Imaging (DTI) data. After the basic theoretical background is disclosed, Chapter 3 shortly introduces the work of other researchers and groups relating our method as well as our motivation and goals, subsequently followed by a mathematically introduction of our method in Chapter 4, with figures in each section to illustrate it. Chapter 5 then shows our method from the implementational point of view. With Chapter 6 and Chapter 7, this diploma thesis gets completed with results, possible applications and a conclusion.



*F. L. Gottlob Frege, *1848 - †1925*
Developed first-order predicate calculus, which is a crucial foundation of computation theory.

2 Background

In this chapter, we will have a look at the theoretical foundations of tensors and second-order tensor fields. In addition to that, later required calculations like fractional anisotropy or eigenvalue decomposition are described. Subsequent to that the data acquisition gets described. An overview is given how the data that is used in this thesis has been acquired.

2.1 Mathematical Foundations

In modern physics *tensors* play an important role. They are used to describe many physical phenomena, like stress and strain in solid materials, or as a basic mathematical concept in general relativity, introduced by Einstein in 1915. Medical imaging is using tensor fields to describe the directional diffusion behavior of water inside the brain. In [WH06], a comprehensive overview of processing and visualization of tensor fields is given.

In this thesis, tensor data plays a fundamental role, since our technique focuses on second-order tensor field visualization. That is why we begin with the mathematical foundations of tensors and some of their properties.

2.1.1 Tensors

The following introduction is based on an overview given by Smirnov et al. [Smi04]. To understand the concept of a tensor we assume an n-dimensional space over the set of real numbers: $V \subseteq \mathbb{R}^n$.

Definition 2.1 (Coordinate System) *Let V be an n-dimensional subspace in $\mathbb{R}^n$. Then, a coordinate system is defined as*

$$\{x^i\}_{i=1\cdots n}.$$

To transform a coordinate system to another system with the same dimension n, lets say $\{\tilde{x}^j\}_{j=1\cdots n}$, we have to apply the rule

$$\tilde{x}^j = \tilde{x}^j(x^1 \ldots x^n). \tag{2.1}$$

By applying the partial differentiation rules to Equation 2.1 it is possible to transform a small displacement dx^i in coordinate system x^i to another coordinate system $\tilde{x}^j$ using

$$d\tilde{x}^j = \frac{\delta \tilde{x}^j}{\delta x^j} dx^j. \tag{2.2}$$

This, furthermore, can be generalized to vectors

$$\tilde{A}^i = \frac{\delta \tilde{x}^i}{\delta x^j} A^j \tag{2.3}$$

or by using common matrix notation:

$$\tilde{A}^i = a_{ij} A^j \text{ with transformation matrix } a_{ij} := \frac{\delta \tilde{x}^i}{\delta x^j}. \tag{2.4}$$

Definition 2.2 (Contravariant Vectors) *Let A be a vector conforming to Equation 2.3. A is then called contravariant vector.*

This transformation rule does not apply to all vectors in the space V. The partial derivatives $\frac{\delta}{\delta x^i}$ are an example for that. They transform with the rule

$$\frac{\delta}{\delta \tilde{x}^i} = \frac{\delta}{\delta \tilde{x}^i} \frac{\delta x^j}{\delta x^j} = \frac{\delta x^j}{\delta \tilde{x}^i} \frac{\delta}{\delta x^j}. \tag{2.5}$$

Analogous to Equation 2.3, we can now generalize this rule to vectors

$$\tilde{A}_i = \frac{\delta x^j}{\delta \tilde{x}^i} A_j. \tag{2.6}$$

order	no. components	common representation
0	1	single scalar value
1	3	three-dimensional vector
2	9	three-times-three matrix

Table 2.1: *Exemplification of some tensor orders and their more commonly known representation.*

Definition 2.3 (Covariant Vectors) *Let A be a vector conforming to Equation 2.6. A is then called covariant vector.*

In Definition 2.3, covariant vectors differ to their notation to contravariant vectors. Covariant vectors have lower indices, whereas contravariant vectors have upper indices.

Now we have the required tools to define the tensor itself.

Definition 2.4 (Tensor) *A tensor $T = T^{j_1,\dots,j_l,\dots,j_s}_{i_1,\dots,i_k,\dots,i_r}$ is now defined as a mathematical object with a dimension n and an order $m = r+s$, the sum of contravariant and covariant indices s and r. Thereby, each index i_k and j_l, goes from 1 to n.*

*Every index of T that complies to the contravariant transformation rule in Equation 2.3 is called **contravariant index**. Analog to this, every index that complies to the rule in Equation 2.6 is called **covariant index** of T. Those indices describe the **type** (r,s) of a tensor.*

Definition 2.4 also brings some implications along. A tensor is practically described by an array of real numbers that is indexed using m indices. As each of these indices i, j, k, ... is an integer number between 1 and n, a tensor is determined by n^p real number components. As tensors are most commonly seen as generalization of scalars, vectors, and matrices, Table 2.1 gives an overview of some very common three-dimensional tensors and their representation in commonly used mathematical terms. As a special case, there are tensors whose transformation matrix, from Equation 2.4, satisfies this relation with the Kronecker delta tensor from Definition 2.5

$$a^k_i a^k_j = \frac{\delta \tilde{x}^k}{\delta x^i}\frac{\delta \tilde{x}^k}{\delta x^j} = \frac{\delta x^i}{\delta x^j} = \delta_{ij}. \tag{2.7}$$

Definition 2.5 (Kronecker Delta Tensor) *Let δ be a tensor of second order. Then, the Kronecker delta tensor is defined by*

$$\delta_{ij} = \frac{\delta x^i}{\delta x^j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{else.} \end{cases}$$

Definition 2.6 (Cartesian Tensor) *Let T be a tensor and a_{ij} its transformation matrix. If Equation 2.7 is applicable to a_{ij}, T is called cartesian tensor.*

Immediately following from Definition 2.6, in a cartesian coordinate system, the equation

$$\frac{\delta \tilde{x}^i}{\delta x^j} = \frac{\delta x^j}{\delta \tilde{x}^i} \tag{2.8}$$

applies and the transformation rules in Equation 2.3 and Equation 2.6 are, as well as the transformation matrices, equal, thus vanishing the difference between covariant and contravariant indices.

2.1.2 Tensor Calculus

Now, that the tensor is defined formally, it is possible to define operations on two tensors or tensors and scalars. Using group theory, it is possible to define a tensor space $\mathscr{W}$ with the operations in Definition 2.7 over the field of real numbers, as described in [Ibe95]. In the following definitions, we assume n to be the dimension and m to be the order of the tensors.

Definition 2.7 (Tensor calculus) *Let S and T be two tensors, both with the same type (r,s), where $m = r+s$ and a tensor U with an arbitrary type (u,v) and order. Furthermore, let x be scalar. Then, the following operations are defined:*

operation	*definition*	*resulting type*
Addition	$R^{j_1 \cdots j_s}_{i_1 \ldots i_r} = S^{j_1 \cdots j_s}_{i_1 \ldots i_r} + T^{j_1 \cdots j_s}_{i_1 \ldots i_r}$	(r,s)
Scalar Multiplication	$R^{j_1 \cdots j_s}_{i_1 \ldots i_r} = x \cdot S^{j_1 \cdots j_s}_{i_1 \ldots i_r}$	(r,s)
Dyadic Product	$R^{j_1 \cdots j_s, j_{s+1} \cdots j_{s+s'}}_{i_1 \ldots i_r, i_{r+1} \ldots i_{r+r'}} = S^{j_1 \cdots j_s}_{i_1 \ldots i_r} \otimes U^{l_1 \ldots l_{s'}}_{k_1 \ldots k_{r'}}$	*(r+u, s+v)*

Now we assume $\mathscr{W}$ to be the set of all tensors with arbitrary type and order, as well as $\mathscr{W}^* \in \mathscr{W}$ to be the set of all tensors with the same type. The defined operations (addition, scalar multiplication, and dyadic product) now have the following properties:

Addition, $\mathscr{W}^* \times \mathscr{W}^* \to \mathscr{W}^*$:

(f1) $\forall A,B \in \mathscr{W}^* : A+B \in \mathscr{W}^*$ (Closure)

(f2) $\forall A,B,C \in \mathscr{W}^* : A+(B+C) = (A+B)+C$ (Associativity)

(f3) $\forall A,B \in \mathscr{W}^* : A+B = B+A$ (Commutativity)

(f4) $\forall A \in \mathscr{W}^* : A+O = O+A = A$ (Identity)

(f5) $\forall A \in \mathscr{W}^* : A+(-A) = (-A)+A = O$ (Inverse)

Scalar Multiplication, $\mathbb{R} \times \mathscr{W} \to \mathscr{W}$:

(g1) $\forall x,y \in \mathbb{R} \wedge \forall A \in \mathscr{W} : x \cdot (y \cdot A) = (x \cdot y \cdot A)$ (Associativity)

(g2) $\forall x,y \in \mathbb{R} \wedge \forall A,B \in \mathscr{W} : (x+y) \cdot A = x \cdot A + y \cdot A \wedge x \cdot (A+B) = x \cdot A + x \cdot B$ (Distributivity)

(g3) $\forall A \in \mathscr{W} : A \cdot 1 = 1 \cdot A = A$ (Identity)

Dyadic Product, $\mathbb{R} \times \mathscr{W} \to \mathscr{W}$:

(d1) $\forall x \in \mathbb{R} \wedge \forall A,B,C \in \mathscr{W} : x \cdot (A \otimes B) = (x \cdot A) \otimes B = A \otimes (x \cdot B) \wedge A \otimes (B \otimes C) = (A \otimes B) \otimes C$ (Associativity)

(d2) $\forall A \in \mathbb{W} \wedge \forall B,C \in \mathscr{W}^* : A \otimes (B+C) = A \otimes B + A \otimes C$ (Distributivity)

Definition 2.8 (Tensor space) *Let $\mathscr{W}$ be the set of all tensors with arbitrary type and $\mathscr{W}^* \in \mathscr{W}$ the set of all tensors with the same type. The algebraic structure $(\mathscr{W}, +, \cdot, \otimes)$ is called tensor space over the field of real numbers, when the operations $+, \cdot$ and $\otimes$ comply to the rules (f1) to (f5), (g1) to (g3) and (d1) to (d2).*

The operations and Definition 2.8 differ from vector spaces in an important point: the addition operator is only defined over the real subset $\mathscr{W}^* \subset \mathscr{W}$. So, only the set $\mathscr{W}^*$ builds a vector space over the field of real numbers $\mathscr{K}$.

2.1.3 Second-Order Tensor Fields

In this thesis, we focus on second-order tensors only. That is why the general tensor definition from Section 2.1.1 will be used to show some special aspects of second-order tensors and second-order tensor fields. First, we define second-order tensors in general and then, the special case of symmetric second-order tensors.

Definition 2.9 (Second-Order Tensor) *Let T be a tensor. Then T is called second order tensor, if it is one of either types $(2,0)$, $(1,1)$, or $(0,2)$. In cartesian space, the difference between contravariant and covariant indices disappears, so the tensor T_{ij} is represented by n^2 values.*

Definition 2.10 (Second-Order Tensor as Matrix) *Let T_{ij} be an arbitrary second order tensor in an n^{th}-dimensional cartesian space. Then, the tensor can be interpreted as $n \times n$-matrix with all its properties and rules.*

In the following, we interpret the second-order tensors as matrices, since the used coordinate system is irrelevant for our purposes. As Definition 2.10 implies, we also rely on the common matrix operations and rules. One common matrix operation we will use with second order tensors later on is matrix transponation:

$$A = \left(a_{ij}\right) \rightarrow A^T = \left(a_{ji}\right) \text{ with } A^{n\times n}. \tag{2.9}$$

With the matrix toolset, it is now possible to define several properties for second-order tensors in a similar way they are defined for matrices.

Definition 2.11 (Second-Order Symmetric Tensor) *Let T_{ij} be a second order tensor. Then, it is called symmetric, if and only if $T_{ij} = T_{ij}^T = T_{ji}$.*

As every three-times-three matrix, arbitrary second-order tensors can be seen as the sum of a symmetric and an anti-symmetric part:

$$T = \underbrace{\frac{1}{2}(T + T^T)}_{\text{symmetric}} + \underbrace{\frac{1}{2}(T - T^T)}_{\text{antisymmetric}}. \tag{2.10}$$

For an arbitrary second-order tensor T, the characteristic equation encodes its eigenvalues, its determinant and the tensor's trace. The characteristic equation is then defined as

$$\det|T - \lambda\,\delta_{ii}|. \tag{2.11}$$

The solutions of the characteristic equation are the tensor's eigenvalues. In second-order case, the Kronecker tensor δ_{ii} equals the identity matrix of the same dimension n as the tensor.

Definition 2.12 (Trace) *Let T be a second order tensor. The trace of T then is defined as the sum of its diagonal elements:*

$$tr(T) = \sum_{0<i<n} T_{ii}.$$

Definition 2.13 (Determinant) *Let T be a second order tensor. Then the value* $\det(T)$*, or $|T|$, is defined by the Leibnitz formula (Leibnitz equation)*

$$\det(T) = \sum_{\sigma \in S_n} sgn(\sigma) \prod_{i=1}^{n} T_{i,\sigma(i)}$$

and is called determinant of T.

Definition 2.13 uses the Leibnitz formula to define the determinant for arbitrary $n \times n$-matrices, where sgn is the sign function of permutations in the permutation group S_n. In our case, a fast method to calculate the determinant of three-times-thee matrices is the *rule of Sarrus*. It is a special case of the Leibnitz formula and since we assume our second order tensors to be $n \times n$-matrices, is defined for three-dimensional second-order tensors by

$$\det(T) = aei + bfg + cdh - gec - hfa - idb. \tag{2.12}$$

with T defined in matrix notation:

$$T = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

Definition 2.14 (Eigenvalues) *Let T be a second-order tensor and let $e_1, \dots e_n \in \mathbb{R}^n$ be an orthonormal basis. Furthermore, let R be a rotation matrix. Then, T can be written as*

$$T = R \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix} R^T,$$

where $\lambda_1 \dots \lambda_n$ are called eigenvalues. As convention the eigenvalues are sorted: $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$.

In cartesian case, the rotation matrix R is defined by the n unit-vectors, one for each dimension, e_i with $i \in [1,n]$.

$$R = E = \{e_1, e_2, e_3\}. \tag{2.13}$$

The eigenvalues can be calculated using the characteristic polynom in Equation 2.11, from which they are the solutions. Using the eigenvalues, it is possible to calculate every eigenvector of the tensor t.

Definition 2.15 (Eigenvector) *Let T be a second order tensor of dimension n and $\lambda_1 \dots \lambda_i \dots \lambda_n$ its eigenvalues. Then, the solution vectors $x_0 \dots x_i \dots x_n$ of dimension n of:*

$$(T - \lambda_i R)x_i = 0$$

and are called eigenvectors.

In the cases, where T is a symmetric tensor of dimension n, the eigenvalues $\lambda_1 \dots \lambda_n$ are real and the eigenvectors to different eigenvalues are even orthogonal. Another common property of second-order tensors is *positive definiteness*. A tensor is called *positive definite*, if all eigenvalues $\lambda_1 \dots \lambda_n$ are positive.

With the use of Definition 2.9, it is now possible to extend the definition of second-order tensors to fields.

Definition 2.16 (Second-Order Tensor Field) *Let $S \subseteq \mathbb{R}^n$ be a subset in cartesian space of dimension n and $\mathscr{W}^2$ the set of all second-order tensors with the same dimension. Then, the mapping*

$$f : S \to \mathscr{W}^2$$

is called second-order tensor field of dimension n.

Alternatively, the definition can be modified to symmetric second-order tensors and is then called *symmetric second-order tensor field*. If the tensors in $\mathscr{W}^2$, have real eigenvalues, which is always the case if the tensor field is symmetric, the tensor field can also be described by n (eigen-) vector fields of dimension n and n (eigenvalue-) scalar fields.

2.1.4 Tensor Interpolation

Since the subset $S \in \mathbb{R}^n$ may not be continuous inside some given borders, it is necessary to think about interpolating tensors. The practical importance of tensor interpolation is implied by the discrete nature of tensor field datasets. For scalar fields, it is obvious to use a linear interpolation to calculate the value of every point in a discrete field. In vector fields, component wise trilinear interpolation is most commonly used. But it is not as obvious in second-order (and higher) tensor fields. One possibility is to interpolate the raw data acquired directly from an MRI scanner and therefore calculate the tensor at an arbitrary point (inside the field) by using the interpolated raw data. The computational effort is enormous, especially if thousands of tensors are needed in realtime, which disqualifies this method for our approach, although the results would be the best and most accurate.

Since many approaches to visualize tensor fields, do not rely on the tensor directly, but on some derived quantity, like eigenvalues, eigenvectors, or some anisotropy measures like those in Section 2.1.5, it suggests itself to directly interpolate these quantities. This reduces the interpolation problem to interpolation of scalar- or vector fields. However, this could lead to several problems. Assume two adjacent points p_1 and p_2 in the discrete tensor field f. Furthermore assume, that f is a diffusion tensor field, which means the eigenvectors are orientationless. If two eigenvectors, one at each point p_i, have contrary orientations, the interpolation between these points produces eigenvectors, that are not equal to these eigenvectors, that would be interpolated if the orientation in p_1 and p_2 is the same (apart from orientation difference). This problem gets illustrated in Figure 2.1, where the two vectors v_1 and v_2 get interpolated while ignoring their different orientation. An interpolation without taking care of the vector's orientation, causes a rotation of the vectors, which also could lead to zero vectors in the middle of the interpolation domain due to numerical instabilities.

Another obvious problem is, that the linear interpolation of two unit vectors is not a unit vector anymore. Interpolated eigenvectors would leave the vectorspace of unit vectors. Kindlmann et al. [KWH00] introduced a scheme for interpolating eigenvalues and eigenvectors in three dimensions directly by using their *correspondence*. The correspondence in [KWH00] is defined by the sorted eigenvalues and the pairs of eigenvalues (λ_1, λ_2) and (λ_2, λ_3). This correspondence can then also be applied to eigenvectors. Kindlmann et al. also states, that other, more complex, schemes for computing correspondence are possible to avoid problems, for example with critical points.

In our case, it is sophisticated to use component wise interpolation of tensors. Since the interpolation coefficients are independent from the tensor indices, the interpolation of a three-dimensional, second-order tensor in cartesian space (no difference between covariant and contravariant indices) can now be written as the linear combination of the edge tensors of the surrounding volume element.

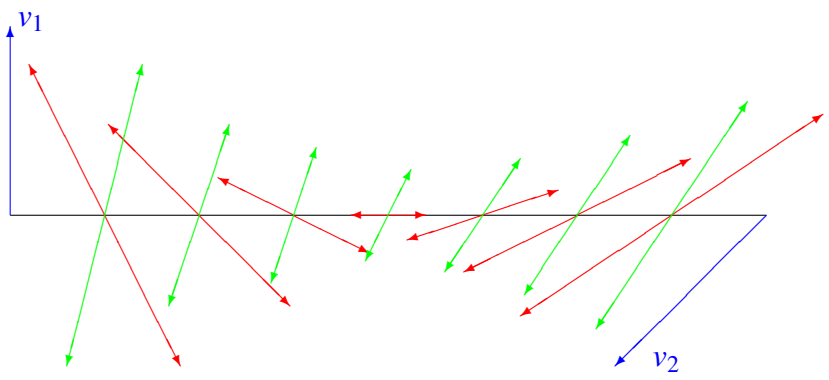


Figure 2.1: *Naive interpolation of two vectors v_1, v_2 (blue) with different orientations. The green vectors are the correctly interpolated vectors if v_2's orientation would be positive just as the orientation of v_1. The red vectors are the wrongly interpolated vectors if v_2's orientation is different from v_1's.*

Second-order tensors are closed under linear combination and in special, linear combination of symmetric tensors remains a symmetric tensor. The addition and scalar multiplication rules from Definition 2.1.2 can be used to proof it. Definition 2.17 shows interpolation of a tensor T on the plane of a triangle with the edge points (p_1, p_2, p_3) and the corresponding tensors.

Definition 2.17 (trilinear interpolation on triangle plane) *Let $\beta_{1...3} \in [0,1]$ be the barycentric coordinates of a point p inside a triangle in three dimensions and T^1, T^2 and T^3 the corresponding three-dimensional second-order tensors at the triangles edges. The interpolated tensor T at point p is then componentwise defined by:*

$$T_{ij} = \frac{\sum_{0<k<3} \beta_k T_{ij}^k}{\sum_{0<k<3} \beta_k}$$

This definition can easily be extended to more complex grid structures, by calculating barycentric coordinates for a point p inside the volume element.

2.1.5 Diffusion Tensors

To complete this chapter, we give an overview to diffusion tensors and their interpretation and characteristics. In [HJ05], a more detailed overview to diffusion tensor imaging is given. It is also used as foundation of this section, but we limit ourselves to some basics of diffusion tensors and diffusion tensor imaging.

In medical visualization, second-order diffusion tensor fields played and play

an important role. Several methods have been introduced to visualize second order diffusion tensor fields and many tensor quantities where introduced in the past. First we define the diffusion tensor and its properties distincting it from the general tensor.

Definition 2.18 (Diffusion tensor) *A diffusion tensor is defined using the three-times-three matrix notation as*

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

As can be seen in Definition 2.18, the diffusion tensors D are symmetric and therefore they have three positive real eigenvalues λ_1, λ_2, and λ_3. Its positive definiteness ensures that there are three eigenvectors (except it is a critical point) v_{λ_1}, v_{λ_2}, and v_{λ_3} associated with the eigenvalues. These eigenvectors are, since the diffusion tensor field is in cartesian space, orthogonal to each other, which makes their pairwise inner product zero:

$$< v_{\lambda_1}, v_{\lambda_2} >=< v_{\lambda_2}, v_{\lambda_3} >=< v_{\lambda_1}, v_{\lambda_3} >= 0 \tag{2.14}$$

$$v_{\lambda_1} \times v_{\lambda_2} = v_{\lambda_3}. \tag{2.15}$$

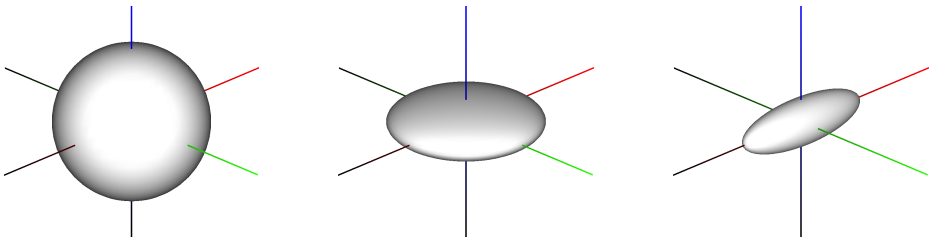
In practice, this is a great advantage and can be used to reduce calculation effort. Another feature is that they are orientationless, which means the eigenvectors just have an absolute value and direction but no orientation. This can cause problems, for example during interpolation which is handled in Section 2.1.4.

With all euphemism about diffusion tensors, there is something to take care about: critical points.

Definition 2.19 (critical point) *Let D be a diffusion tensor with two equal eigenvalues λ_i and λ_j ($i \neq j$). Then D is called critical point.*

In other words, at these points, at least one of the anisotropy measures c_l or c_p becomes zero. The eigenvectors for those equal eigenvalues are therefore not uniquely defined at such a point.

The diffusion tensor can now be interpreted as ellipsoid describing the shape to which water diffuses from a given point in a given time. The eigenvectors of a diffusion tensor D are the spanning radii of this ellipsoid. This is also one of



(a) Isotropy (b) Linear anisotropy (c) Planar anisotropy

Figure 2.2: *Diffusion tensor shapes illustrating ellipsoidal interpretation of diffusion tensors*

the foundations on which the method in this thesis is based on. The eigenvectors describe the main diffusion directions but not their orientation, since they are orientationless. Figure 2.2 illustrates this interpretation.

The ellipsoidal interpretation leads to a characterization of a tensor inside some kind of a space of shapes, defined by three barycentric coordinates, each representing one distinct kind of ellipsoid shape. Westin et al. [WPG+97] introduced such an intuitive domain that spans all possible shapes of tensors using geometric anisotropy metrics

$$\begin{aligned} c_l &= \frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2 + \lambda_3}, \\ c_p &= \frac{2(\lambda_2 - \lambda_3)}{\lambda_1 + \lambda_2 + \lambda_3}, \text{ and} \\ c_s &= \frac{3\lambda_3}{\lambda_1 + \lambda_2 + \lambda_3}. \end{aligned} \tag{2.16}$$

Assuming the eigenvalues are sorted, this defines a triangular domain with the barycentric coordinates c_l, c_p, c_s defining linear, planar, and spherical certainity.

As a direct tensor quantity, the apparent diffusion coefficient uses the diffusion tensor, unlike the shape domain, to describe the diffusion along a given direction.

Definition 2.20 (Apparent diffusion coefficient (ADC)) *Let D be a diffusion tensor and v be a normalized direction vector. Then, the apparent diffusion coefficient (ADC) along v for D is defined as:*

$$a_v = v^T D v$$

Basser et al. [BP96] introduced several rotationally invariant anisotropy metrics like the relative anisotropy or the fractional anisotropy. It measures the degree

of anisotropy in a given voxel, whereas mean diffusivity $MD(D) = \frac{trace(D)}{3}$ is a measure of total diffusion within a voxel.

Definition 2.21 (Fractional Anisotropy) *Let D be a diffusion tensor, MD its mean diffusivity, and $\lambda_1, \lambda_2, \lambda_3$ its eigenvalues. Then, the scalar function defined by*

$$FA(D) = \sqrt{\frac{3}{2}} \sqrt{\frac{\sum_i (\lambda_i - MD(D))^2}{\sum_i \lambda_i}} \in [0,1]$$

is called fractional anisotropy.

A value of 0 is the perfect isotropic diffusion, whereas a value of 1 represents the (hypothetical) case of an infinite cylinder. As *FA* is calculated using only one tensor, it is prone to noise conditions. The main difference in application of mean diffusivity and fractional anisotropy directly results from the following. The mean diffusivity *MD* can be used to distinguish between brain tissue and its surrounding fluid, the cerebrospinal fluid, where high diffusion leads to a high *MD*. The *FA* can be used where the mean diffusivity is not able to distinguish different types of brain tissue; a high *FA*, for example, often indicates white matter tissue.

As we will see later in Section 2.3, most diffusion tensor field visualization methods use combinations of scalar measures, like those above and several vector fields (like eigenvectors) derived from the tensor field to show the structure or special features of tensor fields.

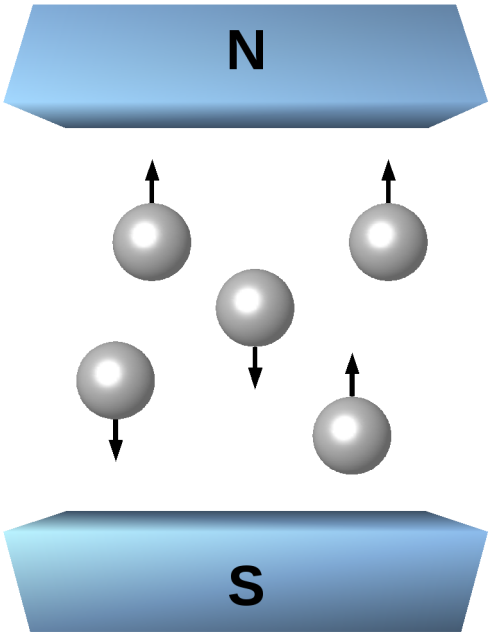


Figure 2.3: *Nuclei in a homogeneous magnetic field. Some of the protons are aligned parallel to field lines and some are aligned anti-parallel.*

2.2 Data Acquisition

In the previous section, diffusion tensors and tensor fields have been introduced. But how are these fields created? In this section we will have a look at the basic ideas behind diffusion tensor data and how it can be processed to extract interesting regions.

2.2.1 Magnetic Resonance Imaging

In this chapter, we overview the method of magnetic resonance imaging. We abstain all the details of MRI, not necessarily needed to get a basic understanding of this method, since it would require a deep knowledge of quantum physics and field theory. In [Nes09] such a simplified overview is given, which, at the same time, is the base of this section together with Preim and Bartz [PB07].

In magnetic resonance imaging (MRI) the properties of hydrogen nuclei in strong magnetic fields are exploited. Protons and neutrons have a natural spin which gives the nucleus a magnetic moment. These small dipole magnets can be aligned either parallel or anti-parallel to a strong homogeneous magnetic field. Protons aligned anti-parallel and parallel cancel each other but a slight excess of protons align parallel to the field which generates a measurable magnetization. Figure 2.3 illustrates this. A complete explanation of why some protons align

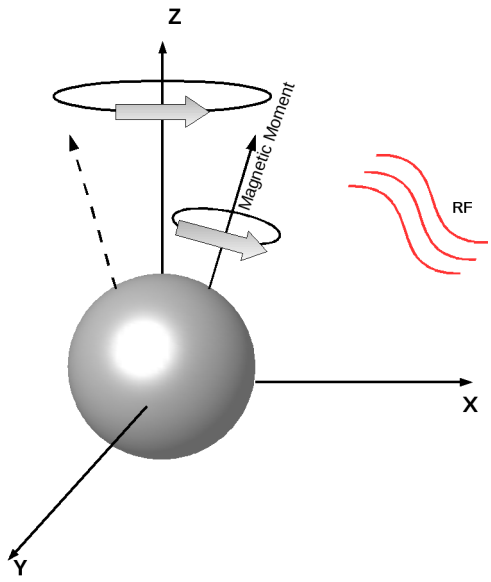


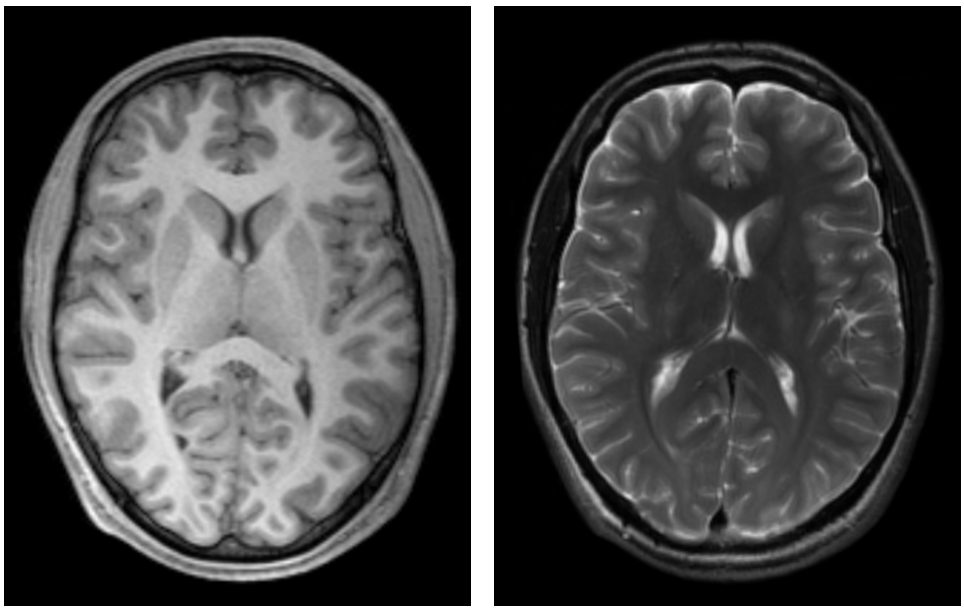
Figure 2.4: *Illustration of a nucleon rotating around its own magnetic axe, while a Larmor frequency field is applied, which causes the nucleon to spiral down to the XY-plane.*

against and some parallel the magnetic field can only be done by using quantum mechanics. Here it suffices to say that the alignment parallel the magnetic field is the lower energetic state. The larger the external field, the larger the difference in energy levels and the larger the excess number of protons aligned parallel to the field. That is also the reason high field scanners have a better signal/noise ratio.

To measure the magnetization, a second, perpendicular magnetic field must be applied. If the field has a special frequency, the Larmor frequency, which is the resonance frequency of the protons, the protons absorb a part of the fields energy and enter a higher quantum energy level and begin to spiral into the XY-Plane (assuming the initial magnet field to be along the Z-Axis) as shown in Figure 2.4. This precession is in sync for all protons; they are in phase.

After releasing the field, basically three things happen. First, the absorbed energy gets retransmitted as RF waves, since rotating magnetic fields produce electromagnetic radiation, which induces electrical energy in nearby coils. This so called *NMR* signal is proportional to the proton density, which is actually the volumetric information.

While radiating RF waves, the protons return to the lower quantum energy level and, therefore, begin to realign with the initial Z-Axis aligned magnetic field, at the expense of the XY component. Since not all the prior absorbed energy is emitted as RF signal, some energy is heating up the surrounding tissue, which is called lattice. This spin-lattice relaxation can be described by an exponential



(a) T1 (b) T2

Figure 2.5: *The gray values in these MRI images, relate to the T1 (left) and T2 (right) relaxation times. It is easy to distinguish between cerebrospinal fluid, gray, and white matter. For example, in the T1 image on the left,* corpus callosum *is nearly white whereas it is dark in the T2 image. Images by courtesy of Mario Hlawitschka.*

curve and the time t, where 63.2% of the original magnetization is restored, is called *T1*.

The third thing to happen after removing the RF field is called spin-spin relaxation. Since the spinning protons create magnetic fields, neighboring protons begin to influence each others, while de-phasing. These random, local interactions cause a cumulative de-phasing across the protons and therefore resulting in a loss of the signal. This loss of signal can be described by an exponential curve, just as T1. Analogous to T1, T2 is defined as the time t where a decay of 63.2% in signal amplitude is reached. Both, spin-lattice relaxation and spin-spin relaxation are called *free induction decay (FID)*.

The times T1 and T2 can now directly be used to distinguish tissue, since water has a long T1 and T2 relaxation time, tissue with a high water portions has higher relaxation times than other tissue types.

The method described above is not quite usable without spatial information. A MRI scanner is able to address volume elements using two additional gradient magnetic fields. During the RF field pulse, a gradient magnetic field is applied to select a slice in Z direction of the scanned volume. This guarantees, that only one slice of protons suffices the Larmor frequency of the RF field. A

second gradient field applied during de-phasing encodes XY information in the frequency and phase of the signal, allowing it to be decoded using Fourier transformation [LOPR97, CJS93]. In Figure 2.5 a T1 and a T2 image of a human brain dataset is shown for comparison. The method can be modified using several protocols, like EPI often used for DTI. These protocols describe various RF pulse sequences and measurement methods to achieve different images, emphasising several structures. In [PB07] a more detailed overview of different protocols and T1/T2 weighting modifications is given.

2.2.2 Diffusion Tensor Imaging

After introducing MRI, we now shortly introduce diffusion tensor MRI. Diffusion tensor imaging, abbreviated DTI, is a often used modality of MRI, besides fMRI. In contrast to functional MRI (fMRI), which measures changes in blood flow and oxygen metabolism during neural activity, DTI detects water diffusion and represents it in a diffusion tensor, as described in Section 2.1.5. Therefore it is an generalization of diffusion-weighted MRI, which detects the amount of diffusion in a volume.

To measure nuclei motion, instead of density as in standard MRI, another gradient magnetic field gets applied. Initially introduced by Basser et al. [BML94], the idea behind this additional gradient is similar to the fields used in Section 2.2.1 to select volumes in the scanning area. The MRI scanner applies this field in multiple directions one after one to only address nuclei moving along a certain direction. The number of moving nuclei along these directions and the overall number of nuclei specifies a diffusion profile, which is used to calculate a symmetric, three-dimensional second-order tensor, the diffusion tensor D.

The data generated by DT-MRI scanners is challenging in processing and visualization. The spatial resolution is lower than in standard MRI and the artifacts, already known from MRI, can get even worse due to the longer scanning time ramifications (like movement of or in the scanned object). Diffusion tensor MRI is also not able to measure very small fibers, due to its limited resolution, as well as crossing fiber paths. To detect crossing or bifurcating fibers, another method like HARDI (High Angular Resolution Diffusion Imaging), is needed, which is able to measure the diffusion in an volume element from more angles than DT-MRI can do.

In spite of its problems, DTI finds many applications. In neurology, diffusion tensor imaging enables scientists and radiologists to find connection paths (fiber tracts) in the brain's white matter between functional areas in the gray matter, or diagnose their absence and therefore neurodegenerative deceases like Alzheimer's. So, DTI is useful to characterize water diffusion anisotropy and diffusion directions in certain tissues, which allows reconstruction of neurological structures and their connection paths inside the brain [ZB02]. These connection

paths, or fiber tracts, coincide with the direction of highest diffusion. This tractography methods are not limited to structures in the brain. It is also possible to extend these methods to find muscle fibers in the heart using diffusion tensor MRI [ZB03].

In the next sections we will introduce some common methods for processing and visualizing diffusion tensor fields. For further details on diffusion tensor imaging, Bartz et al. [PB07] gave a comprehensive overview.

2.3 MRI and DT-MRI Visualization Methods

In the previous sections, the basic mathematical foundations and data sources got described, we now take a short look at some common and nearly omnipresent tensor field visualization techniques.

2.3.1 Color Mapping

Color mapping is a famous visualization method, not limited to medical visualization. It is a general-purpose technique, able to visualize all kinds of data for which a transfer function is available, mapping the input data to a vector in an arbitrary color space. We already have seen color mapping in action: in Figure 2.5, the T1 and T2 relaxation times during magnetic resonance imaging got directly mapped to gray values representing them. These colormaps are mapping a scalar value to a two-dimensional, arbitrary slice in the dataset. Of course color mapping is not limited to this two dimensional case. Color mapping can be applied to slices in datasets, glyph visualizations or complex three-dimensional structures. In general color mapping is a method to map a n^{th}-order tensor at every point p in a space $S \in \mathbb{R}^m, m \in \{1,2,3,\ldots\}$ spanning the data field to an arbitrary color space using a transfer function appropriate for the n-dimensional value. An important requirement for colormaps is to have similar colors for similar, connected areas, since areas with similar colors are interpreted as being connected by the human visual system.

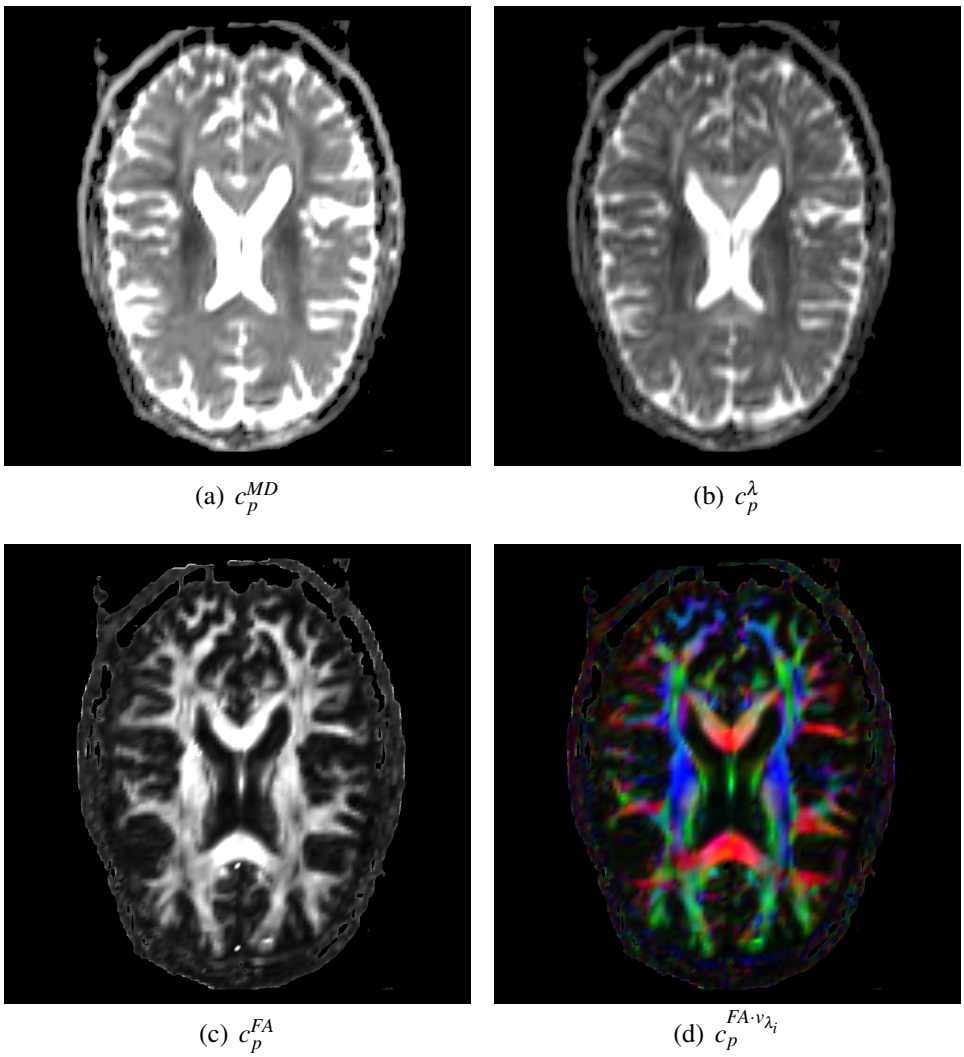
Definition 2.22 (Color mapping) *Let $S \subseteq \mathbb{R}^m$ be an m- dimensional Space, most commonly $m \in \{2,3\}$ and let C be an arbitrary color space. Then, a function*

$$c_p : \mathscr{W}^n \to C$$

is called colormap or transfer function.

Definition 2.22 can be interpreted as a function that maps a tensor to a color. In the case of diffusion tensor images, there are many color mappings available. We will mention only the most commonly used. In Section 2.1.5 some metrics have been introduced for second-order tensors. This leads to the first application, the color coding of anisotropy and diffusion metrics like

$$\begin{aligned} c_p^{FA}(T) &= FA(T) \cdot (1,1,1), \\ c_p^{MD}(T) &= MD(T) \cdot (1,1,1), \text{ and} \\ c_p^{\lambda}(T) &= (\lambda_1, \lambda_2, \lambda_3) \end{aligned} \tag{2.17}$$



(a) c_p^{MD} (b) c_p^{λ}

(c) c_p^{FA} (d) $c_p^{FA \cdot v_{\lambda_i}}$

Figure 2.6: *Example colormaps showing a slice of a human brain DTI dataset. In these images, mean diffusivity is used to clip noise outside the cranium. As described above, each colormap emphasises several parts in the dataset.*

to mention only some of the possibilities. Note that here, only the RGB color space is used. Another widely used application of color mapping is to use directional information extracted from tensors to color each point. Like the color mappings above, the directional information, the eigenvectors need to be extracted from the tensor, since the symmetric tensor's six values can not be mapped to color directly. A first naive approach is to directly map the eigenvectors of T to a color value as in

$$c_p^{v_{\lambda_i}}(T) = \frac{|v_{\lambda_i}|}{\|v_{\lambda_i}\|}. \tag{2.18}$$

This can now be scaled using, for example, fractional anisotropy, to filter out directional information in isotropic regions, where it is useless:

$$c_p^{FA \cdot v_{\lambda_i}}(T) = \frac{|v_{\lambda_i}|}{\|v_{\lambda_i}\|} \cdot FA(T). \tag{2.19}$$

Not explicitly mentioned in the equations above, it is also useful to threshold mean diffusivity or the fractional anisotropy, to blend out several areas not interesting, similar to the effect reached with scaling by *FA* as in Equation 2.19.

In Figure 2.6, several colormaps are compared. Mean diffusivity is used to clip areas not inside the cranium. Figure 2.6(d) shows one of the most common colormaps, which is also used in our method since it helps to find areas of similar diffusion direction.

We leave this topic now since many more mappings have been introduced, which are not needed in our method. For example, the simple linear mapping in RGB space can be replaced by other variants, like cylindric colormaps or just simple grayscale mappings.

2.3.2 Tensor Glyphs

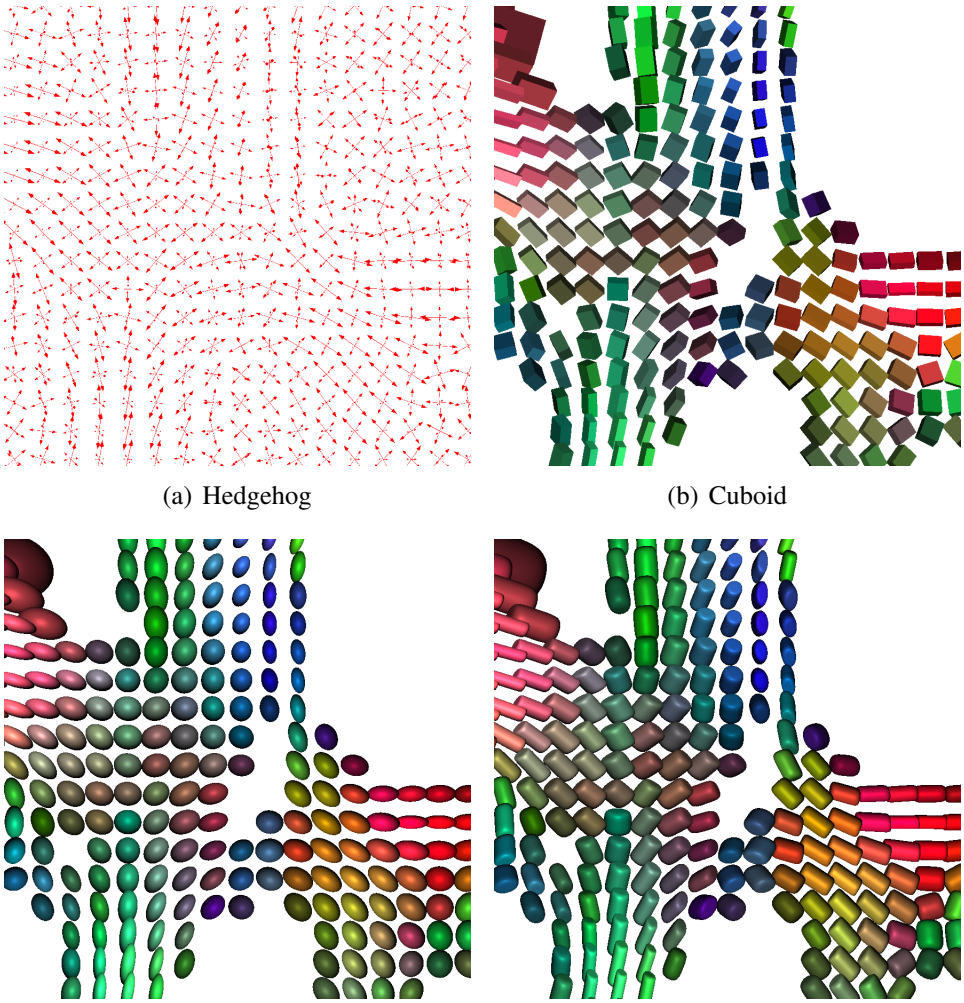
Colormaps have a limitation, tensor glyphs do not have: It is limited to display three distinct metrics at max or one direction for each point. Glyphs are able to fill this gap. Glyphs are objects at every data point in space representing scalar, vectorial, or tensorial information. In the case of diffusion tensor imaging the information to display are the eigenvalues and eigenvectors, as well as the anisotropy or diffusivity metrics in a glyphs color (see Section 2.3.1). In DTI, a glyph's directional transformation is then defined by the eigenvectors, representing the major, medium, and minor diffusion directions and the glyph is scaled by the eigenvalues λ_1, λ_2, and λ_3.

Definition 2.23 (Tensor glyph) *Let G be a glyph geometry and R be a rotation matrix. Then, G can be transformed to a tensor glyph geometry G_T by using*

$$G_T = R^{-1} \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} RG,$$

which uses the tensors eigenvectors to span an orthogonal coordinate system.

Although glyphs may be (geometrically) defined in any possible way, Kindlmann [Kin04b] mentioned some criteria glyphs should match to:



(a) Hedgehog (b) Cuboid

(c) Ellipsoid (d) Superquadrics

Figure 2.7: *Different kinds of glyphs in a small part of a slice in a human brain DTI dataset. Colorization is based on the color mapping function in Equation 2.19. Hedgehog glyphs are nearly unusable for humans to recognize tensor field structures. The color coded glyphs offer the possibility to find areas of similar diffusion directions, although they may be hard to distinguish in their shape.*

(g1) **Continuity.** Minimal changes of neighboring tensors should not lead to discontinuities in neighboring tensor glyph geometry.

(g2) **Uniqueness.** The mapping from a tensor shape to a glyph geometry has to be unique.

(g3) **Unambiguity.** Tensor geometry needs to be unambiguous and recognizable after projection to image space, regardless of viewing direc-

tion.

Many glyph geometries have been explored, but many of them collide with at least one of the above criteria. So, for example using boxes, which represent the diffusion directions, matches (g1) and (g2), but boxes are not distinguishable from every view direction. Using ellipsoids suffers the same problem as cuboids but match the tensor interpretation very well. A solution that meets all the criteria and therefore maps the barycentric tensor shape space spanned by c_l, c_p, and c_s from Equation 2.16 very well, besides being distinguishable from every view direction, are superquadrics. Superquadrics where introduced by A. H. Barr in 1981 [Bar81] and suggested for visualizing DTI data by Kindlmann [Kin04a] in 2004. Since superquadrics are implicit surfaces, triangulation of thousands of superquadrics is a challenging task, even for modern CPU and GPU. Hlawitschka et al. [HES08] introduced a fast, GPU based raytracing method, which allows rendering of thousands of superquadric glyphs in realtime, making superquadric glyphs very usable in interactive exploration of large DTI datasets. In Figure 2.7, superquadrics are compared to ellipsoids, cuboid glyphs, and hedgehog glyphs, which are arrays representing the scaled eigendirections.

2.3.3 Isosurfaces

An isosurface is a surface connecting points of similar properties in space. It is the higher-dimensional correspondent to isolines in two-dimensional space, where points with an equal, fixed isovalue get connected.

Mathematically, it can be described as implicit function over an n-dimensional scalar field.

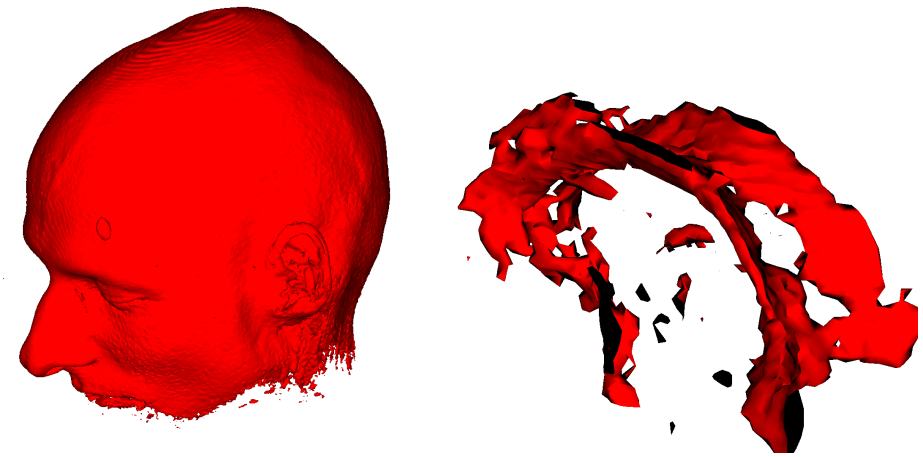
Definition 2.24 (Isosurface) *Let $\varphi : S \to \mathbb{R}$ be a scalar field in an n-dimensional cartesian space $S \subseteq \mathbb{R}^n$. Then, the set*

$$S_c = \{p \in \mathbb{R}^n | \varphi(v) = c\}$$

is called isosurface with an isovalue *c. As implicit function, an isosurface is defined as the solutions of*

$$\varphi(p) - c = 0, \text{ with } p \in S$$

The implicit character of isosurfaces and the discrete nature of the underlaying scalar field requires complex methods to triangulate them. The famous marching cubes technique to triangulate isosurfaces was introduced 1987 by Lorensen et al. [LC87]. The basic idea behind marching cubes is, as the name already implies,



(a) T2 dataset with isovalue 0.72 (b) FA of DTI dataset at isovalue 0.7

Figure 2.8: *Two isosurfaces triangulated using marching cubes, one using a T2 dataset (left) and another using the fractional anisotropy (right) from a second-order tensor field as the isosurface's scalar property. The dataset in image (b) will be used for demonstration of our method.*

to build the isosurface cube by cube, which makes it a classic divide and conquer approach. In a DTI dataset, the marching cubes algorithm interpolates values inside a cube using trilinear interpolation and on the cube edges using linear interpolation. This causes the isosurface to cut each edge at most one times. The exact values at each corner are not needed. For each cube corner it is just stored whether the value is larger or lower. This information and the corner number is used to make a table lookup, storing each possible upper/lower configuration in a cube. In addition to this, there is also an edge table, containing a triangulation for each possible configuration.

The standard marching cubes algorithm suffers an ambiguity during corner based table lookup. In a cube, where each cube corner is in inverse state to its neighboring corners (positive/negative alternation), where it is not decidable whether to connect the positive or negative corners. Several approaches where explored to improve the algorithm and solve this problem [Dü88, NH91]. Besides approaches to solve ambiguity problems of marching cubes, there where also several speed improvements explored. So it is possible to divide the data volume using octrees [WVG92] or span spaces [LSJ96], assigning each cube a minimum and maximum to decide whether to take a cell into account before touching the cell.

For a more complete overview to isosurfaces, it is advised to read the corresponding chapter in [HJ05]. We close this section with an image of an isosurface of a human brain in Figure 2.8.

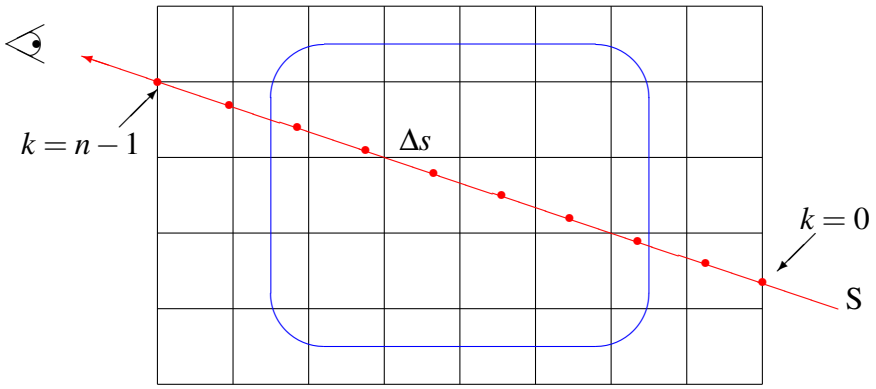


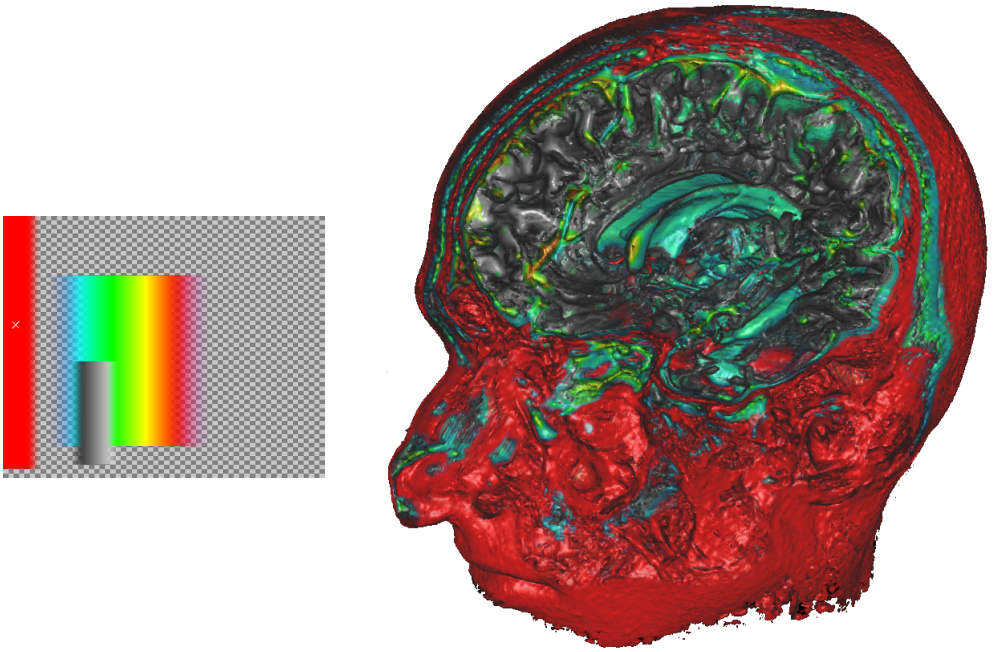
Figure 2.9: *Schematic illustration of the general idea behind direct volume rendering in discrete case. A ray S is sent through the volume and crosses the media, bordered by the blue oval. The sampling density described by Δs mainly depends on the used interpolation and lighting model. The sampling theorem thereby sets the minimal sampling frequency.*

2.3.4 Direct Volume Rendering

In the previous section, isosurfaces where introduced, which belongs to a class of techniques called indirect volume rendering. In contrast to direct volume rendering, indirect volume rendering creates intermediate representation of the volume data and renders it.

In the 1980's several method where introduced to avoid triangulation of clouds, dusty surfaces, or other density based volumes. These methods, namely [Bli82, KVH84], adopted the idea of ray-tracing to visualize volumetric data and therefore build the foundation of direct volume rendering. It is assumed, that the volume to be rendered is filled with a media, whose optical properties are defined by the scalar values inside the volume. To reach maximum flexibility, the mapping from a value at a given point to a color can be done using transfer functions, similar to those described in Section 2.3.1. This allows emphasizing different parts of the volume through varying the emission and absorption properties of a certain structure. For diffusion tensor images, anisotropy measures have proven to be appropriate [KWH00] as scalar property to be used in transfer function definition and, therefore, for color and opacity.

As in ray-tracing, a ray gets send through the volume for each pixel of the viewport, but the final color is not determined by the first hit of the ray with an object or maybe after several reflections. The final color on the viewport is defined by composition of the color from many sample points along the ray, whereas each sample point is classified using the transfer function. In addition to the transfer function, an opacity transfer function may be specified, to manipulate attenuation/transparency at each sample point. The contribution to the final illumination at each sample finally depends on the illumination model, normally Phong, and



(a) Transfer function (b) Volume rendering

Figure 2.10: *A volume rendering (right) of a T2 MRI dataset. The transfer function (left) colors the brain matter in gray while also emphasizing the* corpus callosum.

the absorption/emission properties at this point.

The direct volume rendering method can be described mathematically and illustrative as in Figure 2.9.

Definition 2.25 (discrete volume rendering equation) *Let Δs be the sampling step size and let the ray be defined as $S = k \cdot \Delta s$ with $k = 0 \ldots n-1$. Then, the intensity $I(S)$ can be calculated by combining every contribution Q, which is defined by the transfer function and the transparency t at each sampling point k:*

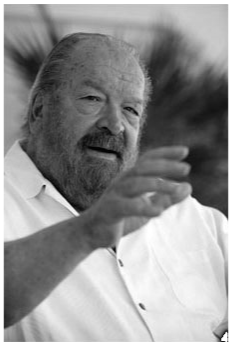
$$I(S) = I_0 \prod_{k=0}^{n-1} t_k + \sum_{k=0}^{n-1} Q(k \cdot \Delta s) \cdot \Delta s \prod_{j=k+1}^{n-1} t_j.$$

The volume rendering equation in Definition 2.25 mathematically specifies the direct volume rendering procedure, but lets much freedom to how it is evaluated. Many direct volume rendering algorithms got introduced. In general, they can be

classified in image-order, object-order, and hybrid methods. Object order methods like splatting [Wes91] map the volume elements to the image plane, whereas image-order methods, as raycasting, map pixel-wise. Splatting does not use rays but uses a convolution mask (splat) on each volume element to reconstruct the volume on the image plane. Since the pure raycasting procedure suffers several performance problems, the hybrid method shear warp [LL94] was introduced in 1994, where all slices in the volume get sheared (object-space) and the final image gets warped to have the same result as an arbitrary orthographic projection but with rays parallel to one axis.

Of course, there are many more approaches for direct volume rendering, which would exceed this thesis, so we refer to [PB07] for a more comprehensive overview.

We close this section with Figure 2.10, showing a direct volume rendered T2 dataset and its corresponding transfer function.



*Carlo Pedersoli (Bud Spencer), *1929*
Real multi-talent: Actor, filmmaker, swimmer,
Doctor of Law, and owner of multiple patends.

3 Related Work, the Motivation, and the Goals

In this chapter, we want to frame the scope of this thesis, thus distinguishing the newly elaborated method from other approaches. At first, we give an overview over the current work directly related to our approach.

3.1 Related Work

Hotz et al. [HFH+04] introduced Physically Based Methods (PBM) for tensor field visualization in 2004 as a means to visualize stress and strain tensors arising in geomechanics. A positive definite metric that has the same topological structure as the tensor field, is defined and visualized using a texture-based approach resembling Line Integral Convolution (LIC). Besides other information, eigenvalues of the metric can then be encoded by free parameters of the texture definition, such as the remaining color space. Whereas the method's implementation for parameterizable surfaces topologically equivalent to discs or spheres is straightforward, implementations for arbitrary surfaces remains computationally challenging.

In 2009, Hotz et al. [HFHJ09] enhanced their approach to isosurfaces in three-dimensional tensor fields. A three-dimensional noise texture is computed in the data set and a convolution is performed along integral lines tangential to the eigenvector field.

LIC has been used in vector fields to imitate *Schlieren* patterns on surfaces, that are generated in experiments, where a thin film of oil is applied to surfaces, which show patterns along the air flow. In vector field visualization, image space

LIC is a method to compute Schlieren-like textures [vW02, vW03, LJH03, GL05] in image space, intended for large and non-parameterized geometries. Besides the non-trivial application of image space LIC to PBM, it has certain drawbacks. Mainly, due to the noise pattern being defined in image space, it does not translate the same way the surface moves and, therefore, when interacting with the data, the three-dimensional impression is lost. A simple method proposed to circumvent this problem is animating the texture pattern by applying randomized trigonometric functions to the input noise. Weiskopf and Ertl [WE04] solved this problem for vector field visualization by generating a three-dimensional texture, that is scaled appropriately in physical space.

3.2 Motivation

The methods mentioned above cover two types of visualization: image space based LIC for vector fields and physically based tensor field visualization for second-order tensor fields in geomechanics. The method from Hotz et al. [HFH+04, HFHJ09] is able to visualize physical properties of tensor fields. But the method suffers generality. It is designed for geomechanics and is not capable to offer realtime user interaction. In addition to that, it is limited to simple geometry. A great advantage of this approach is the capability to show eigendirections together with eigenvalues; i.e. by varying point density in the initial noise texture.

The image space based approach to open large and complex geometry to classical vector field LIC in realtime, on the other hand, offers enough flexibility to adopt this method to many problems, that can be described using vector fields. But it is not able to show physical properties of tensor fields, simply by using an eigendirection of the field.

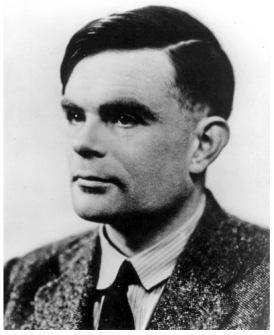
Our novel approach is motivated by combining the advantages of both mentioned methods, which directly leads to the goals to achieve.

3.3 Goal

For short, the goal is to combine both types of approaches. Simply putting both methods into one does not solve the problem. In the next chapters, we introduce a method, including implementation, which moves the problem into image space and uses the basic ideas from the above methods to create an interactive and general technique, not limited to some type of scientific data or limited by special claims to the type of geometry. Since our method is capable of realtime user interaction, we also focus on regaining three-dimensionality of the data during operations like rotation and scaling. For summing-up, the main goals to achieve where:

- Visualization of physical properties of tensor fields
- Generality/flexibility in its possible applications
- Extensibility; i.e. for higher order tensor data
- No or minimal claim to the input data
- Realtime user interaction
- Preservation of spatial perception during interaction
- Feasibility on todays (graphics-) hardware

The method employed in this thesis will meet this requirements mostly. *Of course*, our technique suffers several limits and problems, mainly by its image space nature and the hardware restrictions. In Chapter 7, we will have a critical look onto it, showing its advantages but also its limitations and problems.



*Alan Turing, *1912 - †1954.*
Among other things, he created the mathematical foundations of todays computer science.

4 Method

In this chapter, we employ a multi-pass rendering technique that consists of four major rendering passes as outlined in Figure 4.1. After generating the basic input textures once, the first pass projects all required data into image space. Pass two performs a silhouette detection that is used to guarantee integrity of the advection step computed by multiple iterations of pass three. Eventually, pass four composes the intermediate textures in a final rendering. In every section, an accompanying image is shown to illustrate the results of every rendering pass.

4.1 Projection into Image Space

First, we project the data into image space by rendering the surface using the default OpenGL rendering pipeline. Notably, the surface does not need to be represented by a surface mesh, but any other representation that provides proper depth and surface normal information works just as well (e.g., ray-casting methods for implicit surfaces, cf. Knoll et al. [KHH+07]). In the same rendering step, the tensor field is transformed from world space to object space, i.e., each tensor T is projected onto the surface.

Definition 4.1 (Object Space Projection) *Let P be an symmetric projection matrix, describing projection to the object space and T a second-order tensor. Then T gets projected to object space by:*

$$T' = P \cdot T \cdot P^T.$$

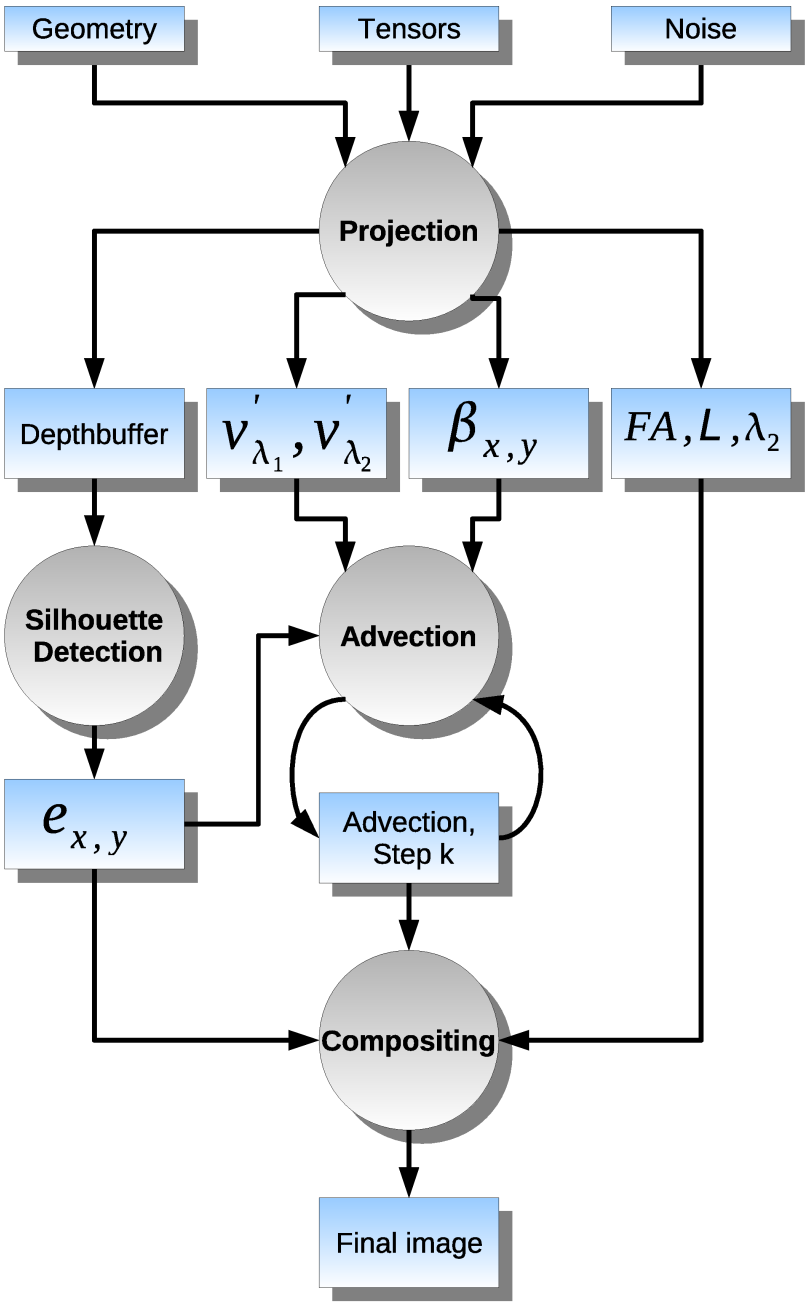


Figure 4.1: *Flowchart indicating the four major steps of the algorithm: projection, which transform the data set in an image-space representation and produce the initial noise texture on the geometry; silhouette detection, required for the advection step and the final rendering; advection, which produces the two eigenvector textures; and final compositing that composes intermediate textures for final visualization.*

To project T to the surface of the three-dimensional object, P can be defined as:

$$P = \begin{pmatrix} 1-n_x^2 & -n_y n_x & -n_z n_x \\ -n_x n_y & 1-n_y^2 & -n_z n_y \\ -n_x n_z & -n_y n_z & 1-n_z^2 \end{pmatrix}. \tag{4.1}$$

The camera viewing system configuration and the available screen resolution imply a super- or sub-sampling of the data. We obtain an interpolated surface tensor in every pixel which is decomposed into the eigenvector/eigenvalue representation using a method derived from the one presented by Hasan et al. [HBPA01]. These eigenvectors, which are still defined in object space, are projected into image space using the same projection matrices M_M and M_P used for projecting the geometry to image space, usually the standard *modelview* and *projection* matrices OpenGL offers:

Definition 4.2 (Image Space Projection) *Let v_{λ_i} be an eigenvector in object space and the matrices M_P and M_M be the OpenGL projection and modelview matrices. Then, the eigenvector v'_{λ_i} is transformed to image space by:*

$$v'_{\lambda_i} = M_P \times M_M \times v_{\lambda_i}, \text{ with } (i \in 1,2).$$

Even in the special case of symmetric second-order tensors in $\mathbb{R}^3$, which have three real-valued eigenvalues and three orthogonal eigenvectors in the non-degenerate case, in general, the projected eigenvalues are not orthogonal in two-dimensional space.

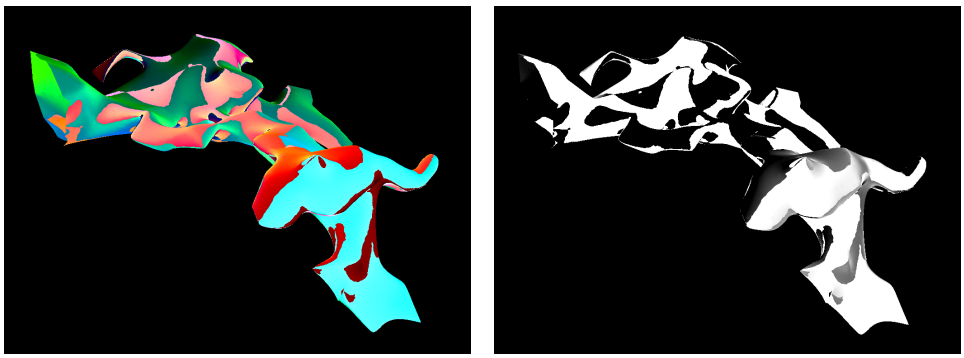
Definition 4.3 (Maximum Norm (L_∞-norm)) *Let $v \in \mathbb{R}^2$ be a vector in $\mathbb{R}^2$. The maximum norm of v is then defined by:*

$$\|v\|_\infty = max\{|v_x|, |v_y|\}.$$

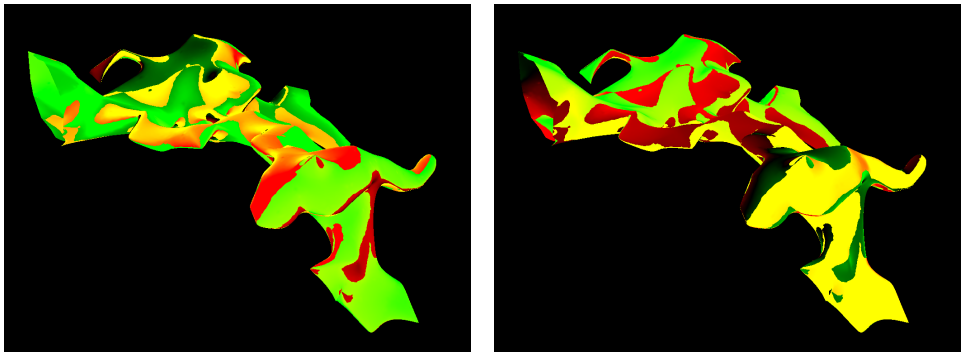
To simplify further data handling, we scale the eigenvectors using the maximum norm:

$$v''_{\lambda_i} = \frac{v'_{\lambda_i}}{\|v'_{\lambda_i}\|_\infty} \text{ with } i \in \{1,2\} \text{ and } \|v'_{\lambda_i}\|_\infty \neq 0. \tag{4.2}$$

The special case $\|v'_{\lambda_i}\|_\infty = 0$ only appears when the surface normal is perpendicular to the view direction and, therefore, can be ignored. The maximum norm (L_∞-norm) ensures that one component is 1 or -1 and, therefore, one avoids numerical instabilities arising when limited storage precision is available, and can



(a) RGB Channels (b) Alpha Channel



(c) v''_{λ_1} (d) v''_{λ_2}

Figure 4.2: *Both eigenvectors v''_{λ_1} and v''_{λ_2} scaled to $[0,1]$ and stored in red, green, blue and alpha channel of a texture (a and b). In addition, we separated both eigenvectors into two images, each showing the x and y component in the red and green color channel (c and d) for illustration.*

use memory-efficient eight-bit textures. In Figure 4.2, the transformed eigenvectors v''_{λ_1} and v''_{λ_2} finally got scaled to the range $[0,1]$ and are then stored in the red, green, blue and alpha channel of a texture, waiting to be transferred to advection step, where these vectors get unscaled to v'_{λ_1} and v'_{λ_2} again.

4.2 Initial Noise Texture Generation

In contrast to standard LIC approaches, to achieve a proper visual representation of the data, high-frequency noise textures, such as white noise, are not suitable. Therefore, we compute the initial noise texture using the reaction diffusion scheme first introduced by Turing [Tur52] to simulate the mixture of two reacting chemicals, which leads to larger but smooth "spots" that are randomly and almost uniquely distributed (cf. Figure 4.3 and Figure 5.4(b)). For the discrete case, the governing equations are:

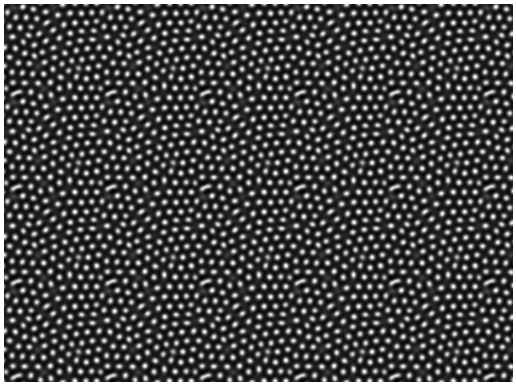


Figure 4.3: *Hundred per hundred pixel reaction diffusion texture created using Alan Turing's method. It got tiled to create a texture of required size, since reaction diffusion is computational expensive. Minor artifacts resulting from tiling can be seen, but ignored in practice due to the advection step.*

Definition 4.4 (Diffusion Reaction Scheme) *Let $a_{i,j}$ and $b_{i,j}$ be two discrete grids indexed by i and j. Furthermore let D_a and D_b be the diffusion constants for each grid. Then Turing's reaction diffusion scheme is defined by:*

$$\Delta a_{i,j} = F(i,j) + D_a \cdot (a_{i+1,j} + a_{i-1,j} + a_{i,j+1} + a_{i,j-1} - 4 \cdot a_{i,j}),$$
$$\Delta b_{i,j} = G(i,j) + D_b \cdot (b_{i+1,j} + b_{i-1,j} + b_{i,j+1} + b_{i,j-1} - 4 \cdot b_{i,j}),$$

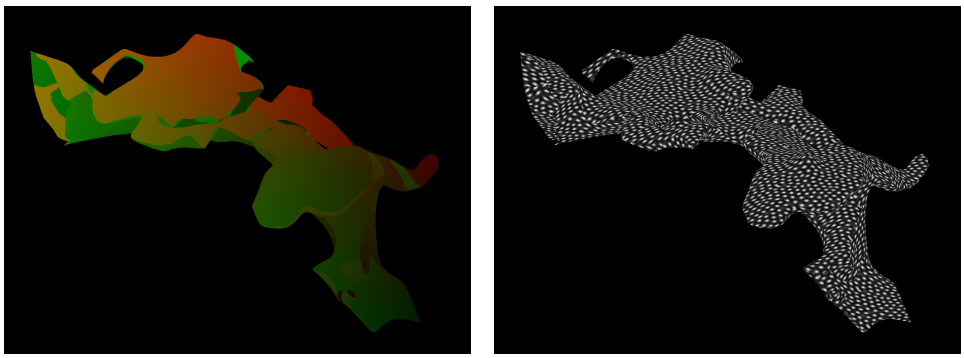
with the functions F and G:

$$F(i,j) = s(16 - a_{i,j} \cdot b_{i,j}) \text{ and } G(i,j) = s(a_{i,j} \cdot b_{i,j} - b_{i,j} - \beta_{i,j}).$$

Here, we assume continuous boundary conditions for the grid coordinates i and j to obtain a seamless texture in both directions. The scalar s allows control over the size of the spots where a smaller value of s leads to larger spots. The constants D_a and D_b are the diffusion constants of each chemical. We use $D_a = 0.125$ and $D_b = 0.031$ to create the input textures, as seen in Figure 4.3.

4.3 Noise Texture Transformation

Mapping the initial texture to the geometry is a difficult and application-dependent task. Even though there exist methods to parameterize a surface, they employ restrictions to the surface (such as being isomorphic to discs or spheres), require additional storage for texture atlases (cf. [PCK04, IOK00]) and, in general, require additional and often time-consuming pre-processing.



(a) Texture coordinate t (b) Geometry mapped noise

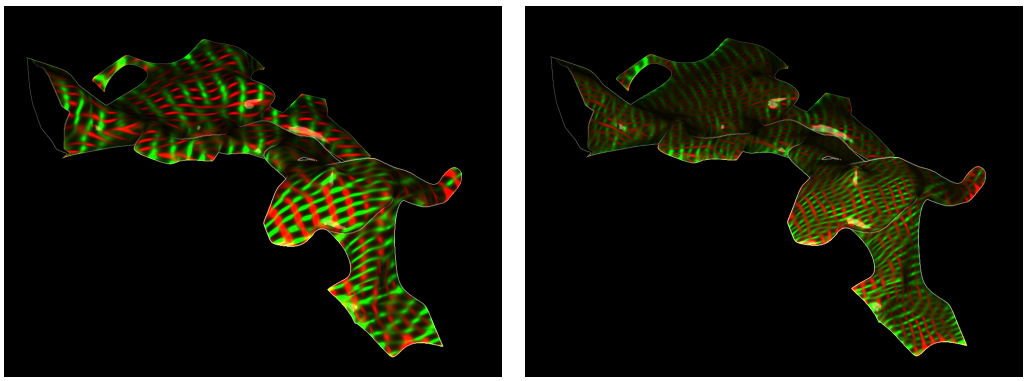
Figure 4.4: *Illustration of the mapping of texture coordinates t to the surface in red and green channels in (a). Discontinuities are the change from one to zero in texture coordinate space and, due to the periodic texture, do not result in discontinuities after texture mapping. In (b), the input noise texture from Figure 4.3 got mapped to the geometry using the texture coordinate t.*

Another solution, proposed by Turk et al. [Tur91], calculates the reaction diffusion texture directly on the surface. The great disadvantage of this method is the computational complexity. Even though these approaches provide almost distortion-free texture representations, isosurfaces, for example, may consist of a large amount of unstructured primitives, which increases the pre-processing times tremendously.

Whereas previous approaches for image space LIC either use parameterized surfaces to apply the initial noise pattern to the surface or use locally or globally defined three-dimensional textures [WE04], we define an implicit parameterization of the surface that provides an appropriate mapping of the noise texture to the surface.

We start by implicitly splitting the world space in voxels of equal size, filling the geometry's bounding box, i.e., we define a regular grid. Each voxel i is described by its base coordinate b_i and a constant edge length l. The seamless reaction diffusion texture is then mapped to the surface of each of these voxels (cf. Figure 4.4(a) and Figure 4.4(b)).

To assign a texture coordinate to each vertex, the object space coordinate is transformed to the voxel space that is described by a minimum and maximum coordinate whose connecting line is the bounding box' diagonal. Points v_g on the geometry are transformed to v_{voxel} using a voxelization.



(a) $l = 0.1$ (b) $l = 0.2$

Figure 4.5: *Comparison of two different values of l to demonstrate the possibility for dynamic refinement of the input noise to achieve different levels of detail. The exact visual result furthermore hardly depends on the size and scaling of the input noise texture.*

Definition 4.5 (Voxelization) *Let v_g be a point on the geometry in object space and l the voxel size. The vector $(b_{min_x}, b_{min_y}, b_{min_z})$ furthermore marks the voxel space root. The vertex transformed to the voxelized space is then defined by:*

$$v_{voxel} = v_g \cdot \begin{pmatrix} l & 0 & 0 & -b_{min_x} \\ 0 & l & 0 & -b_{min_y} \\ 0 & 0 & l & -b_{min_z} \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

The vertex v_{voxel} is subsequently transformed from voxel space to local coordinates on the voxel's faces by:

$$v_{hit} = v_{voxel} - \lfloor v_{voxel} \rfloor. \tag{4.3}$$

The texture coordinates are those two components of v_{hit}, where the surface normal's components are not the maximum. The following Definition 4.6 formalizes this process.

Definition 4.6 (Texture Coordinate Mapping) *Let v_{hit} be the coordinates of v_{voxel} on the voxel's faces and n the surface normal at v_g. The texture coordinates are then defined by:*

$$t = (v_{hit_i}, v_{hit_j}), \text{ with } i \neq j \neq k \wedge (n_k = max\{n_i, n_j, n_k\}).$$

Regardless of its simplicity, this method allows a continuous parameterization of the surface space that only introduces irrelevant distortions for mapping the noise texture. Obviously, the mapping is continuous but not C^1-continuous, which is not required for mapping the noise texture as discontinuities in the first derivatives vanish in the advection step.

Another positive aspect of this mapping is the possibility of a change of scale that is not available in the approaches of, e.g., Turk et al. [Tur91]. By changing the size of voxels during the calculation, different frequencies of patterns can easily be produced and projected to the geometry. This allows a change in resolution of the texture as required for automatic texture refinement when zooming. A comparison of two different levels of detail can be seen in Figure 4.5.

4.4 Silhouette Detection

To avoid advection over geometric boundaries, a silhouette of the object is required to stop advection in these areas [LJH03]. Otherwise, tensor advection would lead to a constant flow of "particles" across surface boundaries which renders the surface's geometry and topology unrecognizable.

Definition 4.7 (Laplacian Filter Kernel) *Let*

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

be the Laplace operator. Then, the Laplacian filter approximates the Laplace operator in discrete case:

$$D^2_{xy} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

A standard three-by-three Laplacian filter applied to the depth values followed by thresholding has proven to be suitable for our purposes and creates the silhouette image:

$$e : (x,y) \rightarrow s, \text{ with } x,y,s \in [0,1]. \tag{4.4}$$

The silhouette image e is shown in Figure 4.6 and as red color channel in the intermediate texture shown in Figure 5.4.

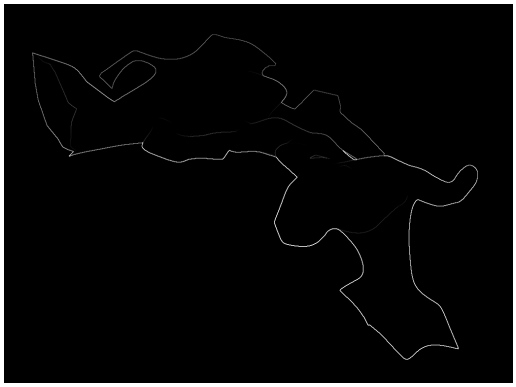


Figure 4.6: *Silhouette created using Laplacian filter. This texture is used to avoid advection across geometry boundaries.*

4.5 Advection

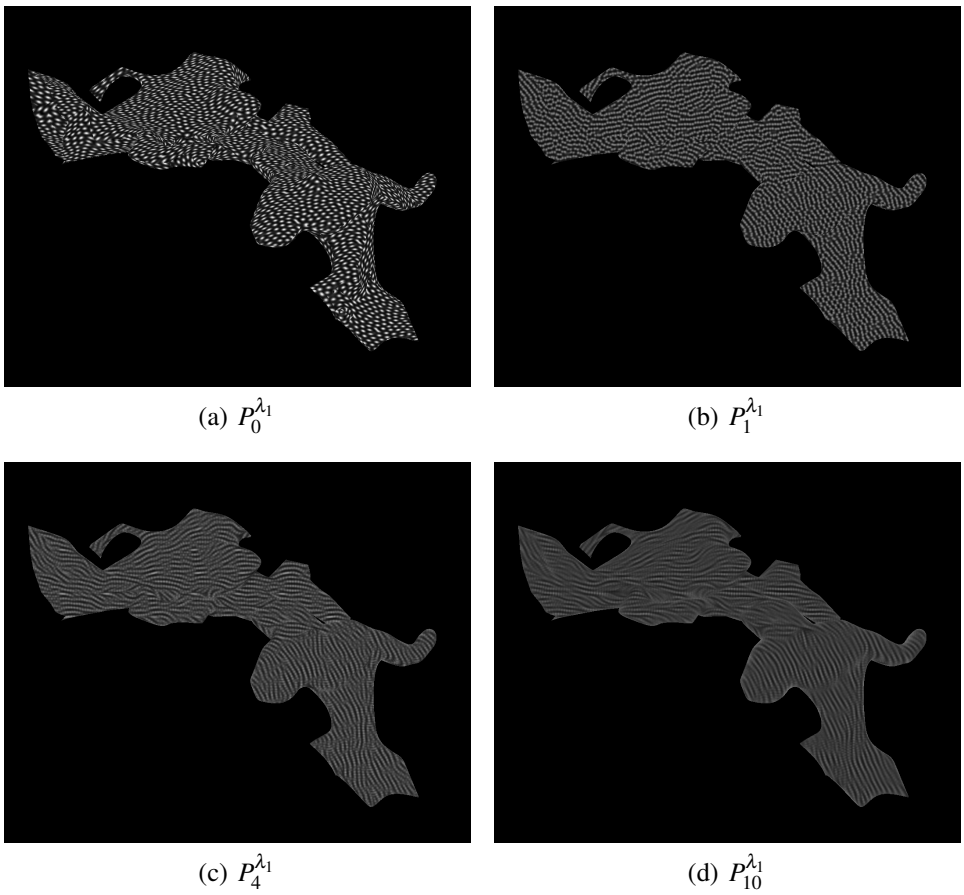
We have discussed how to project the geometry and the corresponding tensor field to image space. With the prepared image space eigenvectors and the input noise texture, mapped to geometry, advection can be done. In our method, we use a simple Euler integration applied to both vector fields. With Euler's method some particle can be followed along a stream. In our case, we cannot calculate streamlines at each position of both vector fields, as normally done in LIC. We directly advect the noise input texture with the vector fields given. This decision was based on the fact that massively parallel architectures like modern GPUs are able to perform this task in parallel for each pixel a hundred times per second. Formally, the advection step can be described as follows: First, we assume an input field P to be a continuous function, defined in a two-dimensional domain.

Definition 4.8 (P-mapping) *Let P be a discrete, two-dimensional field. The continuous mapping function f_P is then defined as:*

$$f_P : (x,y) \rightarrow p, \text{ with } x,y,p \in [0,1].$$

The mapping function f_P is a function returning the input value of the field P at a given position. Continuity is ensured with interpolating values in between. With this in mind, the advection as iterating process can now be described with:

Definition 4.9 (Advection Iteration) *Let P be a discrete, two-dimensional grid with a continuous mapping function f_P. Furthermore, let $\beta_{x,y}$ be the geometry*



(a) $P_0^{\lambda_1}$ (b) $P_1^{\lambda_1}$

(c) $P_4^{\lambda_1}$ (d) $P_{10}^{\lambda_1}$

Figure 4.7: *Advection of eigenvector field, corresponding to λ_1, snapshotted at different iterations. In (a), the geometry mapped input noise can be seen as originator of the iteration. Images (b), (c), and (d) show the advection iteration after 1,4, and 10 iterations with $k = 0.05$.*

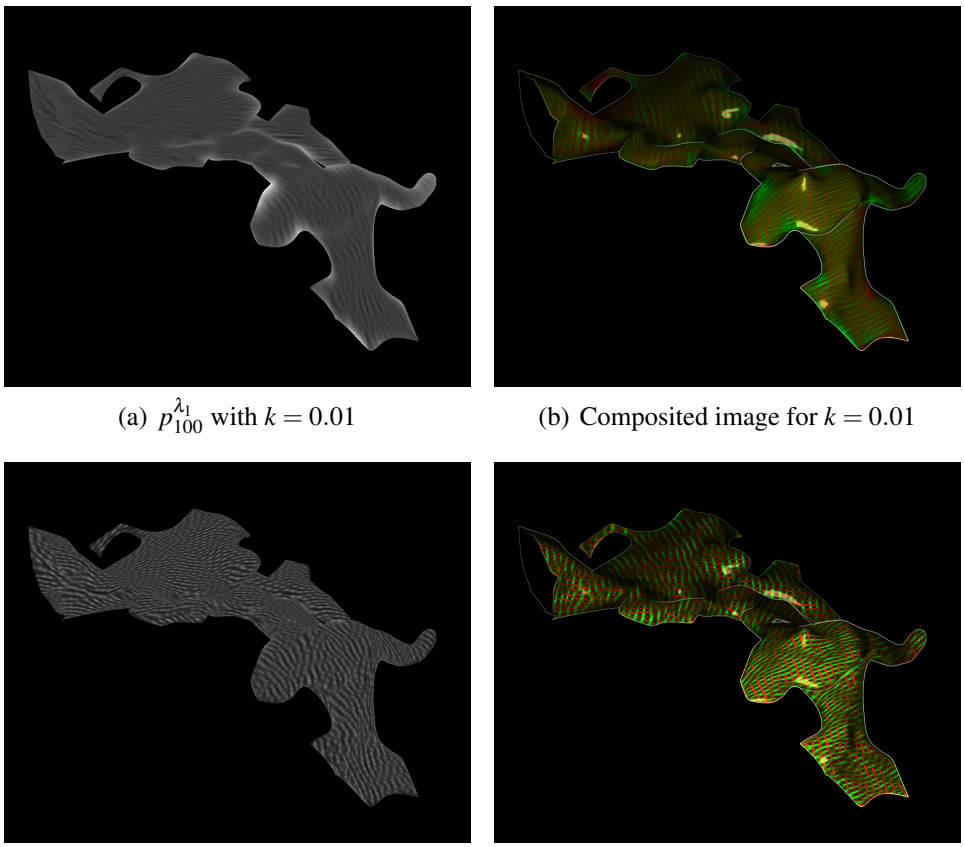
mapped noise field. Then, the grid can be advected cell-wise using an iterative process:

$$\forall x, y \in [0,1] : \forall \lambda \in \{\lambda_1, \lambda_2\} :$$

$$P_0^{\lambda} = \beta_{x,y},$$

$$P_{i+1}^{\lambda} = k \cdot \beta_{x,y} + (1-k) \cdot \frac{f_{P_i^{\lambda}}(x + v'_{\lambda_x}, y + v'_{\lambda_y}) + f_{P_i^{\lambda}}(x - v'_{\lambda_x}, y - v'_{\lambda_y})}{2}.$$

The iterative advection process has to be done for each eigenvector field separately with separate input and output fields p_i. The value at a given point is a mix of the input noise and the iteratively advected input noise from the prior steps. The



(a) $p_{100}^{\lambda_1}$ with $k = 0.01$ (b) Composited image for $k = 0.01$

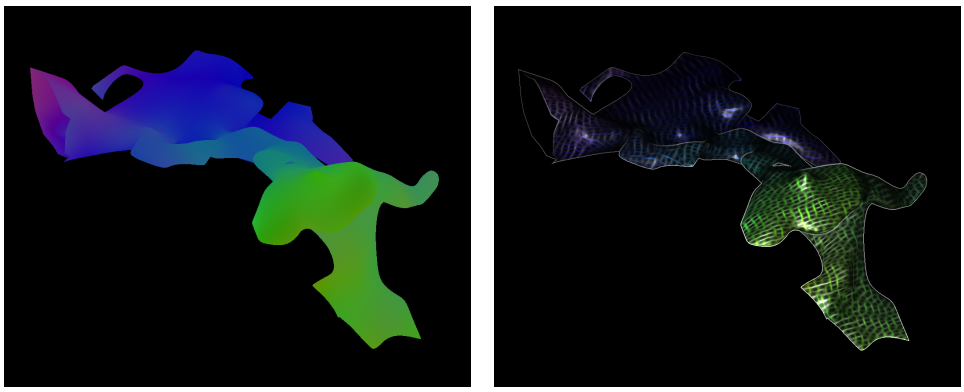
(c) $p_{100}^{\lambda_1}$ with $k = 0.09$ (d) Composited image for $k = 0.09$

Figure 4.8: *Advection of eigenvector field, corresponding to λ_1, using different values for k. A smaller value results in a more smooth and blurred advection/composited image, whereas a larger value,* 0.09 *in our case, results in a more coarse surface and, therefore, in a more clear and contrast-rich composited image.*

ratio between the original input noise β and the previously advected image is determined by k. The scalar factor k defines the "roughness" of the advected image, as illustrated in Figure 4.8. Since the eigenvectors v'_{λ_j} do not have an orientation, the advection has to be done in both directions. The iteration can be stopped when the value change exceeds a threshold, which is, depending on k, reached after a few iterations. In Figure 4.7, a series of images shows the advection process after several iteration steps.

4.6 Compositing

An ultimate rendering pass composes the temporary textures for final visualization. Whereas pixels that are not part of the original rendering of the geometry



(a) $c_p^{FA \cdot v_{\lambda_i}}$ colormap on surface (b) Composited image

Figure 4.9: *Composition of both eigenvector fields in (b) using the $c_p^{FA \cdot v_{\lambda_i}}$ colormap from Section 2.3.1, which is, just for illustration, separately shown in (a). Here, only the major eigendirection is colored and the second, minor direction is colored in gray to visually separate them from each other.*

are discarded using the information from the depth buffer, for all other pixels the color values at a point (x,y) in image space after k iterations is defined by the colormap in Definition 4.10.

Definition 4.10 (Coposition to RGB Space) *Let $P_k^{\lambda_1}$ and $P_k^{\lambda_2}$ be the previously advected fields for both eigenvector fields after k iterations. Furthermore, e is the silhouette image and $light(\mathscr{L}_{x,y})$ an arbitrary lighting function. Then, the composition of $p_k^{\lambda_1}$ and $p_k^{\lambda_2}$ to RGB space is defined by:*

$$R = \frac{r \cdot f_{p_k^{\lambda_2}}(x,y)}{8 \cdot f^2_{p_k^{\lambda_1}}(x,y)} + e_{x,y} + light(\mathscr{L}_{x,y}),$$

$$G = \frac{(1-r) \cdot f_{p_k^{\lambda_1}}(x,y)}{8 \cdot f^2_{p_k^{\lambda_2}}(x,y)} + e_{x,y} + light(\mathscr{L}_{x,y}), \text{ and}$$

$$B = e_{x,y} + light(\mathscr{L}_{x,y}).$$

The scalar factor r is used to blend between the two chosen tensor directions. Definition 4.10 is designed to increase the intensity of an eigenvector field's LIC value at a given point, when the other eigenvector field's value is low at this point. This creates a mesh resembling the tensor field's structure, as Figure 4.5 and Fig-

ure 4.8 already have illustrated. To reduce the effect of light sources on the color coding, we use a separate lighting function *light* that, while manipulating the intensity, does not effect the base color of the mesh structure. Even though Blinn-Phong shading [Bli77] has proven to provide the required depth cues, additional emphasis of the third dimension using depth-enhancing color coding has proven to provide a better overall understanding of the data [CCG+08].

This step furthermore discloses some possibilities. It is also possible to blend in a color map, based on fractional anisotropy or the eigenvalues, just like those introduced in Section 2.3.1. Figure 4.9 shows such a combination of colormapping, using $c_p^{FA \cdot v_{\lambda_i}}$ and our compositing method.

*Konrad Ernst Otto Zuse, *1910 - †1995. Computer pioneer, whose greatest achievement was the world's first functional, programmable computer, the Z3.*

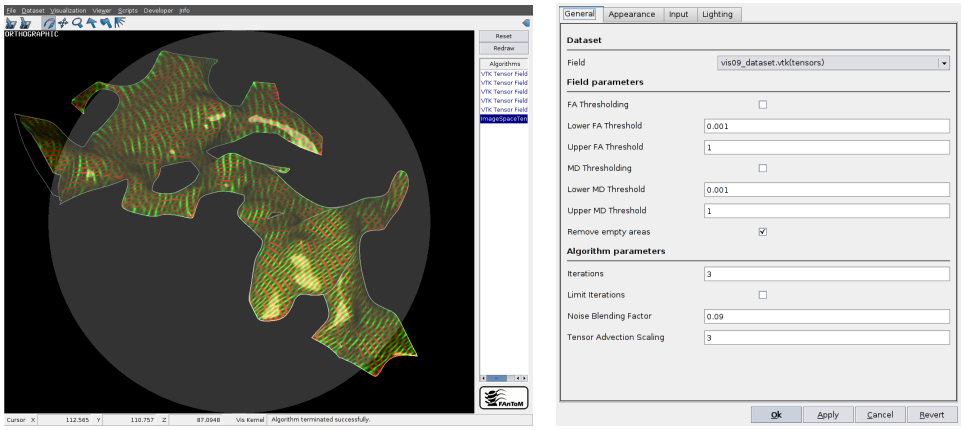
5 Implementation

Our implementation is not limited to a special kind of geometry. It is able to handle almost every tensor field defined on a surface. It is, for example, possible to calculate an isosurface on a derived scalar metric, like fractional anisotropy or on a second data set to generate a surface in a three-dimensional data domain. Other methods include hyper-stream surfaces [DH92], wrapped streamlines [ESM+05], or domain-dependent methods like dissection-like surfaces presented in [ASH+09]. The only requirement for the surface is that it is non-selfintersecting and smooth normals are provided as they are required for the projection step and for proper lighting. The noise texture can be pre-calculated in a pre-processing step or stored in a file as it is independent of the data.

5.1 Framework

Before concretely talking about an implementation, it is necessary to specify the framework needed. Our method is embedded into the framework, the software system "FAnToM" (Field Analysis using Topology Methods) (cf. Figure 5.1), developed at the Abteilung für Bild- und Signalverarbeitung at the Universität Leipzig, is offering. It is a modular system, able to load and manage several kinds of data sets, as well as modifying, processing, or visualizing the data. In addition, it offers an C++ API for algorithms, which can then use the data and a very powerful graphics engine to implement new visualization methods or data processing algorithm.

Our novel method uses this API and the graphics engine, namely "FGE", to implement all the CPU-side parts into a FAnToM-algorithm. Every algorithm can



(a) FAnToM GUI (b) Algorithm's profile dialog

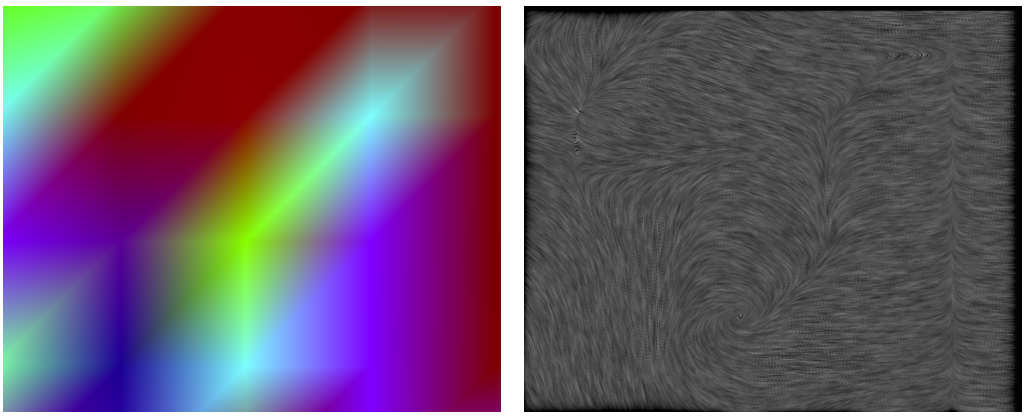
Figure 5.1: *Screenshot of FAnToM's GUI, showing our algorithm running (a) and the corresponding profile dialog (b).*

provide a profile of parameters, which can be modified by the user using a GUI dialog FAnToM is generating. Our implementation uses this profile to allow the user to adopt several parameters, like, for example, k, the blending factor from Definition 4.9 (cf. Figure 5.1(b)).

To effectively do image-space based rendering, the OpenGL shading language GLSL is used, which offers a C-like syntax optimized for strongly local (pixel-wise) parallelism on the graphics processing unit (GPU). All the four steps, as described in Figure 4.1, are implemented in GLSL and, therefore, runs on the GPU parallel for every pixel. The data transfer between these steps is done using textures. They offer four floats in interval $[0,1]$ per pixel and are ideal for data transfer. But, of course, they also have some restrictions that we needed to take care about. In the following sections, we illuminate our visualization method from the implementation point of view and point out some difficulties and our data organization and handling to circumvent them

5.2 Projection

The first step projects the geometry into image space, simply by rendering the geometry and pre-calculating the Phong light intensity $\mathscr{L}_{x,y} \in [0,1]$ at every rendered fragment with the coordinates x and y. In the same step the tensors are projected as well. When the tensors are symmetric, it is sufficient to transfer six floating-point values per vertex to the GPU. In our case, two three-dimensional texture coordinates are used per vertex to upload the tensor information along with the geometry. Assuming the tensor T is now available on the GPU, it is now possible to map the two main directions to the surface described by the normal



(a) Eigenvector Colormap (b) Artifacts

Figure 5.2: *Artifacts in an extremely zoomed part of a tensor field. The vortices in (b) result from sign flips in the eigenvector field and go along these sign-flips. For illustration, the corresponding part of the eigenvector field is shown in (a). If sign flips are ignored, the field at this area is an uniform flow from left to right/right to left.*

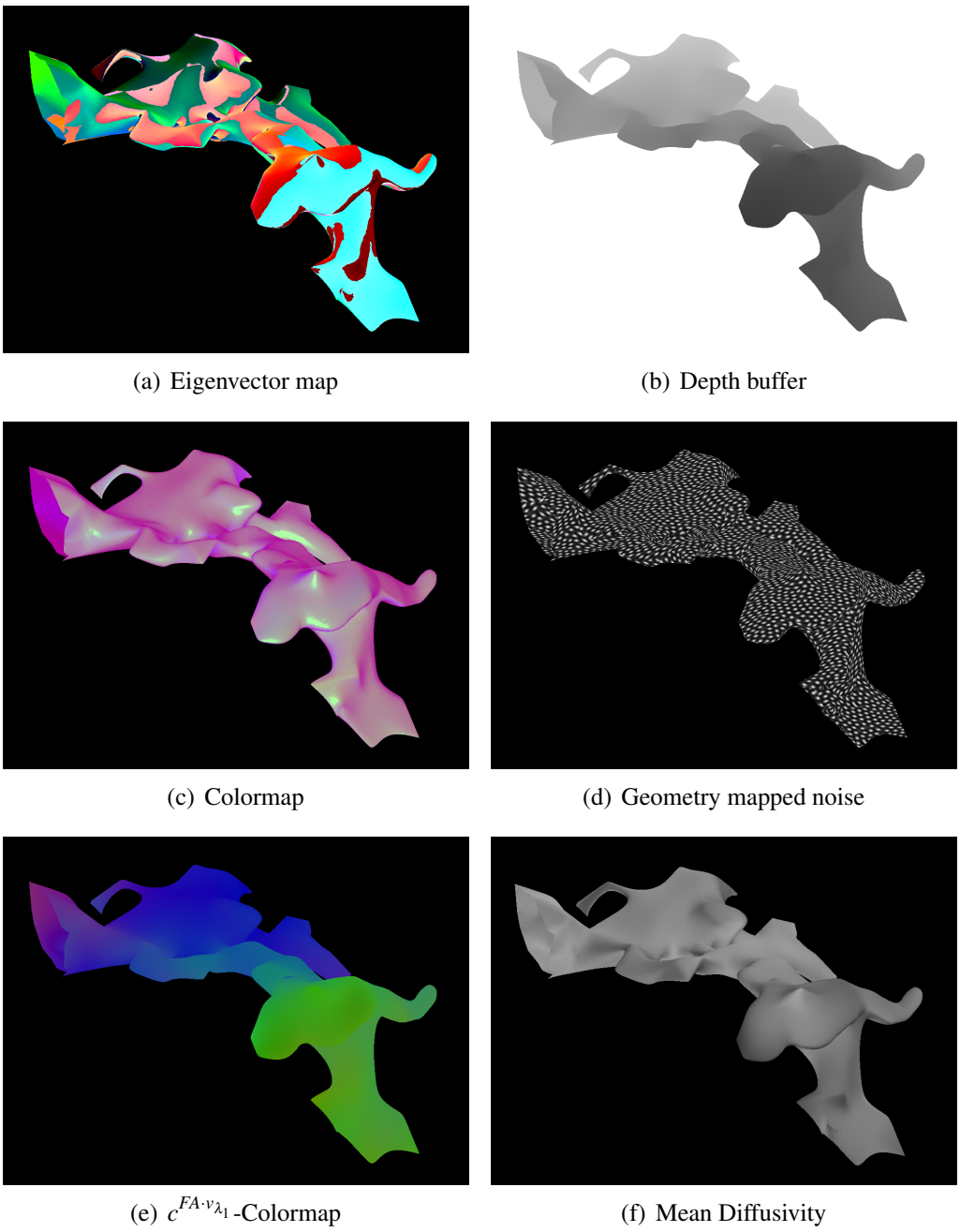
n at the current vertex using Definition 4.1. This projection is implemented on a per-vertex basis in the vertex shader. In contrast, to ensure proper interpolation, eigenvalue decomposition and eigenvector calculation together with image space projection need to be done in the fragment shader. Since the eigenvectors are orientation-less, it is possible to have sign flips between adjacent vertices.

If the interpolation takes place after the eigenvector decomposition, these changes in sign can render the interpolation useless , which could cause severe artifacts during advection, as shown by example in Figure 5.2.

The projection step also includes mapping the noise texture to the geometry. Calculating each vertex' position in one voxel, using the equations from Section 4.3, can be done along the tensor projection. The GPU interpolates those values for each fragment, where it can be used to determine the noise texture element to use.

Since texture space is limited on our hardware, an nVidia GeForce 8800 GTS, to four bound textures per rendering pass and active framebuffer object, it is important to store as much information as possible in each texture. Most data is calculated during the projection step and needs to be stored in, at most, four textures. Table 5.1 provides an overview of the data that needs to be transferred to consecutive steps.

Since $\lambda_1 \geq \lambda_2$, it is not necessary to transfer both values. Normalizing the vector (λ_1, λ_2) using the maximum norm as shown in Definition 4.3 and Equation 4.2, it is enough to transfer the smaller λ_2 because λ_1 always is one. Also, the eigenvectors v'_{λ_1} and v'_{λ_2} need to be scaled since textures are used for transportation



(a) Eigenvector map

(b) Depth buffer

(c) Colormap

(d) Geometry mapped noise

(e) $c^{FA \cdot v_{\lambda_1}}$-Colormap

(f) Mean Diffusivity

Figure 5.3: *Textures showing the output of the projection step. Table 5.1 lists the contents of each of these textures.*

where each value must be in the interval $[0, 1]$.

All these calculations are done by the GPU in our implementation, using the OpenGL's GLSL shading language, offscreen in a *framebuffer object* (FBO).

	texture	channel
v''_{λ_1}	Figure 5.3(a)	R,G
v''_{λ_2}		B,A
fractional anisotropy	Figure 5.3(c)	R
$\mathscr{L}$		G
λ_2		B
β_{t_x,t_y}	Figure 5.3(d)	A
depth buffer	Figure 5.3(b)	L
$c^{FA \cdot v_{\lambda_1}}$	Figure 5.3(e)	RGB
MD	Figure 5.3(f)	A

Table 5.1: *Memory alignment of intermediate textures created during projection step. Texture four ($c^{FA \cdot v_{\lambda_1}}$ and MD) is purely optional and may be used to transfer further colormaps and metrics, as in our case.*

Thus, we avoid the need of rendering the geometry multiple times or even doing those calculations on the CPU, which leads to a large speed gain.

5.3 Silhouette Detection

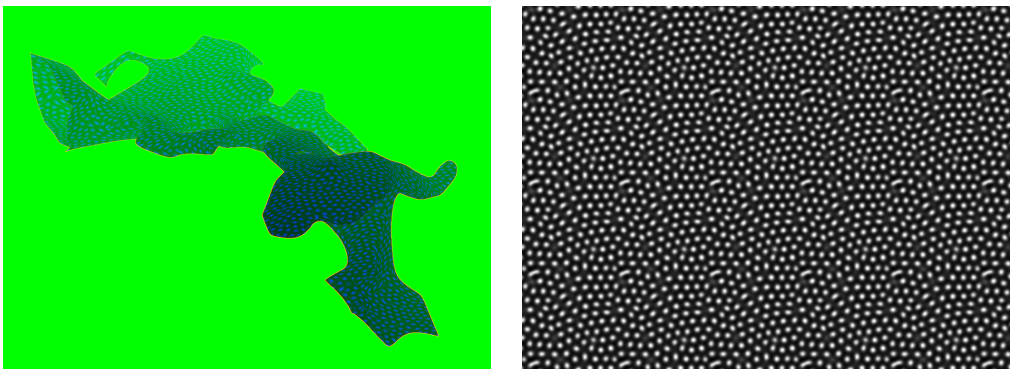
The next step is to apply an edge detection filter, in our case a discrete Laplacian filter kernel, on the depth map, which we have implemented on the GPU using GLSL shaders as a separate offscreen render pass. We merge several input data into one texture to decrease the number of textures needed later. We use red, green, blue, and alpha channels of the edge detection shader output texture, as summarized in Table 5.2.

	texture	channel
edge	Fig 5.4	R
depth buffer	Fig 5.4	G
β_{t_x,t_y}	Fig 5.4	B
β	Fig 5.4	A

Table 5.2: *Input and output textures during the edge detection step.*

This shows that we store the result of the edge detection filter, and we also store the depth buffer value used to calculate the edges, the unprocessed input noise field and the geometry mapped noise in the green, blue, and alpha channels (cf. Figure 5.4). The used input textures are:

- the color map (Figure 5.3(c))



(a) RGB channels (b) Alpha channel

Figure 5.4: *Output of edge detection step. Left: the depth buffer, the edges and geometry mapped noise β_{t_x,t_y} in the red, green, and blue channels, respectively; right: the alpha channel containing the unprocessed input noise β.*

- the input noise texture (Figure 4.3)
- and the depth buffer (Figure 5.3(b)).

The advection step, directly following the edge detection, can "grab" its needed data directly from one texture. During advection, it is not advisable to have all those values in one texture, since just the geometry mapped noise β_{t_x,t_y} and the depth buffer is used. The greatest advantage is achieved during output processing, since the depth buffer is not needed as separate texture, which saves texture overhead. We also transfer the unprocessed input noise in this texture, since our implementation is able to toggle which noise field is used during advection, because near-planar geometry does not need any complex geometry mapping.

5.4 Advection

The advection step iterates the geometry mapped noise texture using the eigenvector color map from the projection step (see Figure 5.3(a)). This step is summarized in the pseudocode for Algorithm 5.1.

The pseudocode provided for Algorithm 5.1 shows the steps that need to be done in each iteration on the CPU, where the output texture from the last iteration is used as input.

In practice, not needed textures have to be freed, since they allocate a lot of memory on the graphics board. It is possible to break iteration after a fixed number of passes and complete the frame. This ensures higher framerates and a more smooth user experience. The consecutive rendering steps can then work on this "intermediate" advection texture to produce an output. In the next frame, the iteration loop can begin at the last iteration texture of the last frame. If done this way,

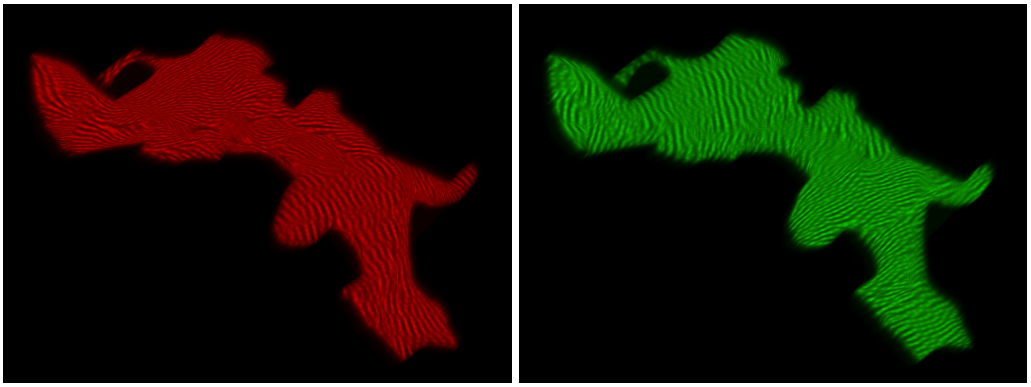


Figure 5.5: *Advection texture. Left: red channel containing the advection image of eigenvector field 1; right: green channel containing the advection image of eigenvector field 2.*

special care has to be taken for texture management and the framebuffer object. Both must not be destroyed at the end of each frame. This iteration allows the GPU code, summarized as Algorithm 5.2, to be executed iteratively and per pixel on the GPU with the bound textures as its input respectively output. Figure 5.5 shows the red and green channels of this output texture after several iterations. Note that, due to the data merging in the prior step, there need to be just three input texture bindings per iteration, instead of four, for depth buffer, edge detection texture, the eigenvector color map and the last iterations output texture. Together, both algorithms implement the advection iteration described in Section 4.5. After some iterations, the geometry-mapped noise gets advected more and more and is ready to be finally processed for output.

Algorithm 5.1 Advection iteration code executed on CPU

```
// bind required input textures
bindTexture(edgeTexture)
bindTexture(eigenvectorColormap)
// create two textures used rotational as input or output
advection1 ← createTexture()
advection2 ← createTexture()
advectionInput ← reactionDiffusionTexture
for i = 1 to MAX do
    bindTexture(advectionInput)
    // select output texture and set it as next input texture
    if i modulo 2 = 0 then
        setOutputTexture(advection1)
        advectionInput ← advection1
    else
        setOutputTexture(advection2)
        advectionInput ← advection2
    end if
    // render bound textures on quad in FBO
    renderQuadOffscreen()
end for
```

Algorithm 5.2 Advection iteration code executed on graphics hardware for each $t \in \{(x,y)|x,y \in [0,1]\}$

Require: edgeTexture: output texture from silhouette detection step
Require: eigenvectorColormap: eigenvectors from projection step
Require: advectionInput: advection step output from previous step
Require: t: to be current texture coordinate
Require: k: to be the ratio from Definition 4.9
Require: edgeBlend: defining the edge intensity during advection
Require: discardable: a function returning true if both directions and their anti directions show to a fragment not containing geometry (uses depth buffer)

// get data
float $depth \leftarrow edgeTexture_t$.G
float $edge \leftarrow edgeTexture_t$.R
vec4 $tensor \leftarrow eigenvectorColormap_t$.RGBA
// unscale and separate eigenvectors, scale to texture space
$tensor \leftarrow (tensor$ - vec4(0.5, 0.5, 0.5, 0.5))* 2.0;
vec2 $ev1 \leftarrow$ vec2($\frac{tensor.R}{textureWidth}$, $\frac{tensor.G}{textureHeight}$)
vec2 $ev2 \leftarrow$ vec2($\frac{tensor.B}{textureWidth}$, $\frac{tensor.A}{textureHeight}$)
if discardable($depth$, $ev1$, $ev2$) **then**
  return vec4(0.0, 0.0, 0.0, 1.0)
end if
// calculate position
vec2 $a1 \leftarrow t - s * ev1$
vec2 $a2 \leftarrow t - s * ev2$
// since the eigenvectors do not have an orientation: advect in both directions
vec2 $a1n \leftarrow t + s * ev1$
vec2 $a2n \leftarrow t + s * ev2$
// the GPU interpolates texels automatically
float $aNoise1$= 0.5 * ($advectionInput_{a1}$.R + $advectionInput_{a1n}$.R)
float $aNoise2$= 0.5 * ($advectionInput_{a2}$.G + $advectionInput_{a2n}$.G)
float $noise \leftarrow edgeTexture_t$.B
return (vec4($aNoise1$, $aNoise2$, 0.0, 1.0) * (1.0-k)) + vec4($noise * k$) + vec4($edge * edgeBlend$)

5.5 Compositing

The final output processing includes blending both advected eigenvector fields with light information and the geometry edges as well as clipping fragments using the depth buffer. The last output texture created during advection iteration gets bound, as well as the color map (Figure 5.3(c)) containing lighting information, and finally, the edge detection output texture Figure 5.4 for clipping and edge blending. The pseudocode for Algorithm 5.3 summarizes output processing.

Algorithm 5.3 output processing code on GPU

Require: advection: last output texture generated during advection
Require: edgeTexture: output texture from silhouette detection step
Require: projectionColormap: output texture from projection step
Require: ratio: blending ratio between eigenvector field 1 and 2
Require: t: to be current texture coordinate

```
// discard fragments not belonging to the geometry
float depth ← edgeTexture_t.G
if depth ≥ ε then
    discard fragment
end if
// get data
vec4 edge ← edgeTexture_t.R
vec4 depth ← edgeTexture_t.G
float light ← projectionColormap_t.G
float evLIC1 ← advection_t.R
float evLIC2 ← advection_t.G
// calculate output
float red ← 2.0 * evLIC1
float green ← 2.0 * evLIC2
float factorR ← ratio / (4*sqr(red))
float factorG ← (1−ratio) / (4*sqr(green))
// lighting
float shine ← light − 0.6
if shine < 0 then
    shine ← 0.0
end if
// do additional color mapping
...
// combine light, both LIC, edge and depth
return (light * 1.5) * (vec4(green * factorR, red * factorG, 0.0, 1.0) +
vec4(shine)) + (1.0 − depth)*edge
```

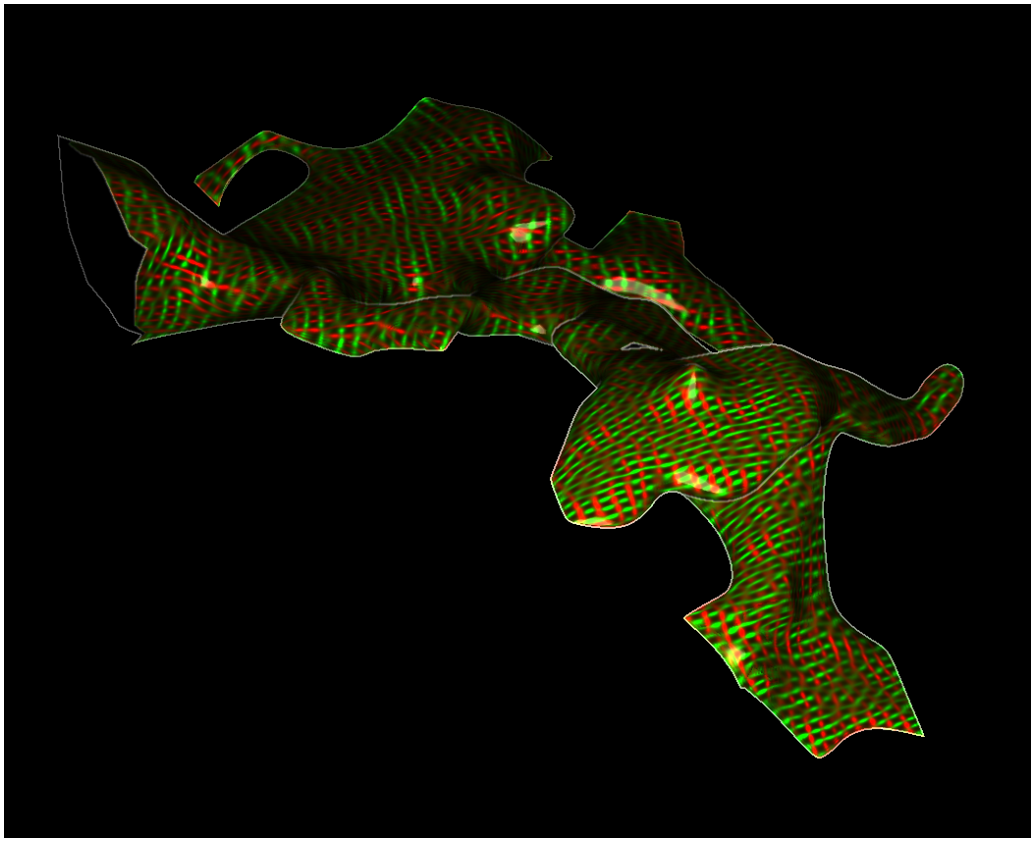
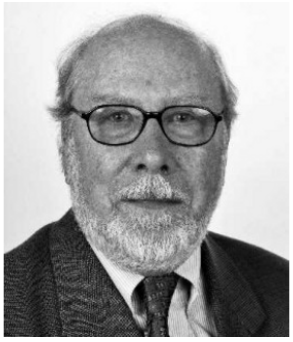


Figure 5.6: *The final image produced by the output processing shader with lighting.*

As mentioned in Section 4.6, several modifications are possible. Since the color map bound in this step also contains the fractional anisotropy, λ_2 and implicitly λ_1 it is possible to blend-in those values to emphasize additional tensor field features. In Figure 5.6, both eigenvector fields were blended with the geometry edges and with Phong luminance, calculated earlier in projection step.

In Chapter 6, many more compositions are shown to illustrate the rich set of possibilities.



*Niklaus E. Wirth, *1934*
Designer of several widely used programming languages like Pascal, Modula, and PL/0.

6 Results

We have introduced a method to create a fabric-like surface tensor LIC in image space, similar to the one introduced in [HFH+04]. We used ideas from [LJH03] to transform the algorithm into image space. Our implementation, using this method, is able to reach frame rates high enough for realtime user interaction. The only bottleneck is the hardware's ability in rendering large and triangle-rich geometry. All further steps can be done in constant time, see Table 6.1.

6.1 Artificial Test Data Sets

We first applied our method to artificial test data sets that have complex topology: a torus, the Bretzel5, and the Tangle data set (cf. [KHH+07]), defined as implicit surfaces:

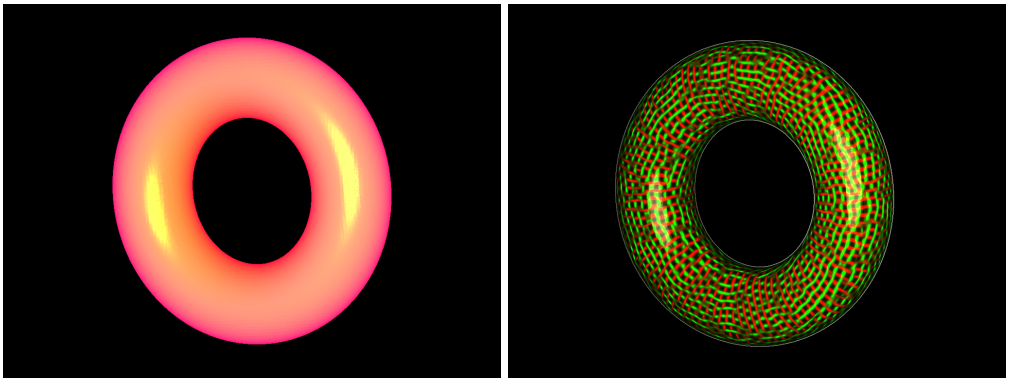
Definition 6.1 (Test datasets) *Let (x,y,z) be a point in $\mathbb{R}^3$. Then, the surfaces are defined implicitly by:*

Torus: $0 = (1-\sqrt{x^2+y^2})(1-\sqrt{x^2+y^2})+z^2-0.125$

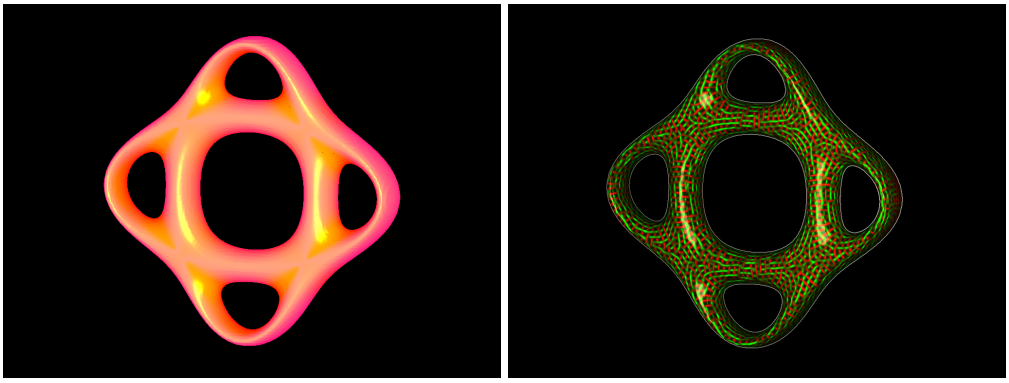
Bretzel5: $0 = ((x^2+.25*y^2-1)*(.25*x^2+y^2-1))^2+z^2-0.1$

Tangle: $0 = x^4-5*x^2+y^4-5*y^2+z^4-5*z^2+11.8+w.$

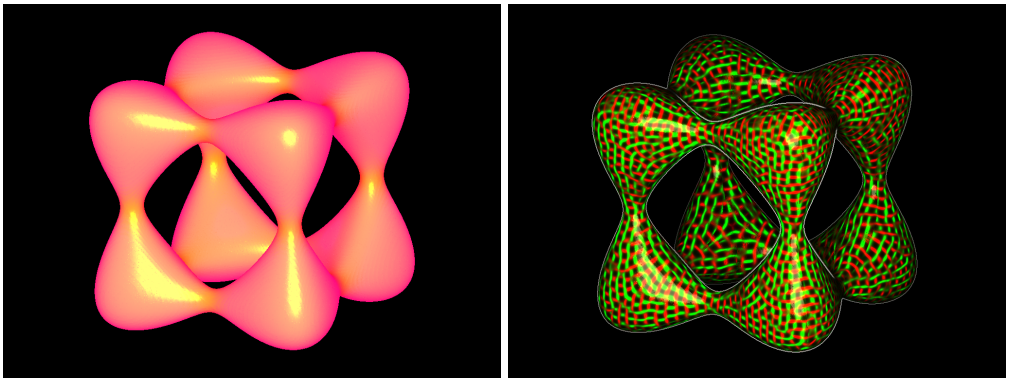
We used the Laplacian on the surfaces as tensor fields. The results displayed in Figure 6.1 show that neither the topology nor our artificial parameterization of the input noise texture influences the quality of the final rendering.



(a) Torus



(b) Bretzel5



(c) Tangle

Figure 6.1: *Analytic test data sets. We applied our method to isosurfaces and the scalar field's Laplacian to demonstrate the suitability for complex surfaces. Shown here is the original surface (left) and the final image using our method for a torus, Tangle, and Bretzel5 data set (Definition 6.1).*

6.2 Diffusion Tensor Imaging Datasets

In the previous chapters, we already have shown many images produced by our method. In this section, we only show several of these images in large size.

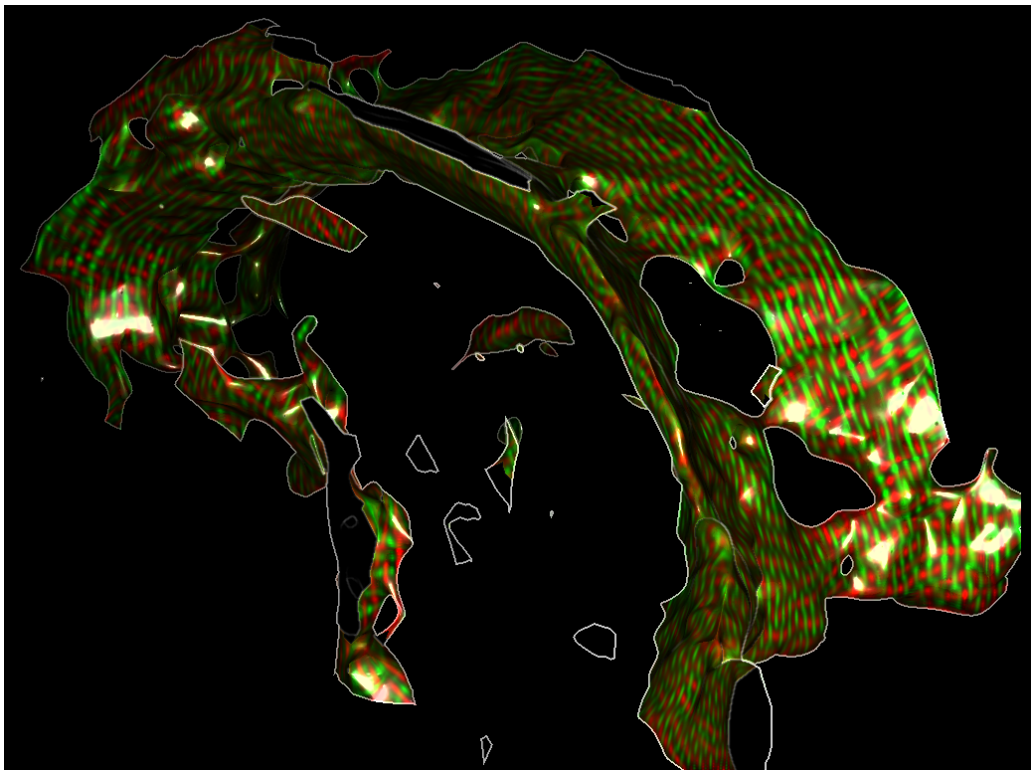


Figure 6.2: *Diffusion tensor data set of a human brain. The* corpus callosum *has been extracted using the fractional anisotropy as scalar metric with the isosurface algorithm at* $FA = 0.7$.

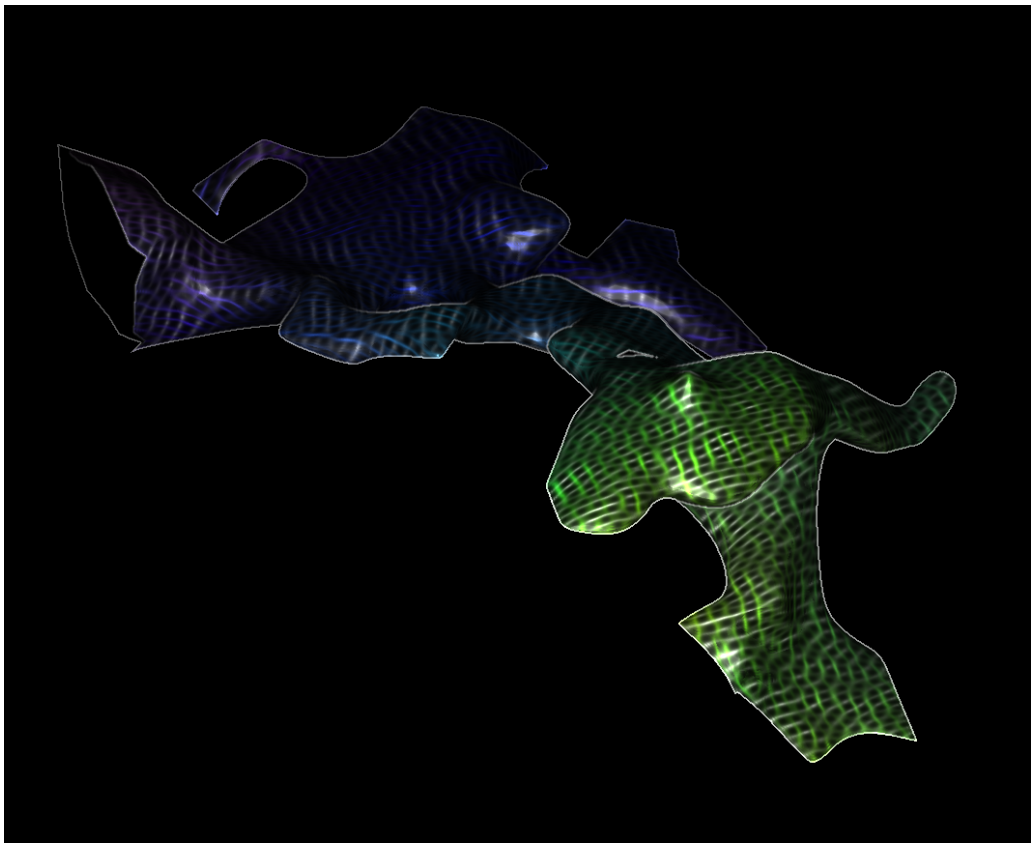


Figure 6.3: *Small part of the* corpus callosum. *The main eigendirection of the diffusion tensor data set is colored using* $c_p^{FA \cdot v_{\lambda_1}}$.

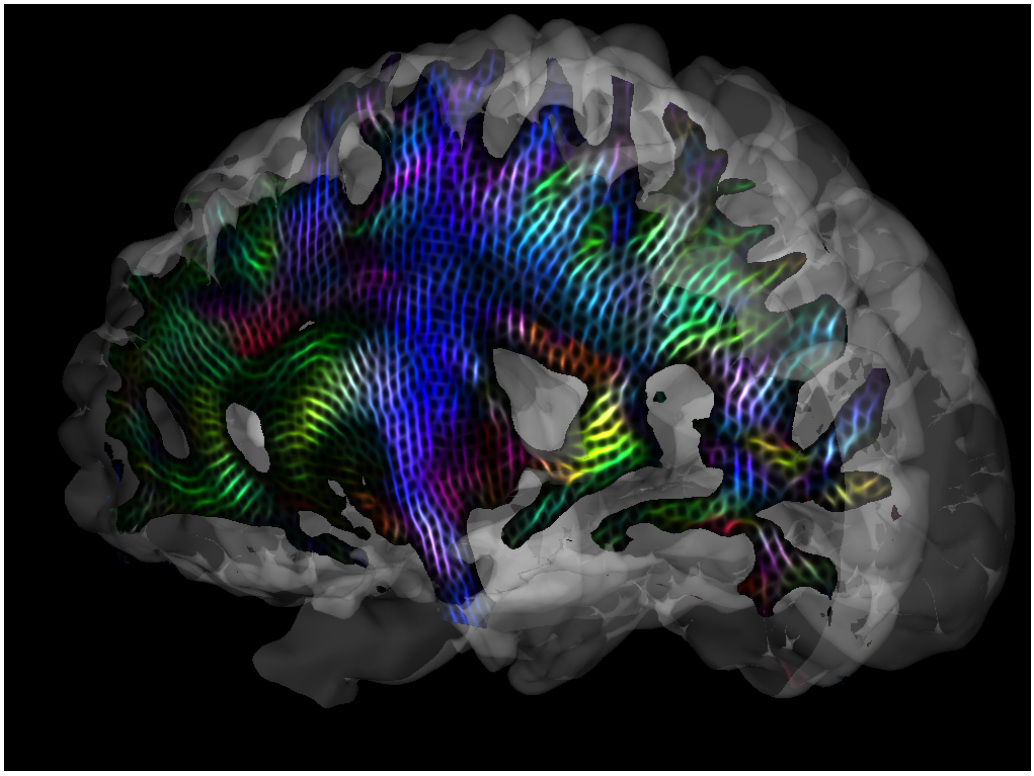


Figure 6.4: *Diffusion tensor data set of a human brain. We employed the method by Anwander et al. [ASH+09] to extract a surface following neural fibers and applied our method with an alternative color coding that is more suitable and can be incorporated more easily into medical visualization tools.*

6.3 Modification for Medical Data Processing

Even though many higher-order methods have been proposed, due to scanner, time, and funding limitations, second-order tensor data is still dominant in clinical application. Medical second-order diffusion tensor data sets differ from engineering data sets because they indicate one major direction whereas the secondary and ternary directions only provide information in areas, where the major direction is not well-defined, i.e., the fractional anisotropy—a measure for the tensor shape—is low. Almost spherical tensors, which indicate isotropic diffusion, occur in areas where multiple fiber bundles traverse a single voxel of the measurement or when no directional structures are present. Therefore, we modulate the color coding using additional information: In areas where one fiber direction dominates, we only display this major direction using the standard color coding for medical data sets, where x, y, and z alignment are displayed in red, green, and blue, respectively. In areas where a secondary direction in the plane exists, we display this information as well but omit the secondary color coding and display the secondary direction

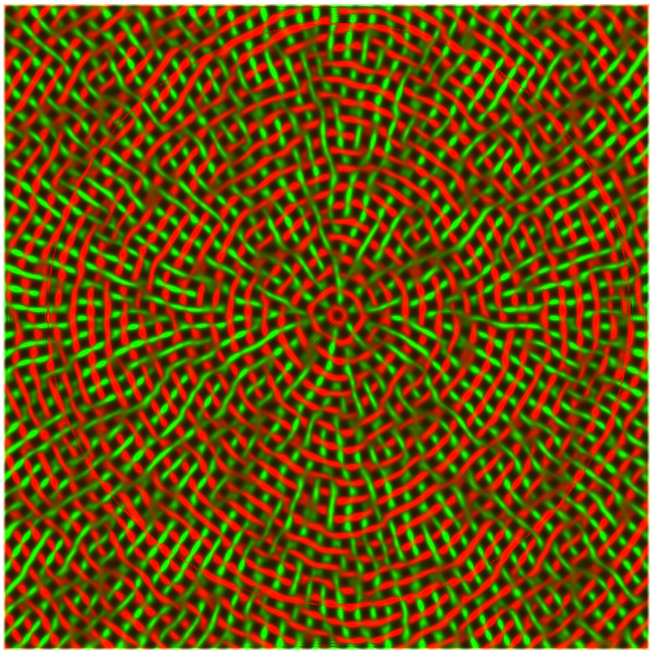


Figure 6.5: *Slice in an analytic strain tensor field.*

in grayscale and always below the primary direction (cf. Figure 6.4). We use the method of Anwander et al. [ASH+09] to extract surfaces that are, where possible, tangential to the fiber directions. Hence, we can guarantee that the projection error introduced by applying our method in the surface's domain remains sufficiently small. Even in areas where the fractional anisotropy is low and the color coding does no longer provide directional information, such as in some parts of the pyramidal tract in Figure 6.4, the texture pattern still provides this information.

6.4 Arbitrary Datasets

Our technique is designed as flexible as possible, thus allowing us to visualize other tensor fields than medical DTI datasets. Generally, arbitrary second-order tensor fields can be visualized using this technique. This may be interesting especially for engineering sciences or physics, since non-symmetric tensor fields, like the analytical point load dataset, play an important role there. To give an example of our approach in mechanics, Figure 6.5 shows a visualization of a slice inside a strain tensor field, giving an imagination of its physical structure.

6.5 Performance

As indicated before, the only "bottleneck" in the visualization pipeline that is strongly geometry-dependent is the projection step. Since the surface needs to be re-rendered in case of user interaction, the performance measures of our method include re-rendering the geometry. The frame rate with geometry not being moved and, therefore, making the projection step and the edge detection step unnecessary, is considerably higher. Our implementation requires only few advection iterations per frame, which ensures high frame rates and smooth interaction. To make the frame rates comparable, in the following tables, user interaction is assumed and, therefore, rendering a single frame always consists of

- one projection step, including geometry rendering;
- one edge detection pass;
- three advection iterations; and
- one output processing pass.

As seen in the previous sections, fragments not belonging to the geometry are discarded as soon as possible without using deferred shading. This also generates performance gain in advection and output processing. In Table 6.1, a selection of data sets with their corresponding number of triangles and tensors are listed. The frame rates shown were measured on an AMD Athlon(tm) 64 X2 Dual Core Processor 3800+ (512K L2 Cache) with a NVIDIA G80 GPU (GeForce 8800 GTS) and 640MB of graphics memory at a resolution of 1024×768 pixels.

The assumption that geometry rendering with projection is the weakest element in this pipeline and that edge detection, advection, and output processing performs at a data-independent frame rate, is confirmed by the frame rate shown in Table 6.2. It confirms that for large geometries, rendering the geometry alone is the dominating component. Since the vertex-wise calculations during projec-

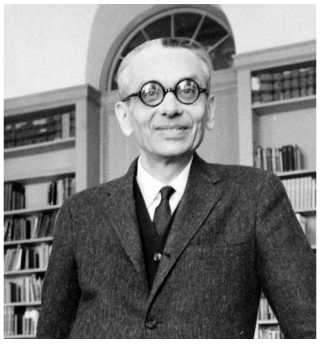
figure	no. triangles	no. tensors	fps	fps (Phong only)
Figure 6.4	41472	63075	32	61
Figure 5.6	58624	88803	30	60
Figure 6.2	571776	861981	14	16

Table 6.1: *Frames per second (fps) for different data sets with given number of triangles and numbers of tensors. The frame rates are compared to simple rendering of the geometry using Phong shading. The frame rates were measured for an AMD Athlon(tm) 64 X2 Dual Core Processor 3800+ (512K L2 Cache) with an NVIDIA G80 GPU (GeForce 8800 GTS) and 640MB of graphics memory at a resolution of* 1024×768 *pixels.*

figure	∅ geometry share
Figure 6.4	72%
Figure 5.6	69%
Figure 6.2	90%

Table 6.2: *The time required by the GPU to rasterize the geometry in comparison to the overall render times, including all steps. The time used to render the geometry clearly dominates the rendering times and reaches up to 90 percent of the overall rendering time even for medium sized geometries.*

tion are limited to tensor projection (Definition 4.1) and vertex projection (Definition 4.5), the most expensive calculations during projection are executed per fragment. This means that the expensive eigenvalue decomposition and eigenvector calculations are only required for fragments (pixels). To further decouple the calculation effort from the geometry's size, the depth test should be performed before performing the eigendecomposition. This can be achieved by first rendering the projected tensors to a texture, and then computing the decomposition on visible fragments only. Nevertheless, this was not necessary for our current data set and screen sizes where the time required to render the geometry itself clearly dominates the time required to compute the texture pattern in image space. This can be seen in the increasing values in Tables 6.2 and 6.1 with increasing size of vertices rendered.



*Kurt Gödel, *1906 - †1978*
He caused a large uproar with his incompletenes theorem in logics and mathematics.

7 Conclusion and Further Work

We have presented a novel method for rendering fabric-like structures to visualize tensor fields on almost arbitrary surfaces without generating three-dimensional textures that span the whole data set at sub-voxel resolution. Therefore, our method can be applied to complex data sets. As major parts of the calculation are performed in image space, the performance of our algorithm is almost independent of data set size, provided that surfaces can be drawn efficiently, e.g., by using acceleration structures to draw only those parts of the geometry that intersect the view frustum or using ray tracing methods.

Whether the surface itself is the domain of the data, a surface defined on the tensor information, or a surface defined by other unrelated quantities (e.g., given by material boundaries in engineering data or anatomical structures in medical data) is independent from our approach. Nevertheless, the surface has to be chosen appropriately, as only in-plane information is visualized. To circumvent this limitation, information perpendicular to the plane could be incorporated in the color coding, but due to a proper selection of the plane that is aligned with our features of interest, this has not been necessary for our purposes.

One major drawback of our approach is the requirement for smooth normals at every point of the surface to achieve nearly distortion-free texture mapping. This is especially problematic on triangulated surfaces, which need to be smoothed using some subdivision algorithm, introducing a lot of additional geometry, which, on the other side, incriminates the GPU. This vicious circle can be broken by combining geometry refinement and geometry optimization, refining only areas of interest or areas visible by the camera. Modern graphics hardware recently introduced *geometry shaders*, a separate shader invoked before vertex shader, thus

allowing it to dynamically create geometry on the GPU. How these shaders may be useful for solving the mentioned problem has to be investigated.

Due to its image space nature, our approach is not able to achieve the exactly same visual results during advection as the LIC algorithm would achieve. This could cause small areas of the composited surface being blurry under certain awkward perspectives. This may be solved using a more sophisticated integration method during advection in combination with textures, able to store real unclamped floats with their full precision, which is not the case for standard textures.

Another feature, needing more investigation, is the variation of the spot density and spot size for the input noise texture mapped to the geometry. By doing this, it will be possible to let certain metrics, like the eigenvalue field, further influence the input texture, resulting in a more sophisticated visual representation of the tensor field's physical properties.

Especially in medical visualization, higher-order tensor information is becoming increasingly important and different methods exist to visualize these tensors, including local color coding, glyphs, and integral lines. Nevertheless, an extension of our approach is one of our major aims. In brain imaging, experts agree that the maximum number of possible fiber directions is limited (typically, a maximum of three or four directions in a single voxel are assumed). Whereas the number of textures can easily be adapted, the major remaining problem at the moment is a lack of suitable decomposition algorithms on the GPU, which are required to ensure a proper interpolation because image-space techniques, by their very nature, resample the data. Maintaining orientations and assigning same fibers in higher-order data to the same texture globally is not possible today and, therefore, requires further investigation.

List of Figures

List of Tables

List of Algorithms

Bibliography

[ASH+09] Alfred Anwander, Ralph Schurade, Mario Hlawitschka, Gerik Scheuermann, and Thomas R. Knösche. White matter imaging with virtual Klingler dissection. *Human Brain Mapping 2009*, 2009.

[Bar81] A. H. Barr. Superquadrics and angle-preserving transformations. *IEEE Comput. Graph. Appl.*, 1(1):11–23, 1981.

[Bli77] James F. Blinn. Models of light reflection for computer synthesized pictures. In *SIGGRAPH '77: Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, pages 192–198, New York, NY, USA, 1977. ACM.

[Bli82] James F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. In *SIGGRAPH '82: Proceedings of the 9th annual conference on Computer graphics and interactive techniques*, pages 21–29, New York, NY, USA, 1982. ACM.

[BML94] P. J. Basser, J. Mattiello, and D. LeBihan. Estimation of the effective self-diffusion tensor from the nmr spin echo. *J Magn Reson B*, 103(3):247–254, March 1994.

[BP96] Peter J. Basser and Carlo Pierpaoli. Microstructural and physiological features of tissues elucidated by quantitative-diffusion-tensor mri. *Journal of Magnetic Resonance, Series B*, 111(3):209 – 219, 1996.

[CCG+08] Alan Chu, Wing-Yin Chan, Jixiang Guo, Wai-Man Pang, and Pheng-Ann Heng. Perception-aware depth cueing for illustrative vascular visualization. In *BMEI '08: Proceedings of the 2008 International Conference on BioMedical Engineering and Informatics*, pages 341–346, Washington, DC, USA, 2008. IEEE Computer Society.

[CJS93] Z. Cho, J. Jones, and M. Singh. *Foundations of Medical Imaging*. Wiley, New York, 1993.

[DH92] Thierry Delmarcelle and Lambertus Hesselink. Visualization of second order tensor fields and matrix data. In *VIS '92: Proceedings of*

the 3rd conference on Visualization '92, pages 316–323, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[Dü88] Martin J. Dürst. Additional reference to "marching cubes" (letters to the editor). *Computer Graphics*, 22(2):72–73, April 1988.

[ESM+05] Frank Enders, Natascha Sauber, Dorit Merhof, Peter Hastreiter, Christopher Nimsky, and Marc Stamminger. Visualization of white matter tracts with wrapped streamlines. In Cláudio T. Silva, Eduard Gröller, and Holly Rushmeier, editors, *Proceedings of IEEE Visualization 2005*, pages 51–58, Los Alamitos, CA, USA, 2005. IEEE Computer Society, IEEE Computer Society Press.

[GL05] Markus Grabner and Robert S. Laramee. Image space advection on graphics hardware. In *SCCG '05: Proceedings of the 21st spring conference on Computer graphics*, pages 77–84, New York, NY, USA, 2005. ACM.

[HBPA01] Khader M. Hasan, Peter J. Basser, Dennis L. Parker, and Andrew L. Alexander. Analytical computation of the eigenvalues and eigenvectors in DT-MRI. *Journal of Magnetic Resonance*, 152(1):41 – 47, 2001.

[HES08] Mario Hlawitschka, Sebastian Eichelbaum, and Gerik Scheuermann. Fast and memory efficient GPU-based rendering of tensor data. In *Accepted for the Proceedings of the IADIS International Conference on Computer Graphics and Visualization 2008*, 24–26. July 2008.

[HFH+04] Ingrid Hotz, Louis Feng, Hans Hagen, Bernd Hamann, Kenneth Joy, and Boris Jeremic. Physically based methods for tensor field visualization. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 123–130, Washington, DC, USA, 2004. IEEE Computer Society.

[HFHJ09] Ingrid Hotz, Z. X. Feng, Bernd Hamann, and Kenneth I. Joy. Tensor field visualization using a fabric-like texture on arbitrary two-dimensional surfaces. In Torsten Möller, Bernd Hamann, and R. D. Russel, editors, *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*. Springer-Verlag Heidelberg, Germany, 2009.

[HJ05] Charles D. Hansen and Chris R. Jonson. *The Visualization Handbook*. Elsevier Butterworth-Heinemann, 2005.

[Ibe95] Hans Karl Iben. *Tensorrechnung*. Mathematik für Ingenieure und Naturwissenschaftler. Teubner, 1995.

[IOK00] Yuya Iwakiri, Yuuichi Omori, and Toyohisa Kanko. Practical texture mapping on free-form surfaces. In *PG '00: Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*, page 97, Washington, DC, USA, 2000. IEEE Computer Society.

[KHH+07] Aaron Knoll, Younis Hijazi, Charles Hansen, Ingo Wald, and Hans Hagen. Interactive ray tracing of arbitrary implicits with simd interval arithmetic. In *RT '07: Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, pages 11–18, Washington, DC, USA, 2007. IEEE Computer Society.

[Kin04a] G Kindlmann. Superquadric tensor glyphs. In *Proceedings of IEEE TVCG/EG Symposium on Visualization 2004*, pages 147–154, May 2004.

[Kin04b] Gordon Kindlmann. *Visualization and analysis of diffusion tensor fields*. PhD thesis, University of Utah, 2004. Adviser-Johnson,, Christopher R.

[KVH84] James T. Kajiya and Brian P Von Herzen. Ray tracing volume densities. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 165–174, New York, NY, USA, 1984. ACM.

[KWH00] Gordon Kindlmann, David Weinstein, and David Hart. Strategies for direct volume rendering of diffusion tensor fields. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):124–138, 2000.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169, New York, NY, USA, 1987. ACM.

[LJH03] Robert S. Laramee, Bruno Jobard, and Helwig Hauser. Image space based visualization of unsteady flow on surfaces. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 18, Washington, DC, USA, 2003. IEEE Computer Society.

[LL94] Philippe Lacroute and Marc Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Com-*

puter graphics and interactive techniques, pages 451–458, New York, NY, USA, 1994. ACM.

[LOPR97] T. Lehmann, W. Oberschelp, E. Pelikan, and R. Repges. *Bildverarbeitung für die Medizin: Grundlagen, Modelle, Methoden, Anwendungen*. Springer, Heidelberg, Germany, 1997.

[LSJ96] Yarden Livnat, Han-Wei Shen, and Christopher R. Johnson. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, 1996.

[Nes09] Moriel NessAiver. Basic nuclear magnetic resonance. `http://www.simplyphysics.com/page2_1.html`, 2009.

[NH91] Gregory M. Nielson and Bernd Hamann. The asymptotic decider: resolving the ambiguity in marching cubes. In *VIS '91: Proceedings of the 2nd conference on Visualization '91*, pages 83–91, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.

[PB07] Bernhard Preim and Dirk Bartz. *Visualization in Medicine*. Elsevier, 2007.

[PCK04] Budirijanto Purnomo, Jonathan D. Cohen, and Subodh Kumar. Seamless texture atlases. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 65–74, New York, NY, USA, 2004. ACM.

[Smi04] A.V. Smirnov. Introduction to tensor calculus. published online: http://www.mae.wvu.edu/faculty/smirnov/tensors/tensors.pdf, Oct, Nov 2003-2004.

[Tuf83] Edward Rolf Tufte. *The Visual Display of Quantitative Information*. Graphics Press USA, 1983.

[Tuf90] Edward Rolf Tufte. *Envisioning Information*. Graphics Press, 1990.

[Tur52] Alan Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London*, 237(641):37 – 72, 1952.

[Tur91] Greg Turk. Generating textures on arbitrary surfaces using reaction-diffusion. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 289–298, New York, NY, USA, 1991. ACM.

[vW02] Jarke J. van Wijk. Image based flow visualization. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 745–754, New York, NY, USA, 2002. ACM.

[vW03] Jarke J. van Wijk. Image based flow visualization for curved surfaces. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 17, Washington, DC, USA, 2003. IEEE Computer Society.

[WE04] Daniel Weiskopf and Thomas Ertl. A hybrid physical/device-space approach for spatio-temporally coherent interactive texture advection on curved surfaces. In *GI '04: Proceedings of Graphics Interface 2004*, pages 263–270, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004. Canadian Human-Computer Communications Society.

[Wes91] Lee Alan Westover. *Splatting: a parallel, feed-forward volume rendering algorithm*. PhD thesis, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 1991.

[WH06] Joachim Weickert and Hans Hagen. *Visualization and Processing of Tensor Fields*. Springer, 2006.

[WPG+97] Carl-Fredrik Westin, Sariel Peled, Hakon Gudbjartsson, Ron Kikinis, and Ferenc A. Jolesz. Geometrical diffusion measures for MRI from tensor basis analysis. In *ISMRM '97*, page 1742, Vancouver Canada, April 1997.

[WVG92] Jane Wilhelms and Allen Van Gelder. Octrees for faster isosurface generation. *ACM Trans. Graph.*, 11(3):201–227, 1992.

[ZB02] L. Zhukov and A. H. Barr. Oriented tensor reconstruction: Tracing neural pathways from diffusion tensor mri. In *Proceedings of IEEE Visualization '02*, pages 387–394, Los Alamitos, CA, 2002. IEEE Computer Society.

[ZB03] Leonid Zhukov and Alan H. Barr. Heart-muscle fiber reconstruction from diffusion tensor mri. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 79, Washington, DC, USA, 2003. IEEE Computer Society.