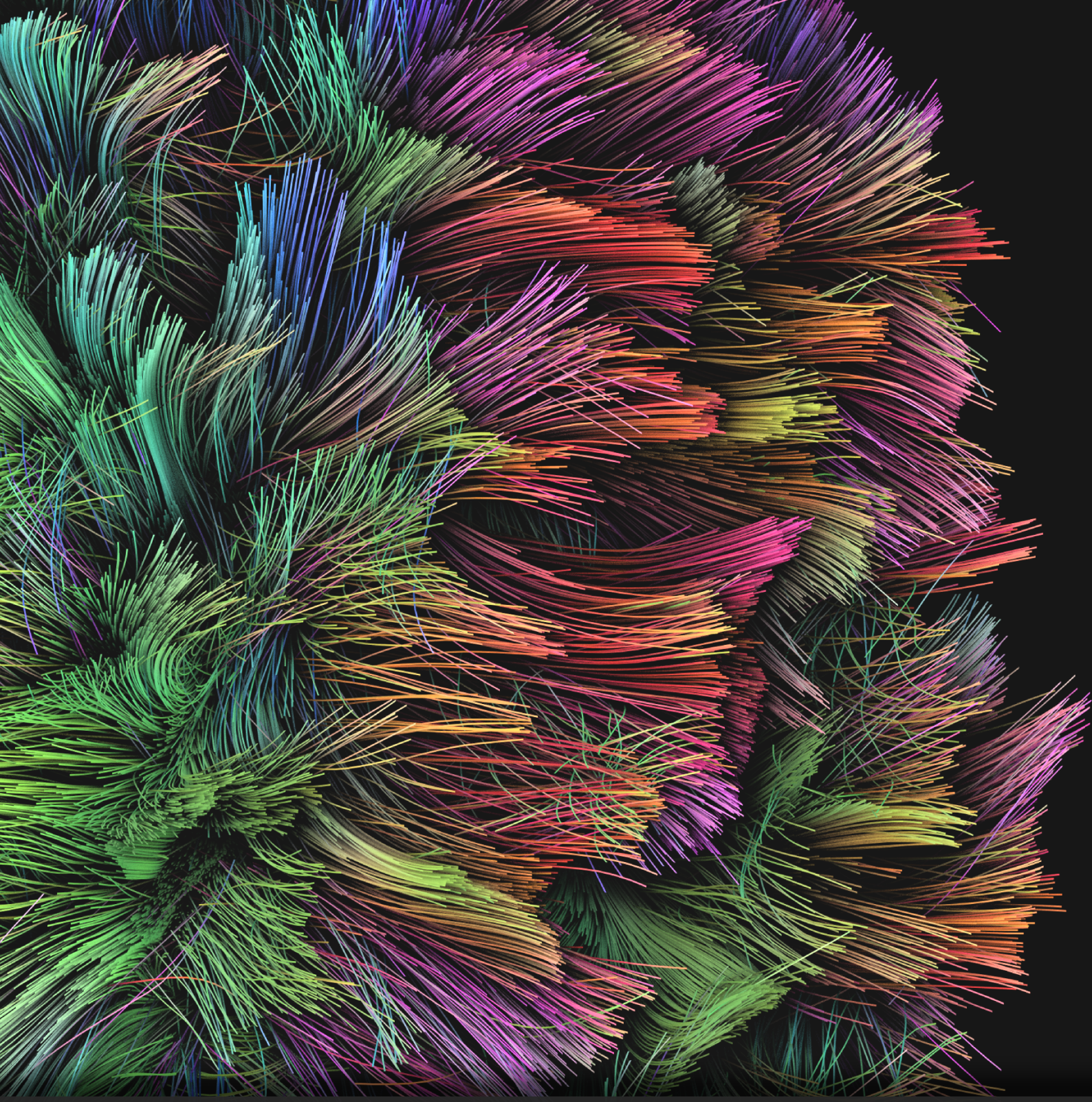


Sebastian Eichelbaum

Applied Visualization in the Neurosciences and the Enhancement of Visualization through Computer Graphics

Applied Visualization in the Neurosciences and the Enhancement of Visualization through Computer Graphics

Der Fakultät für Mathematik und Informatik
der Universität Leipzig
angenommene

D I S S E R T A T I O N

zur Erlangung des akademischen Grades

DOCTOR RERUM NATURALIUM
(Dr. rer. nat.)
im Fachgebiet
INFORMATIK

Vorgelegt

von Diplom-Informatiker Sebastian Eichelbaum

geboren am 3. Juli 1983 in Leipzig

Die Annahme der Dissertation wurde empfohlen von:

1. Juniorprofessor Dr. Mario Hlawitschka, Universität Leipzig
2. Professor Dr.-Ing. Bernhard Preim, Universität Magdeburg

Die Verleihung des akademischen Grades erfolgt mit Bestehen der Verteidigung am 27.11.2014 mit dem Gesamtprädikat summa cum laude.

Selbstständigkeitserklärung

Hiermit erkläre ich, die vorliegende Dissertation selbstständig und ohne unzulässige fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angeführten Quellen und Hilfsmittel benutzt und sämtliche Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, und alle Angaben, die auf mündlichen Auskünften beruhen, als solche kenntlich gemacht. Ebenfalls sind alle von anderen Personen bereitgestellten Materialen oder erbrachten Dienstleistungen als solche gekennzeichnet.

Leipzig, 4. Dezember 2014
.....
(Ort, Datum)

.....
(Unterschrift)

Lebenslauf

Persönliche Daten

Name	Sebastian Eichelbaum, Diplom Informatiker
Geburtsort	Leipzig
Geburtsdatum	3. Juli 1983
Anschrift	Gohliser Straße 20, 04105 Leipzig
E-Mail	kontakt@sebastian-eichelbaum.de

Schulbildung

08/1990 – 06/2002	Friedrich Schiller Gymnasium Leipzig
06/2002	Abitur, Leistungskurse: Mathematik und Physik

Wehrdienst

07/2002 – 04/2003	Grundwehrdienst
04/2002 – 07/2003	Freiwilliger, zusätzlicher Wehrdienst

Akademischer Werdegang

10/2003 – 07/2009	Studium der Informatik, Universität Leipzig. *Schwerpunkt*: Angewandte Informatik, Visualisierung und Bildverarbeitung im Speziellen
07/2009	Abschluss als Diplom Informatiker an der Universität Leipzig
09/2009 – 12/2014	Wissenschaftlicher Mitarbeiter, Abteilung Bild- und Signalverarbeitung, Universität Leipzig *Tätigkeiten*: • Betreuung studentischer Abschlussarbeiten • Durchführung von Seminaren • Projektleitung OpenWalnut • Forschungsarbeit zu Themen der Visualisierung in den Neurowissenschaften und der Angewandten Computer Grafik
11/2014	Verteidigung der Dissertation, Universität Leipzig Abschluss mit *summa cum laude*

Beruflicher Werdegang

01/2007 – 12/2008	Studentische Hilfskraft, Abteilung Bild- und Signalverarbeitung, Universität Leipzig
03/2008 – 08/2008	Praktikum, BMW Werk Leipzig
11/2012 – heute	Mitgründer und Gesellschafter der »Nemtics GbR«
07/2014 – heute	Gründung des Einzelunternehmens »Sebastian Eichelbaum IT Dienstleistungen«

Leipzig, 4. Dezember 2014

....

(Ort, Datum) (Unterschrift)

Danksagung

Ich möchte mich an dieser Stelle bei meinen beiden Betreuern Gerik Scheuermann und Mario Hlawitschka bedanken. Beide haben mich während meiner Promotion unterstützt und hatten immer ein offenes Ohr. Ich möchte mich für die unzähligen Antworten auf meine unzähligen Fragen und die vielen Ideen bedanken. Ein besonderer Dank gilt Mario und Alexander Wiebel dafür, dass sie mich während meiner Zeit als studentische Hilfskraft mit spannenden Aufgaben für die Visualisierung begeistert haben.

Ein großer Dank geht auch an die Abteilungen *Bild- und Signalverarbeitung* und *Wissenschaftliche Visualisierung*. Jeder hatte ein offenes Ohr für Fragen und gemeinsam haben wir so manches Problem angehen können. Besonders erwähnen möchte ich meine ehemaligen und aktuellen Bürokollegen Patrick Oesterling, André Reichenbach und Stefan Philips. Vielen Dank für die vielen spannenden Diskussionen und Zerstreuungsmomente.

Ein besonderer Dank geht auch an das OpenWalnut Team. Jeder hat aktiv beigetragen und so konnten wir eine hervorragende Software entwickeln, die längst über die Grenzen unserer Gruppe hinaus genutzt wird.

Ich möchte mich außerdem bei meinen Koautoren und Kollaborationspartnern bedanken. Wir haben viele Dinge ausprobiert. Nicht alles war ein Erfolg, aber aus vielen Ideen und Experimenten sind gute wissenschaftliche Publikationen hervorgegangen. In alphabetischer Folge: Alfred Anwander, Dana Brooks, Moritz Dannhauer, Bernd Hamann, Mario Hlawitschka, Jens Kasten, Thomas Knösche, Stefan Philips, André Reichenbach, Gerik Scheuermann, Ralph Schurade, Alexander Wiebel und natürlich alle anderen, die mit Ideen, Tipps, Codes und Daten geholfen haben.

Neben allen meinen Kollegen gebührt der größte Dank natürlich meiner Familie, meinen Freunden und meiner herzallerliebsten Bea. Ihr habt mich stets unterstützt, habt mich aufgebaut wenn ich niedergeschlagen war, habt mich motiviert, habt es verstanden wenn ich grummelig war. Danke!

Zusammenfassung

In vielen Bereichen der Wissenschaft nimmt die Größe und Komplexität von gemessenen und simulierten Daten zu. Die technische Entwicklung erlaubt das Erfassen immer kleinerer Strukturen und komplexerer Sachverhalte. Um solche Daten dem Menschen zugänglich zu machen, benötigt man effiziente und spezialisierte Visualisierungswerkzeuge. Nur die Anpassung der Visualisierung auf ein Anwendungsgebiet und dessen Anforderungen erlaubt maximale Effizienz und Nutzen für den Anwender.

Teil I Im ersten Teil meiner Arbeit befasse ich mich mit der Visualisierung im Bereich der Neurowissenschaften. Ihr Ziel ist es, das menschliche Gehirn zu begreifen; von seinen kleinsten Teilen bis hin zu seiner Gesamtstruktur. Um dieses ehrgeizige Ziel zu erreichen nutzt die Neurowissenschaft vor allem kombinierte, dreidimensionale Daten aus vielzähligen Quellen, wie MRT, CT oder funktionalem MRT. Um mit dieser Vielfalt umgehen zu können, benötigt man in der Neurowissenschaft vor allem spezialisierte und evaluierte Visualisierungsmethoden.

Zunächst stelle ich ein umfangreiches Softwareprojekt namens »OpenWalnut« vor. Es bildet die gemeinsame Basis für die Entwicklung und Nutzung von Visualisierungstechniken mit unseren neurowissenschaftlichen Kollaborationspartnern. Auf dieser Basis sind klassische und neu entwickelte Visualisierungen auch für Neurowissenschaftler zugänglich. Anschließend stelle ich ein spezialisiertes Visualisierungsverfahren vor, welches es ermöglicht, den kausalen Zusammenhang zwischen Gehirnarealen zu illustrieren. Das war vorher nur durch abstrakte Graphenmodelle möglich. Den ersten Teil der Arbeit schließe ich mit einer Evaluation verschiedener Standardmethoden unter dem Blickwinkel simulierter elektrischer Felder im Gehirn ab. Das Ziel dieser Evaluation war es, der neurowissenschaftlichen Gemeinde die Vor- und Nachteile bestimmter Techniken zu verdeutlichen und anhand klinisch relevanter Fälle zu erläutern.

Teil II Neben der eigentlichen Datenvorverarbeitung, welche in der Visualisierung eine enorme Rolle spielt, ist die grafische Darstellung essenziell für das Verständnis der Strukturen und Bestandteile in den Daten. Die grafische Repräsentation von Daten bildet die Schnittstelle zum Gehirn des Menschen. Der zweite Teile meiner Arbeit befasst sich mit der Verbesserung der strukturellen und räumlichen Wahrnehmung in Visualisierungsverfahren – mit der Verbesserung der Schnittstelle.

Leider werden viele visuelle Verbesserungen durch Computergrafikmethoden der Spieleindustrie mit Argwohn beäugt. Im zweiten Teil meiner Arbeit werde ich zeigen, dass solche Methoden in der Visualisierung angewendet werden können um den räumlichen Eindruck zu verbessern und Strukturen in den Daten hervorzuheben. Dazu nutze ich ein in der Computergrafik bekanntes Paradigma: das »Screen Space Rendering«. Dieses Paradigma hat den Vorteil, dass es auf nahezu jede existierende Visualiserungsmethode als Nachbearbeitunsgschritt angewendet werden kann.

Zunächst führe ich zwei Methoden ein, die die Wahrnehmung von gitterartigen Strukturen auf beliebigen Oberflächen verbessern. Diese Gitter repräsentieren die Struktur von Tensoren zweiter Ordnung und wurden durch eine Methode namens »TensorMesh« erzeugt. Anschließend zeige ich eine neuartige Technik für die optimale Schattierung von Linien und Punktdaten. Mit dieser Technik ist es erstmals möglich sowohl lokale Details als auch globale räumliche Zusammenhänge in dichten Linien- und Punktdaten zu erfassen.

Summary

The complexity and size of measured and simulated data in many fields of science is increasing constantly. The technical evolution allows for capturing smaller features and more complex structures in the data. To make this data accessible by the scientists, efficient and specialized visualization techniques are required. Maximum efficiency and value for the user can only be achieved by adapting visualization to the specific application area and the specific requirements of the scientific field.

Part I In the first part of my work, I address the visualization in the neurosciences. The neuroscience tries to understand the human brain; beginning at its smallest parts, up to its global infrastructure. To achieve this ambitious goal, the neuroscience uses a combination of three-dimensional data from a myriad of sources, like MRI, CT, or functional MRI. To handle this diversity of different data types and sources, the neuroscience need specialized and well evaluated visualization techniques.

As a start, I will introduce an extensive software called “OpenWalnut”. It forms the common base for developing and using visualization techniques with our neuroscientific collaborators. Using OpenWalnut, standard and novel visualization approaches are available to the neuroscientific researchers too. Afterwards, I am introducing a very specialized method to illustrate the causal relation of brain areas, which was, prior to that, only representable via abstract graph models. I will finalize the first part of my work with an evaluation of several standard visualization techniques in the context of simulated electrical fields in the brain. The goal of this evaluation was clarify the advantages and disadvantages of the used visualization techniques to the neuroscientific community. We exemplified these, using clinically relevant scenarios.

Part II Besides the data preprocessing, which plays a tremendous role in visualization, the final graphical representation of the data is essential to understand structure and features in the data. The graphical representation of data can be seen as the interface between the data and the human mind. The second part of my work is focused on the improvement of structural and spatial perception of visualization – the improvement of the interface.

Unfortunately, visual improvements using computer graphics methods of the computer game industry is often seen sceptically. In the second part, I will show that such methods can be applied to existing visualization techniques to improve spatiality and to emphasize structural details in the data. I will use a computer graphics paradigm called "screen space rendering". Its advantage, amongst others, is its seamless applicability to nearly every visualization technique.

I will start with two methods that improve the perception of mesh-like structures on arbitrary surfaces. Those mesh structures represent second-order tensors and are generated by a method named "TensorMesh". Afterwards I show a novel approach to optimally shade line and point data renderings. With this technique it is possible for the first time to emphasize local details and global, spatial relations in dense line and point data.

Contents

1 Overview

The complexity and size of data in many fields of science is increasing constantly. The technological evolution allows for capturing smaller features and more complex structures. To make this data accessible to the scientists, specialized visualization techniques are required, as efficiency and value for the user can only be achieved by adapting visualization to the specific application area and the specific requirements of the scientific field.

Part I In the first part of my work, I address the visualization in the neurosciences. Neuroscience is an incredibly complex interdisciplinary science, reaching from chemistry, over medicine to computer science and engineering. The shared goal of all involved disciplines is the final understanding of the brain's function, beginning at its smallest parts, the neurons up to its global infrastructure. To achieve this ambitious goal, the neuroscience uses a combination of data from a myriad of sources, like MRI, CT, functional MRI, EEG, and microscopy to mention only some examples. To handle and analyze the diversity of different data types and sources, the neuroscience needs specialized and well evaluated visualization techniques. The importance of proper evaluation of visualization methods cannot be ranked high enough.

During my research work, me and my colleagues intensively collaborated with neuroscientists, who were working in the areas of neuroimaging, cogni-

tive neuroscience, and computational neuroscience. Unfortunately, modern visualization is not actively used. Instead, colormaps on slices and statistical assessment are the de-facto standard for analysing complex simulations and measured data, even though there is a huge interest in visualization. We identified three major reasons for the restraint to use advanced visualization techniques:

1. Most visualization methods are not evaluated and leave too much questions unanswered: Do they show the information needed? Are they reproducible? How does the technique influence the data prior to displaying it? How comparable are the results? Can it be embedded into anatomical context?

2. A lot of visualization techniques are too complex and parameter-dependent.

3. Most published visualization methods are not accessible directly, in terms of software and applicability to a given kind of data.

In this work, I will introduce our contributions to alleviate this set of problems. As a start, I will introduce an extensive software called "OpenWalnut". It forms the common base for developing and using visualization techniques with our neuroscientific collaborators. Using OpenWalnut, standard and novel visualization approaches are available to the neuroscientific researchers too. Afterwards, I am introducing a very specialized method to illustrate the causal relation of brain areas, which was, prior to that, only representable via abstract graph models. I will finalize the first part of my work with an evaluation of several standard visualization techniques in the context of simulated electrical fields in the brain. The goal of this evaluation was to clarify the advantages and disadvantages of the used visualization techniques to the neuroscientific community. We exemplified these, using clinically relevant scenarios.

In part one, the necessary background information is given at the beginning of each chapter, due to their dissimilarity.

Part II The visualization pipeline is usually depicted by three major steps: filtering, mapping, and rendering. Besides the data processing steps, which play a tremendous role in visualization, the final graphical representation (rendering) of the data is essential for understanding structures and features in the data. The graphical representation of data can be seen as the interface between the data and the human mind. The second part of my work is focused

on the improvement of structural and spatial perception of visualization – the improvement of the interface.

Unfortunately, visual improvements using computer graphics methods, originally invented of and for the computer game industry, is often seen sceptically. In the second part, I will show that such methods can be applied to existing visualization techniques with ease to improve spatiality and to emphasize structural details in the data. So, advanced computer graphics in visualization is not only about beautiful pictures, but provides a tremendous benefit, when it comes to understanding the shown data and structures.

The second part will start with an introduction to the modern graphics pipeline and the screen space rendering paradigm, as this is the basis for the shown methods. Its advantage, amongst others, is its seamless applicability to nearly every visualization technique. The background chapter is directly followed by the first two methods that improve the perception of mesh-like structures on arbitrary surfaces. Those mesh structures represent second-order tensors and are generated by a method named "TensorMesh". Afterwards, I show a novel approach to optimally shade line and point data renderings. With this technique it is possible for the first time to emphasize local details and global, spatial relations in dense line and point data.

Each method chapter will begin with an overview on the used input data, how it is usually visualized, and why this can be improved. The chapters close with a critical examination of the method's weaknesses and how they can be solved.

Note Chapters 3 to 5 and 7 to 9 are based on publications I made during my doctoral research. Each publication, talk, and supplemental material is available online at `http://www.sebastian-eichelbaum.de/publications`.

2 Thesis Contributions

This chapter summarizes the contributions I achieved during my doctoral research. It lists the novel techniques, improvements and achievements in order of appearance.

Chapter 3 – OpenWalnut As conceptual program designer and programmer, I contributed major parts of the OpenWalnut core library and graphical user interface. I designed abstract, module-centric interfaces, making OpenWalnut a flexible, adaptable, and useful visualization framework. Originally designed as common base for neuroscientific visualization and research, it is now used by many groups in different fields of science and applications (not only neuroscience). Its open nature allows results to be reproduced and validated, which is tremendously important in science.

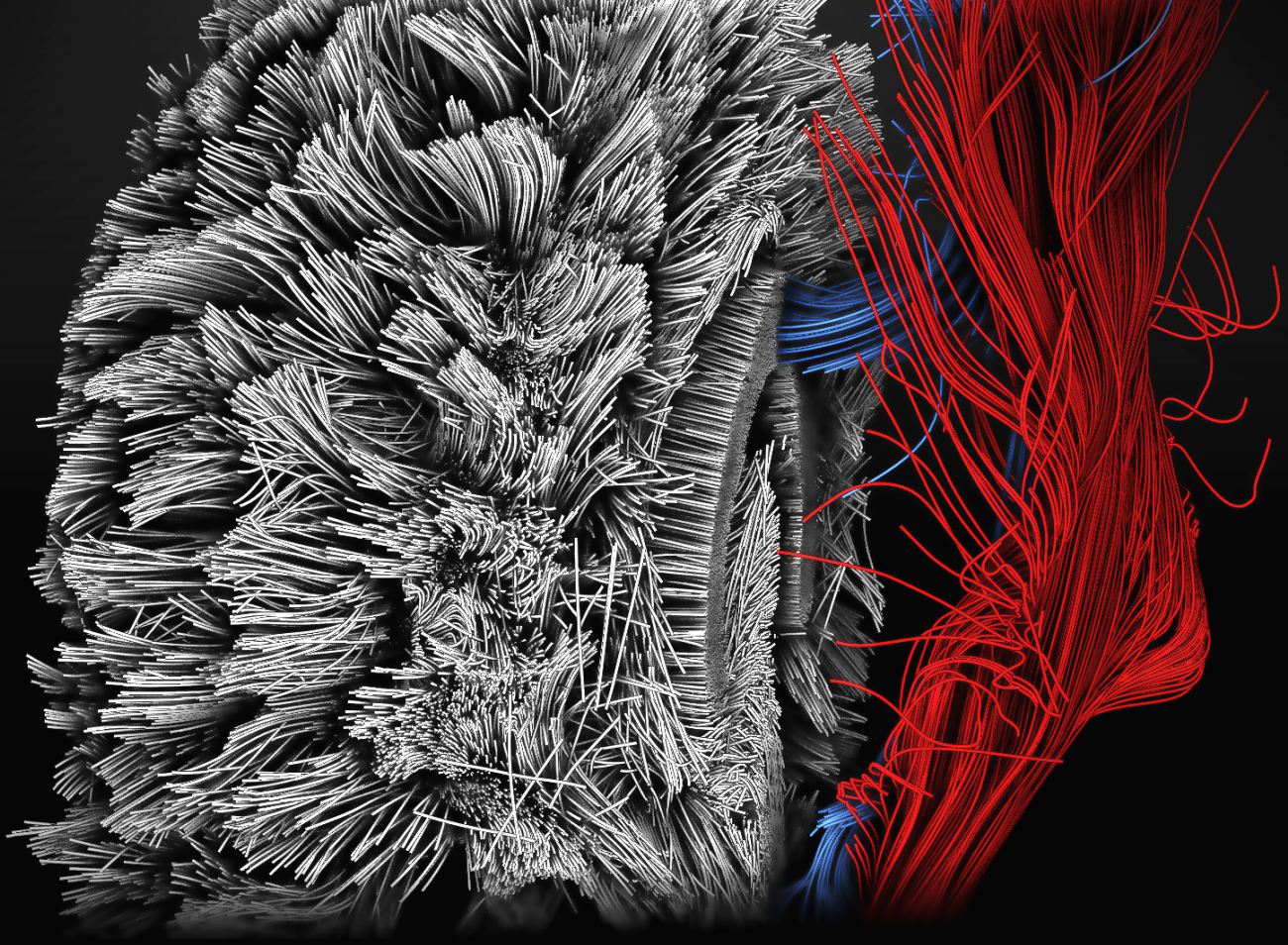
Chapter 4 – Effective Connectivity Visualization I have introduced a novel visualization for effective connectivity models. The method illustrates these models and embeds them into the anatomical context, which was not possible before. I have shown that the transfer of abstract DCM models to a meaningful, anatomy-based visualization is feasible and that it helps to perceive these abstract models from the anatomical perspective.

Chapter 5 – Evaluation of Standard Methods The major contribution in Part I of my dissertation. I evaluated standard visualization techniques (iso-surfaces, DVR, streamlines, LIC) in the context of simulated electrical fields for their usability, advantages, and limitations. I implemented and applied these methods on neuroscientific datasets, I got from one of my colleagues. I have pointed out the possibilities of visualization and the very specific features in his data. With the help of these visualizations, he was able to draw interesting neuroscientific conclusions. Such an evaluation was done the first time in the neuroscience community and we got a lot of feedback regarding visualization and OpenWalnut.

Part I – Visualization for the Neurosciences With OpenWalnut and the visualization evaluation, I helped to make visualization more accessible to the neuroscientists. I have shown that there are more possibilities than colormaps and statistics to analyse data. I have demonstrated several standard techniques in practice and made them available in a tool, the neuroscientists can adapt easily to match the specific needs of a certain analysis task.

Chapter 7 – Improved Perception of Structure I have extended previous work for improved perception of information on surfaces, represented as mesh-like structure. The methods shown are relatively easy and can be adopted in other visualization techniques that display information on surfaces easily. I used the knowledge of this chapter in the evaluation (Chapter 5) and got further positive feedback on the improvement of several surface-based LIC renderings. This underlined the advantages of advanced computer graphics in visualization.

Chapters 8 and 9 – Ambient Occlusion for Lines and Points These chapters show my major contribution to the area of applied computer graphics in visualization. I have developed a novel technique to enhance local structures and global spatial relations in dense line and point data. Especially for line data, this was not possible before and I showed that screen space postprocessing is a valuable tool to improve existing rendering techniques. The proposed method completely works in real-time and is not dependent on any precalculations. It can be applied to arbitrary line and point based visualization techniques.



Visualization in the Neurosciences

Visualization plays a central role in many areas of science, as it helps to grasp the nature of measured and simulated data. Also neuroscience is very interested in visualization. It allows for an effective conveyance of complex data and enables quick qualitative and quantitative assessments. Visualization can unveil structures and properties inside the data that statistical measures cannot.

However, not every visualization technique is equally adequate for different analysis tasks and types of data. Additionally, the vast amount of available techniques makes it hard to decide for an optimal visualization approach. Rapidly evolving, new measurement techniques and simulation models demand for more and more specialized, application-specific visualizations. Practicability studies on exiting visualization techniques are rare, causing a huge uncertainty among neuroscientific researchers about the expressiveness and validity of available methods; not to mention their availability in software tools, if at all.

This part will present our work and contribution to the neurosciences. It will introduce our open and free software framework, where we and other visualization researchers implement methods to make them easily available to neuroscientific users. We present a specific visualization technique to a trending area of neuroscience, helping to understand the information flow in the brain. Finally, we evaluate the usability of several standard visualization techniques for the analysis of electrical fields in the human brain.

Part I

3

OpenWalnut – Open-Source Visualization for Medical and Neuroscientific Data

This chapter is based on the following publications:

[P1] – S. EICHELBAUM, M. GOLDAU, S. PHILIPS, A. REICHENBACH, R. SCHURADE, and A. WIEBEL. **OpenWalnut: A New Tool for Multi-modal Visualization of the Human Brain**. *EG VCBM 2010 Posters*. 2010
Online: http://sebastian-eichelbaum.de/pub10d

[P2] – S. EICHELBAUM, M. HLAWITSCHKA, A. WIEBEL, and G. SCHEUERMANN. **OpenWalnut - An Open-Source Visualization System**. *Proceedings of the 6th High-End Visualization Workshop*. Ed. by W. Benger, A. Gerndt, S. Su, W. Schoor, M. Koppitz, W. Kapferer, et al. 2010, 67–78
Online: http://sebastian-eichelbaum.de/pub10e

[P3] – S. EICHELBAUM, M. HLAWITSCHKA, and G. SCHEUERMANN. **OpenWalnut: An Open-Source Tool for Visualization of Medical and Bio-Signal Data**. *Biomedical Engineering / Biomedizinische Technik*. Ed. by O. Dössel. 2013
Online: http://sebastian-eichelbaum.de/pub13c

3.1 Overview

In the course of ongoing research into neurological diseases and the function and anatomy of the brain, a large variety of examination techniques has evolved. The different techniques aim at findings for different research questions or different viewpoints of a single task. The following are only a few of the very common measurement modalities and parts of their application area:

- *Computed Tomography (CT)* – anatomical information, using X-ray measurements;
- *Magnetic Resonance Imaging (MRI)* – anatomical information, using magnetic resonance especially for soft tissues;
- *Diffusion Weighted MRI (dwMRI)* – directed anatomical information for extraction of fiber approximations;
- *Functional MRI* (fMRI) – activity of brain areas indicated by the blood-oxygen-level dependence (BOLD) effect; and
- *Electroencephalography (EEG)* – activation of certain brain areas, indicated by electric fields.

Considering the different applications, it is evident that, for many research areas, only a combination of these techniques can help answering the posed questions. To be able to analyze data measured by the different techniques, a tool that can efficiently visualize different modalities simultaneously is needed. It has to provide algorithms to analyze their relations and differences.

As we started an intensive collaboration with neuroscientists at the Max Planck institute for human cognitive and brain sciences, we required a common basis for developing algorithms and implementing ideas. Our colleagues at the Max Planck Institute already had a tool called "FiberNavigator" [54], which turned out to be not flexible enough at this point. So we started developing a new visualization framework, initially tailored towards the common requirements of visualization researchers and neuroscientists, *using* our visualization tools.

The reasons why we did not use one of the many existing systems will be the topic of the next section. The remaining chapter reveals some of the architectural details and shows how we fulfil the posed requirements.

3.2 Focus and Reasoning

As mentioned above, it is very crucial to handle a multitude of different kinds of images and signals. Besides this, there are several other criteria for visualization software, especially in a scientific environment.

There are many visualization tools available, which have their specific strengths and weaknesses, but none of them was able to completely fulfill our criteria. In this section, we point out these criteria, building the fundamentals of OpenWalnut's software design and implementation.

At the same time, we use these criteria as a filter on a list of well known and excellent visualization tools. In alphabetical order, these are: Amira [5], MayaVi [122], MedINRIA [125], MeVisLab [129], ParaView [1], and the problem solving environment SCIRun [181].

Open-Source: In a scientific environment, it is very important to be able to reproduce results of other researchers and to comprehend their algorithms and methods. With the help of open-source software, this can be achieved. It provides a possibility to share algorithms and calculation pipelines with others in a common framework, without hiding necessary implementation details. One could argue that it is sufficient to have the source code of the algorithm/method itself and that the underlying framework is not important. This is only true, if the method is completely self-contained, which is not the case usually. Most published methods today reuse existing algorithms and calculation methods, or even extend them.

Besides this, an open-source framework allows for easy extension and adaption of existing methods to new problems or different data modalities.

Not fulfilling: Amira [5] and MeVisLab [129]. Both systems are closed source software. MeVisLab offers open-source modules, but the framework itself is closed.

General Purpose: Very focused software is not able to handle the above mentioned multitude of signal and image modalities. Additionally, in a research environment, it is very often required to find new ways of solving a certain medical or neurological problem. To achieve this, the used software must not limit the researcher in terms of applicability of algorithms and in terms of easy programmatic extensibility.

Not fulfilling: MedINRIA [125]. It is a tool, focused on exploring and processing medical images. It is tailored towards an image processing workflow, and heavily relies on slice based image exploration.

Extensible: As we aimed at a software, which can be used for common research in visualization and neurosciences, a fixed function tool suite has no use for us. We required a software, which allows to add new algorithms and whole processing pipelines with ease. Additionally, it is was required to reuse existing tools. Re-inventing the wheel is a waste of time and our neuroscientific collaborators already had complex processing pipelines written in Python and MatLab, hence the coupling of external libraries and tools is a critical point.

As our list of tools contains only open-source tools at this point, they all can be seen as being extensible. However, making the source code available is insufficient. To use an existing framework and extending it with long-term usability in mind, it is required to have plenty of documentation and an active developer community. Further development of badly documented software, with no support by the original developers, is cumbersome and not feasible in practice.

A note on ParaView [1]: Although ParaView itself is extremely well documented and very extensible, the reliance on VTK [209] posed an obstacle to us, as we planned to harness the processing power of the GPU. Especially the modification of the rendering pipeline (i.e. for multi-pass rendering) and the use of OpenCL was not explicitly included in the design of VTK in 2009. VTK 5.6, released in September 2010, started to include multi-pass rendering functionality with `vtkRenderPassCollection`; too late for us. However, filtering ParaView because of this limitation does not come up to its extensibility.

Not fulfilling: MayaVi [122]. In 2009, MayaVi was practically dead. The development of MayaVi 2, the Python and VTK based successor, was not sufficiently advanced.

Usability – Graphical User Interface (GUI): Most software aims either at the visualization researcher or the neuroscientist, with accordingly designed GUIs. As we aimed at both groups – the neuroscientists, who needs a usable tool to handle and visualize their data and the visualization researchers, who need a powerful, programmable tool with a flexible user interface. Especially the usability of the GUI was an important criterion, excluding a lot of pre-existing tools.

Not fulfilling: ParaView [1] and SCIRun [181]. Both GUIs are very complex and the learning curve is steep. Additionally, in ParaView, the processing pipeline is pretty much kept behind the scenes. This is a disadvantage for the visualization scientist. SCIRun's strength certainly is its powerful data processing capability. It provides visualization tools though, but they are not that sophisticated.

Availability and Accessibility: The hurdle of using software which is not available on the user's operating system, or which needs to be compiled tediously for a system is very high. We required a portable software, which works on different platforms and is available as binary package. Nowadays, this criterion is fulfilled by most software tools and libraries.

Amongst the pure source and binary distribution, we also consider documentation, support, and clean code to belong to *accessible* software. Thereby, we not only refer to documentation and support for users, but also for developers. Unfortunately, a lot of open-source visualization tools fall short at this particular point. Not seldom, open-source releases contain mostly prototypic, badly designed code and lag basic developer and user documentation.

Summary: Surely, there are a lot of visualization tools available we not mentioned explicitly. And surely, a lot of them fulfill the above requirements and would have been a perfect tool for our purpose. Unfortunately, we were not aware of them – or they simply did not comply with our requirements in code quality and documentation.

This was the reason for starting the OpenWalnut project in 2009. We set the above criteria as our objective and started promoting OpenWalnut in the neuroscience community – the positive and negative feedback we got and still get, is the basis to develop a solid and community-driven visualization tool.

3.3 Realization

In the previous section, we defined the objectives for the development of OpenWalnut. During the realization of OpenWalnut, I designed and implemented the OpenWalnut core library and the abstract concepts in it. These concepts are the topic of this section. Additionally, I took the role of the "Benevolent dictator for life (BDFL)" [86], a very well known concept in the open-source

community. I am responsible for the positive advance of the project and I retain the final word in case of irresolvable discussions.

This section provides an overview on the architecture of OpenWalnut and how it complies to the above criteria.

3.3.1 Architecture Overview

OpenWalnut's design was mainly steered by the above criteria, but the need to build a software, used by researchers of two different fields of science, with different requirements, and different expectations, influenced development in two additional ways:

1. OpenWalnut has to be a powerful and easily expandable framework for visualization researchers, allowing them to implement algorithm prototypes and ideas quickly and easily while,
2. providing an intuitive graphical user interface for neuroscientist researchers, who include OpenWalnut in their daily research tasks.

Whereas the first point asks for a flexible and extendible framework, the second introduced the need for a high level of interactivity and responsiveness of the application.

To achieve these ambitious goals, it is important to split functionality and interface. Known and famous in the context of object-oriented programming, this principle (refer to the excellent work of Gamma et al. [60]) allows a powerful and complex framework under the hood of a simple interface, the GUI in our case.

Core Library OpenWalnut is completely centered around the module-principle. Everything in OpenWalnut is a separate module: each algorithm, each dataset, each visualization method. So, instead of implementing a fixed amount of algorithms into a library, we decided to implement the common utility code, required to write algorithms, visualizations, and datasets. We call this the "OpenWalnut Core Library (OCL)". The OCL contains math functionality, general data handling, a graphics engine, and the module interface. The graphics engine is as an extension to the OpenSceneGraph library [144], implementing several simplifications and wrappers on top of it. Worth mentioning is our sophisticated multi-pass rendering code, allowing for easy and efficient implementation of custom GPU rendering pipelines.

Modules Instead of mentioning all the details about utility codes in the OCL, we go on with the module concept in OpenWalnut. The OCL provides the so-called "module container". It implements the concept of the data flow graph. This is a very common concept in visualization and other data processing tools. For details on different visualization pipeline approaches, we refer to the work of Moreland [133]. In the context of visualization, the data flow concept defines data sources (sources), algorithms (filters), and visualizations (sinks) as nodes in a graph. Edges between the nodes represent the transport of data between them. We use a similar notion to the ones used in [133], and, according to this survey, we use distributed execution control with pipeline parallelism.

In OpenWalnut, the nodes are called "modules" and the module container accommodates them and their connections (edges). The module container provides a programming interface to add modules, remove modules, list modules, list connections, and connect outputs with inputs. The user interface (UI) can utilize this to depict the state of the module graph and to modify it, according to the user and script input. In OpenWalnut, the UI can be a graphical user interface (GUI), a command-line interface, or a script interface.

Additionally, an important characteristic of modules in OpenWalnut is that modules have a local view only and run in their own thread. They do not know anything about the module graph, other modules, or to whom they are connected. This avoids side-effects in the visualization pipeline. The fact that modules run in their own thread yields a high level of parallel execution in complex networks. The survey [133] calls this "pipeline parallelism".

Modules: Connectors A module has exactly one possibility to interact with other modules residing in the container. Modules can define so called *connectors*. These connectors represent the input and output channels of a module and define the exact type of data this connector supports. This ensures that modules always get the right kind of data at the correct input. A change in an input causes the module to be notified. When a module changes its own outputs, the changes automatically propagate along the graph and wake up directly depending modules, allowing them to process the new data. There is no central execution control. When an update on an input occurs, the module wakes up and handles it. In [133], this is called distributed execution control.

The whole architecture is designed to use the push mechanism to propagate data, states, and other information. The OCL makes heavy use of this and provides callback and signaling mechanisms for nearly all possible operations.

This ensures that the module container and OCL does not need to be polled somehow by the UI or by the modules.

Modules: Properties The remaining task is the communication of module parameters and other settings to the user/script.

During the design process, we always avoided that modules have to know the UI or need to specify their UI representation directly. Therefore, modules are equipped with a mechanism called *properties*.

These properties enable the module developer to simply define parameters or settings without any knowledge of their representation. A module can, for example, define a property of type `double` with a name and a description associated with it. In addition, the developer can define constraints, for example that it only accepts positive values. The interesting part here is, that it is up to the specific UI to decide about the (graphical) representation of those parameters. To stick with the double-property example, the UI can decide whether a slider or a text box is more appropriate for a property. It mainly uses the constraints defined on a property to find a proper representation.

Whenever the user modifies a property in the UI or script, the property automatically checks, whether the new value is valid, by using the beforementioned property constraints. If the value is invalid, the property rejects it and the UI can somehow show it to the user. If the value is valid, it is set for the property and the automatic change-propagation ensures that all observers, especially the module owning the property, are notified about the value change. A module can then wake up from its sleep state to handle the new value. As the properties implement the *observable pattern* [60], they can be used in a variety of ways in OpenWalnut.

Modules: Containers are Modules With an increasing amount of fine-grained algorithm implementations in modules, the complexity of the needed UI interaction increases tremendously. This is especially true, if results of algorithms need to be reused as input for other modules quite often.

To circumvent the problem, OpenWalnut allows using module containers as modules. As mentioned above, these containers accommodate multiple modules and their connections. A container can then forward outputs and inputs from modules inside to the outside world. This allows a module container to look and behave like a normal module. The module programmer can recombine modules in a certain way to map a work-flow or visualization use case, without revealing the underlying complexity. This, on the one hand,

hides complexity from the user and makes the software more intuitive. On the other hand, it allows programmers to reuse existing modules and algorithm networks with ease in their own modules.

UI and GUI The UI represents the front-end to the core library and its data flow graph. The abstract definition of modules, their inputs and outputs, as well as their abstract property description allows the UI to create a clean and structured interface automatically. This way, the UI looks the same for all modules and directly reflects the flow graph and the OCL concept. This being said, one can see Figure 3.1 as a schematic view on the OpenWalnut core: a container populated with modules, the settings of the modules, and the graphics engine output.

Before we used the flow graph to represent the processing pipeline, we have used a linear list of data and applied modules on them. This style of pipeline representation is fairly standard in many medical image processing tools, hence our neuroscientist colleagues requested this scheme too, as they were used to it already. Initially designed to hide the complexity of the flow graph, it turned out to be more confusing than simplifying. So, we changed this to a direct flow graph representation, which is far more intuitive. It directly represents the flow of data along a processing pipeline.

Architecture Summary To summarize the above paragraphs, one can say that the core of OpenWalnut is the OCL – a library, providing an interface to a data flow graph. The modules and their connectors are used to populate the module container in the OCL, building the data flow graph. Parameters and settings of modules are defined using the abstract properties interface, allowing the module developer to describe the required value, without providing a specific implementation. The pipeline parallelism and the intensive use of callback mechanism ensures responsiveness of the UI/GUI at all times and the abstract module interfaces allow for a straight-lined and structured user interface.

Using the OCL, we implemented a graphical user interface and a Python script interface as UI. We also deliver a lot of modules to provide the user and other developers with a set of sophisticated visualization and processing techniques. The OCL, the two UIs, and the extensive set of modules builds up the package, we call OpenWalnut.

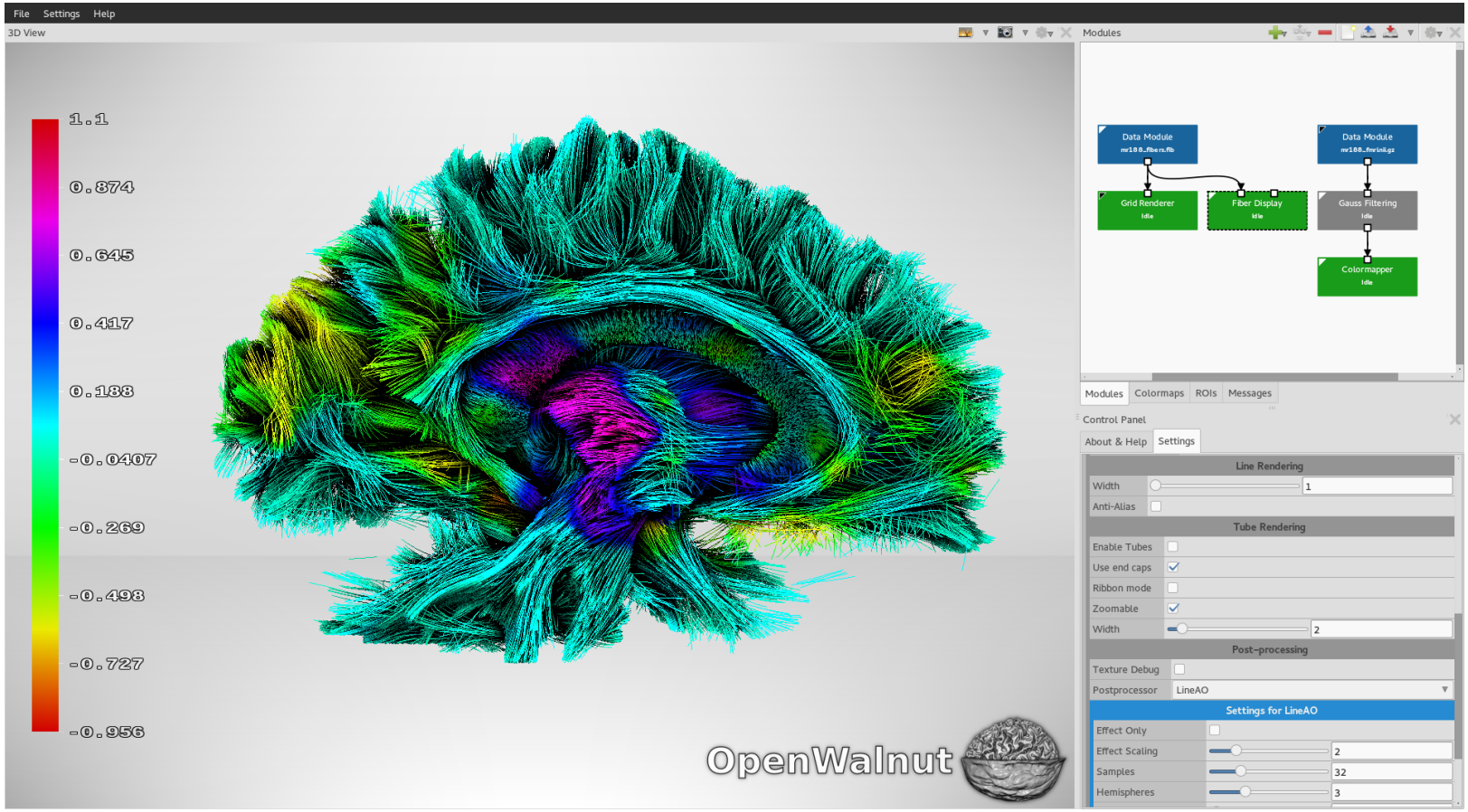


Figure 3.1: *The GUI of OpenWalnut. On the top-right of the window, the data flow graph is shown. It directly reflects the modules and their connections. The lower-right part shows the module properties of the currently selected module ("Fiber Display" in this case). The properties can be grouped by the module developer and an additional tab provides help on the module. The remaining part of the window is used for rendering the scene. Additionally, all operations that apply to a certain sub-window, are placed at the top of that sub-window, instead of a global tool-bar.*

3.4 Results

Now, that the notions and the basic concepts in OpenWalnut are known, we focus on how OpenWalnut blends into the before-mentioned criteria and what we have achieved so far.

Open-Source, Availability and Accessibility OpenWalnut is licenced under the terms of the lesser GNU general public license 3, LGPLv3 for short. The source is distributed at our website. This makes OpenWalnut open-source.

The complete source is written in portable C++, using only portable external libraries, like Qt4 for the GUI and OpenSceneGraph for the graphics output. The advantage is that all required libraries are available on Linux, Windows and Mac OS, making OpenWalnut available on all major platforms. We deliver a binary distribution in the NeuroDebian repository [69], providing even easier access to the neuroscience community.

Last but not least, we provide extensive documentation and support for developers and users. We deliver online documentation, extensive programming tutorials, proper code documentation, a beginners tutorial, open issue tracking, and mail support. This way, we are lowering the first hurdle for users and developers tremendously. Our undergraduate students work and develop with OpenWalnut in less than a week, without help, just by using the provided documentation. The high quality of our code, ensured by strict code guidelines, and the high quality documentation guarantee the accessibility to OpenWalnut and visualization.

The documentation, the sources and binary distributions are available online at `http://www.openwalnut.org`.

General Purpose and Extensible OpenWalnut uses the data flow principle. It is extremely flexible and does not fix the user to a certain image processing pipeline. It provides a maximum of flexibility for neuroscientists, by allowing them to combine data and algorithms in nearly any way.

Moreover, the easy-to-use programming interface allows developers to provide new modules without any hassle. It is easy to utilize the strengths of external libraries, or to completely develop new methods from scratch. OpenWalnut does not entail any unnecessary limitations to modules.

Usability – Graphical User Interface (GUI) OpenWalnut focuses on a clean and straight interface, which is centered around the data-flow network, as

shown in Figure 3.1. The GUI does not clutter the interface with additional setting-windows, several tool-bars, or awkwardly placed buttons and GUI elements. As the GUI is created automatically, we can ensure a consistent look and feel at all times. This lowers the entry curve tremendously and ensures that an user can get used to the software very fast.

Besides this, OpenWalnut tries to avoid global tool-bars and menus. All operations are placed at the sub-window to which they relate. For example, the "Load"-button to load a flow graph from a project file is not placed at a global tool-bar. It is placed at the top area of the *flow graph window*, because it modifies the *flow graph*. The "Screenshot"-button is at the top of the rendering window, because it saves the contents of the rendering window to disk. This is consistently done for all operations.

We avoid complex and complicated GUI dialogs for algorithms, provide useful default values for all parameters, direct visual feedback for parameter changes, and allow to combine complex data pipelines into containers, to hide their complexity. This and the structured GUI ensures a high level of usability for visualization researchers and the scientific user of OpenWalnut.

Besides the GUI, we provide a Python script UI. We are working with developers from SCIRun to integrate OpenWalnut with SCIRun via Python. This allows for a more direct coupling of visualization with a data processing pipeline.

3.4.1 Limitations

OpenWalnut is not perfect. It falls short for all the cases where one needs a criterion fulfilled, which contradicts one of the above. For example, if a very specific GUI is needed, our GUI implementation is of no use. Fortunately, the OpenWalnut core library makes it easy to develop new user interfaces from scratch.

Focus on Regular Grid Data Also noteworthy is the fact that OpenWalnut completely focuses on data, organized in regular grids. Other types of grids and completely new dataset types can be added easily in a collection of modules, since the OCL does not limit the developer in what he transfers through module connectors. This means, the OpenWalnut code does not need to be touched to implement new types of data or grids.

Synchronization As the data flow graph uses a decentralized execution control (cf. Moreland [133]), it can get problematic to synchronize parallel and repeated execution of algorithms. Consider the following scenario: there is a scalar field generator. It outputs the field to a spatial derivation module and to a module showing an isosurface. The spatial derivation module outputs a gradient vector field to the isosurface module, but takes some time to compute. The isosurface module uses the gradients as normals for lighting. So, the isosurface module depends on two inputs, whose data might arrive at different points in time. This is a synchronization problem in the data flow graph, which is not yet solved in OpenWalnut. I already started thinking about *barriers* in the data flow graph for synchronization. The barrier-concept is widely used in parallel programming and might solve our synchronization problem too.

Mac OS X Support Another, yet pressing issue is our Mac OS X support. Although OpenWalnut compiles and works on this operating system, it can be cumbersome sometimes. We rely on a lot of external libraries (Boost, Qt, and OpenSceneGraph). If, due to a change made by Apple, one of these libraries does not work anymore, OpenWalnut does not work or compile anymore. This happens to be the case for the OpenSceneGraph library quite a lot recently. Especially the combination of OpenSceneGraph with the GUI programming system Qt is problematic on Mac OS X.

3.5 Future Work and Conclusion

3.5.1 Future Work

OpenWalnut is far from being done. We know the weaknesses and we are working on them. We permanently refine the system to reach even more users and researchers. The user documentation will be extended to contain video tutorials. The module-specific help has to be written for a lot of modules, and we tackle the limitations mentioned in the Section 3.4.1. In other words, there are a lot of things to do and we are facing this challenge.

3.5.2 Conclusion

In the last sections, I introduced OpenWalnut, how it works in principle, and how it complies to our posed criteria. Of course, it is hard to scientifically substantiate the above arguments. This is why we explicitly point out that

there are many tools available for handling and visualizing medical images and bio-signal data. Each tool has its advantages and disadvantages. We do not try to re-invent the wheel, nor do we claim our project is superior to the others. We created a tool, which complies to a certain set of criteria and tries to reuse as many as possible of existing tools and frameworks. The ***extensive documentation, the flexible, but easy-to-use programming interfaces***, as well as our long term planning makes OpenWalnut ***attractive to many researchers and developers*** in the medical and neuroscientific fields. We provide them with an open and ***powerful tool*** to explore their data – or to create new ways of visualizing them. It is the tool we use at our department to develop ***high quality visualizations*** and to share our ideas with our collaborators and the scientific community.

The ***increasing number of users*** from different groups all around the world and the positive feedback we get, fortifies the claim that OpenWalnut is a solid visualization tool with a mentionable benefit for the scientific community, not only in the neurosciences.

4

Visualizing Effective Connectivity of the Brain

This chapter is based on the following publication:

[P4] – S. EICHELBAUM, A. WIEBEL, M. HLAWITSCHKA, A. ANWANDER, T. KNÖSCHE, and G. SCHEUERMANN. **Visualization of Effective Connectivity of the Brain**. *Proceedings of the 15th International Workshop on Vision, Modeling and Visualization (VMV) Workshop 2010*. Ed. by R. Koch, A. Kolb, and C. Rezk-Salama. 2010, 155–162
Online: http://sebastian-eichelbaum.de/pub10c

4.1 Overview

The Data: Connectivity Models Diffusion-weighted magnetic resonance imaging (DW-MRI) has become a window to the anatomical structures of the human brain and allows *in-vivo* reconstruction of fiber tracts that form neural networks. Although the size of single nerve fibers is far below the resolution capabilities of today's imaging devices, neuroscientists use tracked fiber clusters intensively to understand the human brain's structure, in particular its *connectome*, i.e., the wiring scheme of the brain — ***the structure***.

On the other hand, electroencephalography (EEG), magnetoencephalography (MEG) and functional MRI (fMRI) allow scientists to measure functional coherences between the activation in different brain areas in response to external stimuli — ***the function***.

A major goal in neuroscience is the understanding of structure-function relationships. To combine both, the anatomical knowledge and the experimental results in models, representing the influences of structural connections in an experimental context, many approaches have been developed. One of these models is *Dynamic Causal Modeling* (DCM). DCM was introduced by Friston et al. [58] and can be seen as a generalization of structural equation modelling (SEM) by McIntosh and Gonzalez-Lima [124]. For more details on DCM and its relation to SEM, please refer to Penny et al. [150]. The basic idea is to find a reasonable model that represents interacting cortical regions. DCM aims at making estimations about the causal architecture of coupled brain regions and, even more interesting, how this coupling is influenced by experimentally induced stimuli. In 2009, Stephan et al. [195] introduced an approach to include tractography-based, anatomical knowledge into DCM and have provided the first formal evidence that anatomical knowledge can improve DCM.

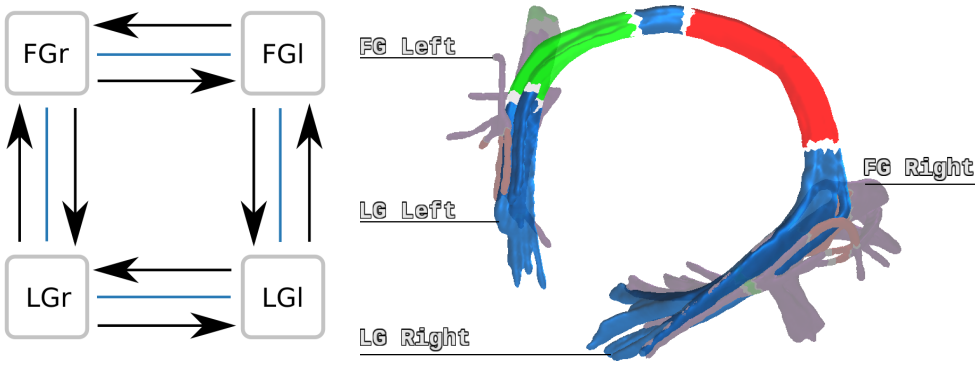
To summarize, DCM allows to describe *effective connectivity* as the combination of anatomical structure and functional causality, including intensity of *information transfer* between cortical regions. As a result, effective connectivity can be calculated between two cortical regions and is a measure for their causal relation. For each pair of anatomically connected regions, two effective connectivity values exist, one in each direction along the same anatomical path. This means that effective connectivity is a directed value, describing the information transfer from A to B and from B to A on the same anatomical path connecting A and B.

Visualization of Connectivity Visualization of medical data is a wide-spread field and many approaches have been developed to visualize almost all imaging and measurement modalities separately or in conjunction with each other. In the context of our work, methods regarding several kinds of connectivity are of special interest. Fiber tractography has been introduced to provide a global view on locally acquired data [9]. To interactively explore the white matter pathways, it has proven advantageous to precalculate a large number of fiber tracts in advance and selectively filter them, by using regularly shaped regions of interest, as done by Akers et al. [2] and Blaas et al. [17]. Another approach is to create large-scale structural brain networks, describing anatomical connection between several cortical regions of the brain [68]. These networks can be visualized by graphs and even by embedding them into the three-dimensional context of the brain, where they can be explored interactively. One well known exploration tool is the ConnectomeViewer [61]. For further details with regard to visualization and a comprehensive overview on this topic, please refer to chapter 15 of *Visual Computing for Medicine, Second Edition: Theory, Algorithms, and Applications* by Preim and Botha [153].

Functional connectivity is another challenge. The literature on useful visualizations for this kind of connectivity is relatively sparse. The vast amount of data requires statistical methods to find significant relations, which can be understood best, by providing an underlying anatomical context and interactive tools to selectively view the information provided. An example for exploring functional relationships is the BrainMiner tool as shown by Mueller et al. [135] and Welsh et al. [219].

Visualization of effective connectivity has not yet been investigated in detail. Usually, effective connectivity is shown as an abstract graph, without any anatomical meaning. Each node in the graph represents a certain area of the brain and is connected to other nodes with directed, weighted edges, denoting the effective connectivity.

The Problem Two-dimensional graph layouts do not allow the inclusion of anatomically guided geometric relationships and are, therefore, not able to show the structure-function relationship properly. Additionally, the visualization of two values in opposing directions on the same anatomical connection is a serious problem for common visualization methods. Prior to our method, there was no way to reconstruct the anatomical embedding of a DCM model and to visualize it.



(a) Abstract model (b) Our anatomical visualization

Figure 4.1: *Our goal is to convert the abstract graph model in (a) to the anatomically meaningful visualization in (b). (b) provides insight into the underlying anatomical structures, helping to understand the formerly abstract model.*

Our Solution We present a method to circumvent the above problems. We combine anatomical and effective connectivity to embed the DCM model into its underlying anatomical context. The anatomical pathways, transporting the information between each pair of connected regions, are extracted and used to project the effective connectivity. We are using an animation technique for relative visualization of effective connectivity on anatomical structures.

We transfer the previously abstract model to an anatomically fortified visualization, as shown in Figure 4.1.

4.2 Background

As mentioned above, DCM aims at making estimations about the causal correlation of coupled brain regions and, even more interesting, how this coupling is influenced by experimentally induced stimuli.

DCM uses ordinary differential equations, whose parameters correspond to directed effective connectivity. This allows the construction of reasonable models of cortical regions, interacting with each other. These models are supplemented with a forward model, which describes how the neuronal activity of each node maps to the fMRI-measured responses. This way, the best-fitting model and its parameters (the effective connectivity) can be identified.

To further understand how DCM models the change of neural states, assume that each region in the brain is a state variable in the neuronal state vector x. Then, the change of x in time can be modeled using the equation:

$$\frac{dx}{dt} = Ax + \sum_{j=1}^{m} u_j B^{(j)} x + Cu. \tag{4.1}$$

Here, the variable u models the exogenous inputs, i.e., inputs from outside the system. The matrix A represents the coupling of nodes without any input. Basically, A is a connection matrix of nodes. The matrices $B^{(j)}$ describe the change induced by the j-th input u_j to the fixed coupling in matrix A (e.g., learning, attention, etc.). The matrix C describes the influence of direct input due to, e.g., stimuli.

This model of neural dynamics is used, in combination with a biophysically motivated model (the hemodynamic model), as a forward model to actually estimate the resulting Blood Oxygen Level Dependency (BOLD) signal. The estimated BOLD signal can be compared with fMRI measurements and the best fitting model is selected for each time step. The change of the state vector x over time describes the effective connectivity. Friston et al. [58] describes how the connectivity parameters in the matrices A, B, C can be calculated using an entirely Bayesian approach.

As the details and mathematics in the background would go beyond the scope of this thesis, we refer the reader to the excellent work of Friston et al. [57] and Stephan et al. [196].

To summarize the whole process for clarification, one can say that the causal influences of several cortex regions are modeled by using the neural state equation (4.1). It describes changes of the neural state vector according to external stimuli. These neural states are translated to an estimated BOLD signal and compared with measured fMRI BOLD signals. The change over time describes the effective connectivity. These models can be expressed as graphs, as Figure 4.2 illustrates. It shows the fusiform gyrus (FG) and lingual gyrus (LG) on the left and right hemispheres. The blue lines represent the anatomical connectivity and the black arrows denote the effective connection. The grayed-out parts depict the external stimuli that have been applied to this particular model. The actual effective connectivity values were taken from the original work of Stephan et al. [194].

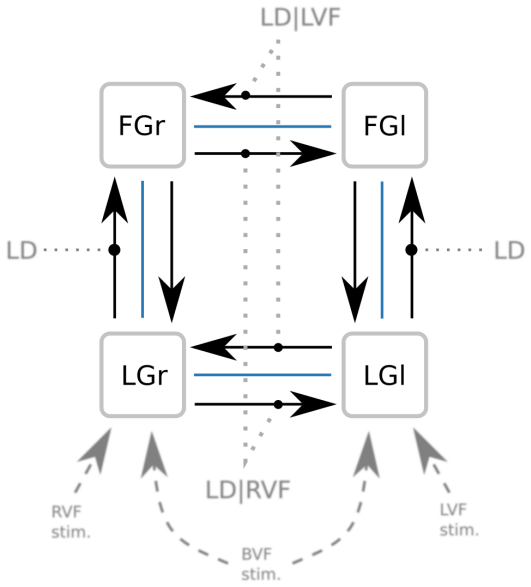


Figure 4.2: *Example effective connectivity graph. Shown are the involved regions, fusiform gyrus (FG) left and right (FGl and FGr) as well as lingual gyrus (LG) left and right (LGl and LGr). The regions are connected anatomically (blue) and, as modelled in the DCM model, by effective connections (black arrows). The connection modulation (gray dotted lines) has been modeled by task and stimulus properties. Several stimuli have been applied as individual events to the lingual gyrus in the left visual field (LVF), right visual field (RVF) and both visual fields (BVF). For more details, see [194].*

Anatomy in DCM In 2009, Stephan et al. [195] extended this approach to include tractography-based anatomical knowledge into DCM. They showed that anatomical knowledge can improve DCM tremendously. Without diving too much into the details, the modification is a Gaussian-shrinkage prior, defined as a monotonic function and depending on anatomical connectivity. As a result, the models best matching the real fMRI BOLD signal were those models that caused an increase in effective connectivity with increasing anatomical connection.

With this in mind, the need for a combined visualization providing both, the effective connectivity as well as the anatomical connectivity foundation is obvious.

4.3 Method

Now, as the basic concepts of effective and anatomical connectivity have been described, we will continue with how we visualize this kind of data in an anatomical context. We start by extracting the anatomy behind the input DCM model. This is done by selecting the fibers between all connected regions

of the model. We go on, building a volumetric representation of the fibers. Finally, we use the volume to render a surface and animate the flow of "data-packages" along it.

Notations In the following sections, the effective connectivity graph is handled as a weighted directed graph: (V, A, e). Each node $r \in V$ is anatomically represented by a region of the brain and each arc $c \in A$ correlates with a cluster of fiber tracts, corresponding to the anatomical connection. The weighting function $e(i, j)$ provides the directed effective connectivity value for each connected pair of regions r_i and r_j. Another convention, we will use in the following sections, is to treat each fiber tract as an ordered sequence of points: $f = \{x | x \in \mathbb{R}^3\}$ in the set of all fiber tracts $f \in F$. In a real-world dataset, this ordering is defined by the set of line segments $f^{segments} = \{(a, b) | a, b \in f\}$. This set also defines two designated elements:

$$f_{x_0} \in f \text{ with } \quad \neg\exists w \in f : (w, f_{x_0}) \in f^{segments} \text{ and} \tag{4.2}$$

$$f_{x_{|f|-1}} \in f \text{ with } \quad \neg\exists y \in f : (f_{x_{|f|-1}}, y) \in f^{segments}, \tag{4.3}$$

denoting the first and the last vertex of the fiber tract, if the order in f is assumed to be the order of appearance of each vertex along the tract. Practically spoken, this is the usual way of storing line data as an ordered list of coordinates.

4.3.1 Fiber Tract Selection

As the graph in Figure 4.2 implies, the anatomical connection is always defined between two distinct areas of the brain. These regions need to be known beforehand and can be extracted in several ways. In our example, the *fusiform gyrus* (FG) and *lingual gyrus* (LG) have been segmented manually. An alternative to manually segmenting the required regions is the use of atlas-based methods (i.e. Rohlfing et al. [160]).

With the help of the segmented regions, the selection of all fiber tracts, belonging to the anatomical connections in A, can be done by checking whether a fiber $f \in F$ connects the regions r_i and r_j with $r_i, r_j \in V$ and, therefore, by classifying them to belong to a cluster $C_{(r_i, r_j)}$. To actually perform this selection, Blaas et al. [17] presented a fast selection method for regular masks, boxes in their case. As our classification needs to be done with irregular masks

and needs to be computed only once, such optimization strategies are not worth the additional computational effort.

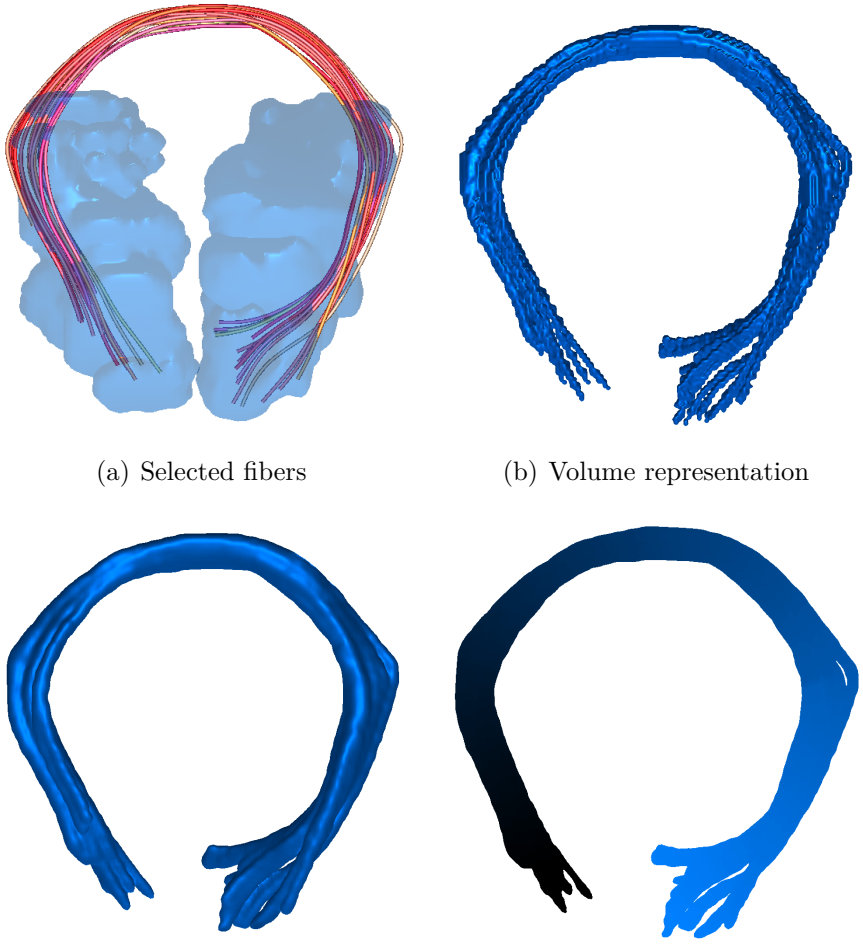
We classify each fiber tract, by simply testing each fiber tract's vertices $x \in f \in F$ against both target regions r_i and r_j, while loading the preintegrated fiber tract dataset from file. To avoid testing all vertices against all voxels of all regions, we apply a bounding box test for each vertex against the region bounding boxes to early discard vertices. The remaining vertices get tested against the exact voxel representation of the regions. This yields the list of vertices belonging to one of the regions r_i or r_j. Using the ordering of all vertices of a fiber in f and the classification of vertices to regions, we can easily reconstruct the fibers going from a region r_i to a region r_j. When starting at the first vertex in the region r_i and stopping at the last vertex of region r_j, we effectively cut the fibers to the right length. In practice, this is done by iterating along each fiber. As this fiber selection is straight forward, we omit further details here. The only thing to keep in mind is the sampling theorem. It is important to ensure that the distance between two vertices is more than two times lower than the size of a voxel in the region mask data.

Figure 4.3(a) shows a part of the forceps occipitalis selected by the left and right lingual gyrus, supplemented with the corresponding masks.

4.3.2 Fiber Tract Volumetric Representation

Depending on the location of the selection regions $r \in V$ in the brain, the amount and density of the fibers may vary. This becomes problematic for surface-based animation. The animation might not even be perceptible if the fiber tract cluster is too thin or too sparse. To avoid this problem, we create a volume representing the cluster of fibers. The volume can then be postprocessed to close holes or for thickening the cluster's volumetric representation. An alternative is the approach by Enders et al. [49], which calculates a wrapping surface around the fiber tracts. However, it may create surfaces not wrapping the whole cluster, especially if it contains strongly diverging fiber tracts.

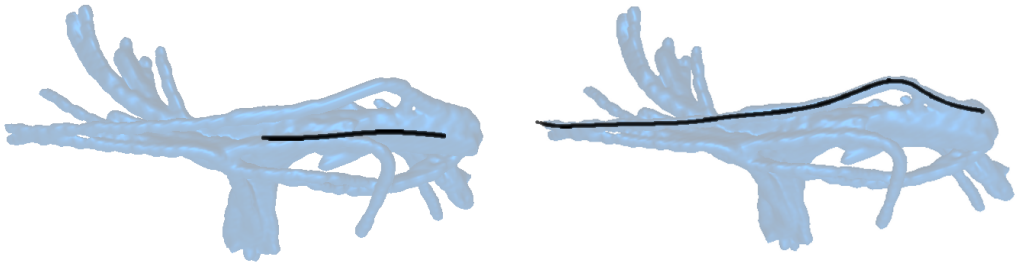
Voxelization of three-dimensional lines and line segments is covered in many publications. We are using a three-dimensional variant of the Bresenham algorithm [22] for line rasterization with anti-aliasing. This is similar to the idea in Wu's line algorithm [227]. As both algorithms are sufficiently well known, we do not go into details here.



(a) Selected fibers (b) Volume representation

(c) Smoothed volume (d) Parameterization of the volume

Figure 4.3: *The selected part of the* forceps occipitalis *between the left and right lingual gyrus (LG). (a) The fibers are filtered by the regions. (b) The volume representation of the selected fibers. (c) Applying the Gaussian filter once yields a smooth surface, maintaining the anatomical structure. (d) A parameterization along a designated parameterization fiber is used to characterize the main direction of the fiber tract cluster at each point in the volume.*



(a) Centerline as in Enders et al. [49] (b) Longest fiber tract

Figure 4.4: *The fiber tracts between the right fusiform gyrus (FG) and the right lingual gyrus (LG). (a) The centerline is too short to properly parameterize the volume along the main direction of the bundle. (b) The longest line solves the problem but it is not necessarily in the center of the bundle.*

Volumizing all fiber tracts $f \in C_{(r_i,r_j)}$ for all $(r_i, r_j) \in A$ yields several three-dimensional, discrete fields. They describe the anatomical path of information transfer for each pair of connected regions r_i and r_j:

$$v_{(r_i,r_j)}(x, y, z) \in [0, 1]. \tag{4.4}$$

The resolution is defined by the length of the smallest segment in the fiber tract data.

Figure 4.3(b) shows the resulting volume as an isosurface. The surface is very rough and, therefore, does not look natural. Applying a single, discrete Gaussian filter iteration to the volumized fiber tracts $v_{(r_i,r_j)}$ smooths the surface, while keeping the anatomical structure of the fiber tract cluster. The isovalue used in both examples is 0.3, as both datasets are in the interval of $[0, 1]$.

Until now, there is no information about the direction, nor the tangential information of the underlying fiber tract cluster available during rendering. A second volume containing a parameterization of the fiber tract cluster itself is needed. We first tried using the centerline approach by Enders et al. [49], which averages equidistantly sampled tracts to build the centerline. Correspondingly, a relatively large amount of short tracts cause the centerline to also degenerate to a short line. It does not cover the whole cluster in length, thus making it suboptimal for parameterization. The fiber tracts between the right fusiform gyrus and the right lingual gyrus are an example for this, as Figure 4.4(a) shows. The longest fiber tract in the cluster, called f^{param}, is selected for parameterization instead. Even though the longest line might not be in the

center of the cluster, nor represents it the main direction of the cluster, it is very well suited for parameterization.

To finally parameterize the volumized fiber tract cluster, an additional $parameterize_{f^{param}}(x, y, z)$ function is used, which calculates the parameter for a given point in relation to the parameterization fiber tract f^{param}. The parameterization field p is then, similar to $v_{(r_i,r_j)}(x, y, z)$, defined voxelwise:

$$p_{(r_i,r_j)}(x, y, z) = parameterize_{f^{param}}(v_{(r_i,r_j)}(x, y, z)). \tag{4.5}$$

The *parameterize* function is defined the following:

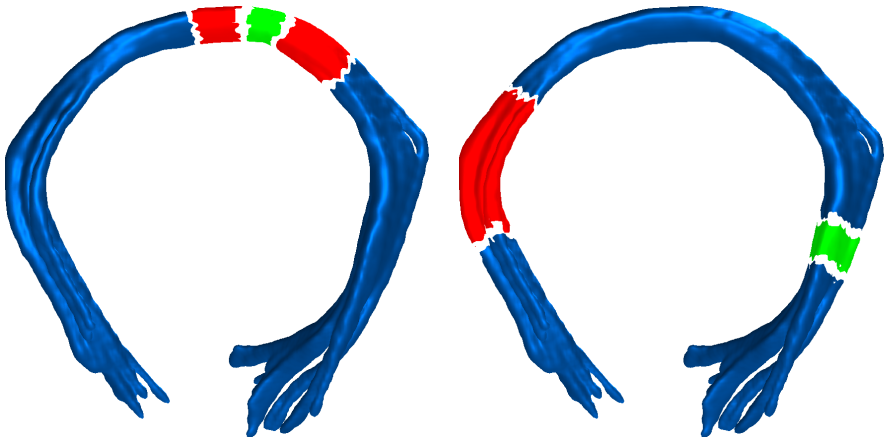
$$parameterize_{f^{param}}(x, y, z) = \underbrace{|f^{param}_{x_0}, ..., x_{nearest}|}_{\text{length of } f^{param} \text{ if cut at } x_{nearest}} . \tag{4.6}$$

The fiber tract vertex $x_{nearest} \in f^{param}$ is the nearest vertex of the parameterization fiber tract to the voxel (x, y, z). So, the distance traveled along the parameterization fiber tract up to the voxel's nearest vertex $x_{nearest}$ parameterizes the cluster – a longitudinal parameter. The parameterization field needs to be scaled from the interval $[0, |f^{param}|]$ to the interval $[0, 1]$. This normalization allows uncomplicated upload to the GPU via textures. We denote the normalized field as $p^{scaled}_{(r_i,r_j)}$. Figure 4.3(d) shows the parameterization of the masked part of the *forceps occipitalis* using color coding.

As the volumized fiber tract field $v_{(r_i,r_j)}(x, y, z)$ was smoothed earlier, the parameterization field $p_{(r_i,r_j)}(x, y, z)$ needs to be calculated only for those voxels, with a non-zero value and can be done during the Gaussian filter iteration. This ensures a continuous parameterization for all voxels involved in the anatomical path.

4.3.3 Effective Connectivity Animation

At this point, we already have the anatomical representation of the input model. This section now describes, how we represent the effective connectivity value as moving beams on the volumized tracts. As effective connectivity is a directed information, we use two beams on a tract volume to represent both "information-packages". These packages are moving from one region to the other in a real-time animation to support the metaphor of information-package transfer along a physical wire. Their length represents the effective connectivity value; the amount of data in the information-package.



(a) With overlapping (b) No overlapping

Figure 4.5: *The final rendering of the fiber tract cluster from left to right lingual gyrus. The effective connectivities along the shown connection are represented by bars. We call them beams. These beams get animated to move along the anatomical connection. This corresponds to the metaphor of moving information packages along a wire. (a) This image was made at a time step, where the beams of LG left→LG right and LG left←LG right do overlap to show how we combine both beams. (b) This image shows the beams without overlapping at another time step.*

After the fiber tracts have been selected and volumized, a smooth surface can be rendered. Typically, the marching cubes algorithm [113] is used for triangulation of volume data. Although our animation approach works on triangulated surfaces too, we are using a GPU-based ray tracing [50, 94, 95, 211] for isosurface extraction and rendering. With this approach, we circumvent any possible triangulation-related problems and achieve a topological correct surface, which renders even the thinnest fiber tract branches correctly. As this kind of volume rendering is well known, we omit the details here. Figure 4.3(b) shows the GPU ray-traced isosurface with gradient based per pixel lighting. Another, rather important note here is that we will use the term *fragment* in this section. This represents the pixel on screen, for a point on the surface. As we use ray-casting in object space, we always have a three-dimensional, object space coordinate associated with it. This allows easy access to the parameterization field $p^{scaled}_{(r_i,r_j)}(x,y,z)$. For further information on the GPU notations and coordinate spaces, we refer the reader to Chapter 6.

As the parameterization field was uploaded to the GPU too, we apply effective connectivity animation on the GPU for every fragment on the surface, by classifying each surface pixel to belong to either the highlighted pixels, highlight-border pixels, or to the non-highlighted part of the surface. Those

highlighted beams, representing the "information-packages", move from r_i to r_j and vice versa. The different sizes represent the effective connectivity value.

To ease the following descriptions, we will describe only one moving package in the direction r_i to r_j, along the corresponding anatomical connection in $v_{(r_i,r_j)}(x, y, z)$. In the next paragraphs, we call those "packages" only "beams", as they look similar to moving beams on the surface. Due to the graphics hardware architecture, a fragment has its local view only. Neighboring pixels are not accessible for write or for read. This is, why we need to calculate the current midpoint m of the "information-package" for each fragment in dependency of time t:

$$m = ((t + o) \cdot v) \; mod \; (k + \frac{k}{3}) - \frac{k}{6} \tag{4.7}$$

The Equation (4.7) is very simple and uses two parameters. The current time t with an offset o in milliseconds and the velocity v. It is the well known physical relationship between distance, time, and velocity. The offset parameter is used to avoid that the beam $r_i \rightarrow r_j$ starts at the same moment as the beam $r_i \leftarrow r_j$. This creates a better impression, as it does not look as artificial as if they would have been started at the same time, both meeting exactly in the middle of the cluster volume. The modulo operation ensures that there is a periodic beam-movement from r_i to r_j, along the current voxel's gradient in parameterization space in k units along the fiber tract cluster. In our implementation we are using $k = 100$, which creates a smooth movement. To ensure that the beam does not abruptly end when the middle of the beam reaches the end of the cluster and to ensure that the beam does not pop up on the other side with the beam's middle at the beginning of the cluster, the interval is stretched. In other words, this means that m also covers the invisible part of the parameter space, large enough to contain half of a beam's maximum size, in this case $\frac{k}{6}$ on each end of the parameter space. For more details on the size of the beams, see Section 4.3.3.

The value of m represents the current position of the beam. On the surface of the tract cluster, this can be seen as a iso-line in the parameterization field on this surface. It is perpendicular to the main direction (the current gradient) on the surface. This line moves, depending on time and speed, along the

surface. It is used for classifying the current fragments value in $p_{(r_i,r_j)}(x,y,z)$ (cf. Equation (4.8)), whether it belongs to the current beam:

$$b_{(r_i,r_j)}(x,y,z) = |m - k \cdot s_{(r_i,r_j)} p^{scaled}_{(r_i,r_j)}(x,y,z)| - \frac{l_{(r_i,r_j)}}{2}. \tag{4.8}$$

The Equation (4.8) describes an environment of size $l_{(r_i,r_j)}$ (the length of the beam from r_i to r_j), around the current beam-center m. The parameter s is used to ensure equal speeds and the correct size-relation between the beams for all beams on all fiber tract clusters and is the relation between the fiber used for parameterization of the cluster (r_i, r_j) and one of the parameterization fibers of all the clusters in A:

$$s_{(r_i,r_j)} = \frac{|f^{param}_{(r_i,r_j)}|}{|f^{param}_{ref}|}, (r_i, r_j) \in A, \text{ref} \in A. \tag{4.9}$$

The reference fiber tract is arbitrary.

Finally, b can be used as a predicate for each fragment at the current coordinate, whether it is

- inside the beam: $b < -\epsilon$,
- on the border of the beam: $b \in [-\epsilon, 0]$, or
- outside of it: $b > \epsilon$.

The variable ϵ denotes the border width. Ignoring s, this simply tests, whether the actual fragment along the main direction of the fiber tract cluster is near the current position m.

The final pixel color can now be determined using an arbitrary colormap. In our implementation, we use the following mapping, where $c_{(r_i,r_j)}$ is the color for the beam from r_i to r_j and $c_{(r_j,r_i)}$ represents the beam color from r_j to r_i respectively:

$$c_{fragment} = \begin{cases} c_{(r_i,r_j)} \text{ if } & b_{(r_i,r_j)} < -\epsilon \\ c_{(r_j,r_i)} \text{ if } & b_{(r_j,r_i)} < -\epsilon \\ c_{(r_i,r_j)} \text{ if } & b_{(r_i,r_j)} < -\epsilon \wedge \\ & b_{(r_j,r_i)} < -\epsilon \wedge \\ & l_{(r_i,r_j)} \leq l_{(r_j,r_i)} \\ c_{(r_j,r_i)} \text{ if } & b_{(r_i,r_j)} < -\epsilon \wedge \\ & b_{(r_j,r_i)} < -\epsilon \wedge \\ & l_{(r_i,r_j)} > l_{(r_j,r_i)} \\ white \text{ if } & b_{(r_j,r_i)} \in [-\epsilon, 0] \\ c_{surface} & \text{else.} \end{cases} \tag{4.10}$$

The surface itself has an user defined color $c_{surface}$, which is set, if the fragment does not belong to either one of the beams. Equation (4.10) also covers the case, where the beams overlap. If this is the case, the color of the smaller beam is used. Blending both colors would irritate the user too much. The white border around each beam ensures that the beam is visible even if the contrast between the beam's color $c_{(r_i,r_j)}$ or $c_{(r_j,r_i)}$ and the surface color is very low. Figure 4.5 shows two time steps of the animation, one with overlapping beams.

Determining the length of the beams

The effective connectivity, represented by the specific beam, is used to define its length. To have the length and, especially, their relation between each other consistent, we map the interval $[0, 1]$, representing the smallest and largest beam to the interval of the involved effective connectivities. We use the connectivity graph's weighting function e and find its minimum and maximum:

$$[min\{e(i,j)|(r_i,r_j) \in A\}, max\{e(i,j)|(r_i,r_j) \in A\}] \tag{4.11}$$

The mapping function $l_{(r_j,r_i)}$ has to map between $[0, 1]$ and the interval of the smallest to the largest connectivity value. This mapping can be adapted to the possible cases. In our examples, the effective connectivities did not differ too much. We used a linear mapping. If the effective connectivities vary very strongly, a logarithmic scale can help to avoid many very small beams of not distinguishable size and very few large beams. It is worth mentioning that the beam length interval $[0, 1]$ itself needs to be mapped to the actual beam sizes. This mapping is strongly dependent to the parameter k of the above Equations (4.7) and (4.8). A good choice is to set the beam sizes to $[\frac{k}{100}, \frac{k}{3}]$, which creates beams not too small and avoids extremely large beams covering the whole surface. In the end, this defines our $l_{(r_j,r_i)}$ to be the mapping of $min\{e(i,j)|(r_i,r_j) \in A\}$ (smallest effective connectivity value) to $\frac{k}{100}$ and $max\{e(i,j)|(r_i,r_j) \in A\}$ (largest effective connectivity value) to $\frac{k}{3}$ accordingly.

4.3.4 Labeling

Due to the scaling and the movement of the beams on the surface, it is difficult to read the actual effective connectivity value. Only relationships can be seen. Therefore, our approach is supplemented with some labeling features,

allowing the user to see the real effective connectivity value and the names of the involved regions. Figures 4.6 and 4.7 show the example fiber cluster with the corresponding labels.

To avoid that the labels overlap the actual surface and animation, we have implemented a boundary labeling approach. We place the labels, containing the region names, at the left and right side of the scene. The horizontal leader lines point to the beginning of each fiber tract cluster, defined by the first vertex of the longest line. This labeling approach is very simplistic, but works well for a few labels, as in our case. There are much more sophisticated techniques available. Oeltze-Jafra and Preim [142] present a survey on labeling approaches, focusing on medical visualization applications. In a more complex effective connectivity scenario with a lot of involved regions, we strongly recommend to use more advanced technique to avoid unnecessary overlap of labels.

4.3.5 Summary

In the last sections, we introduced the three major steps needed to visualize effective connectivity models anatomically. We have searched the fiber tracts belonging to the arcs in the model's graph, by using anatomical region masks for each node r in the graph. This created an anatomical representation of the model. We voxelized, parameterized, and rendered the previously selected tracts. During rendering, we use the programmable GPU pipeline to classify each pixel on the surface, whether it belongs to a beam or not. Using an animation scheme, we were able to animate the information-transfer between brain regions on the basis of their anatomical connection.

4.4 Results

In this section, we demonstrate our method for two different types of datasets: a real dataset (cf. Figures 4.6 and 4.7) obtained by DCM with tractography-based priors and an artificial dataset (cf. Figure 4.8). The artificial data is also derived from real anatomy, but the shown connections and connectivities are made up for increased complexity.

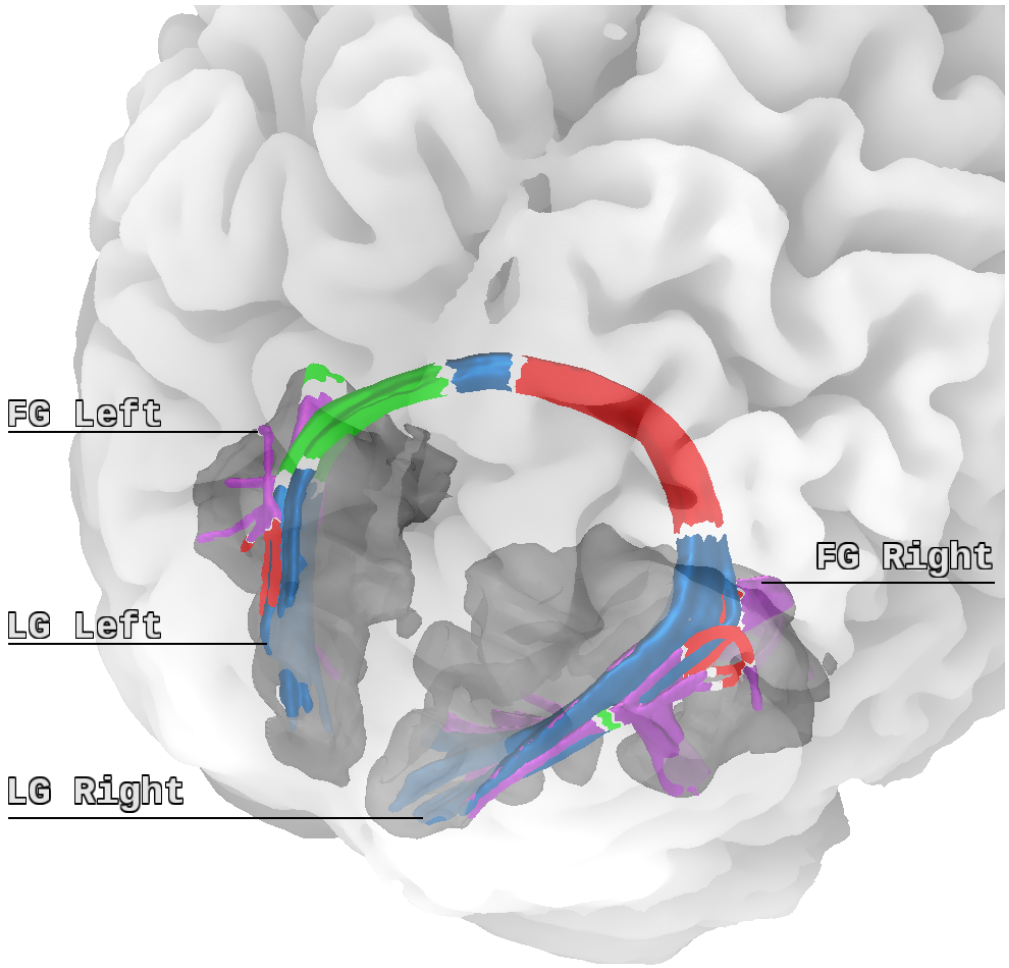
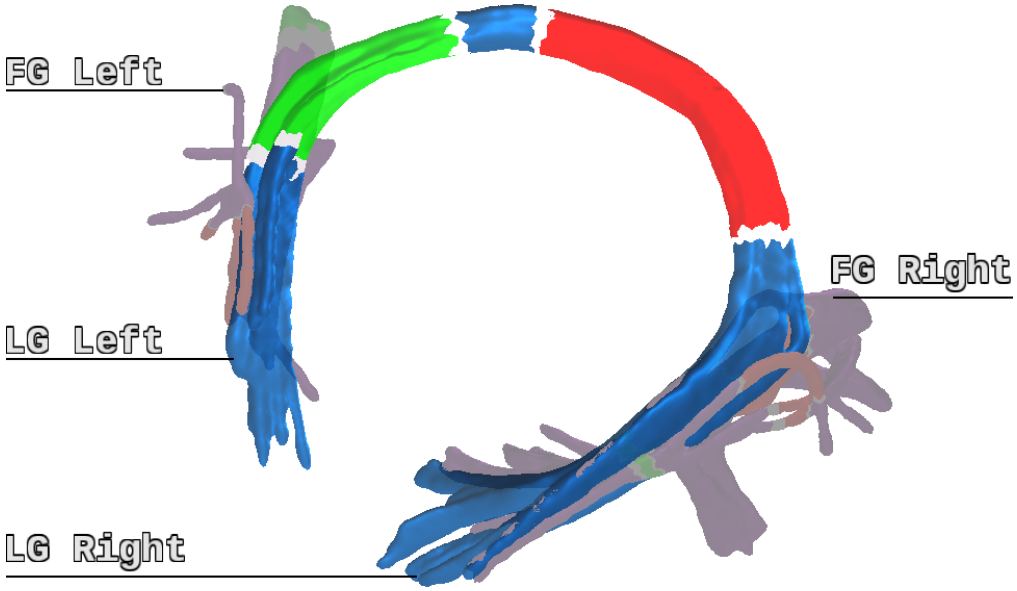
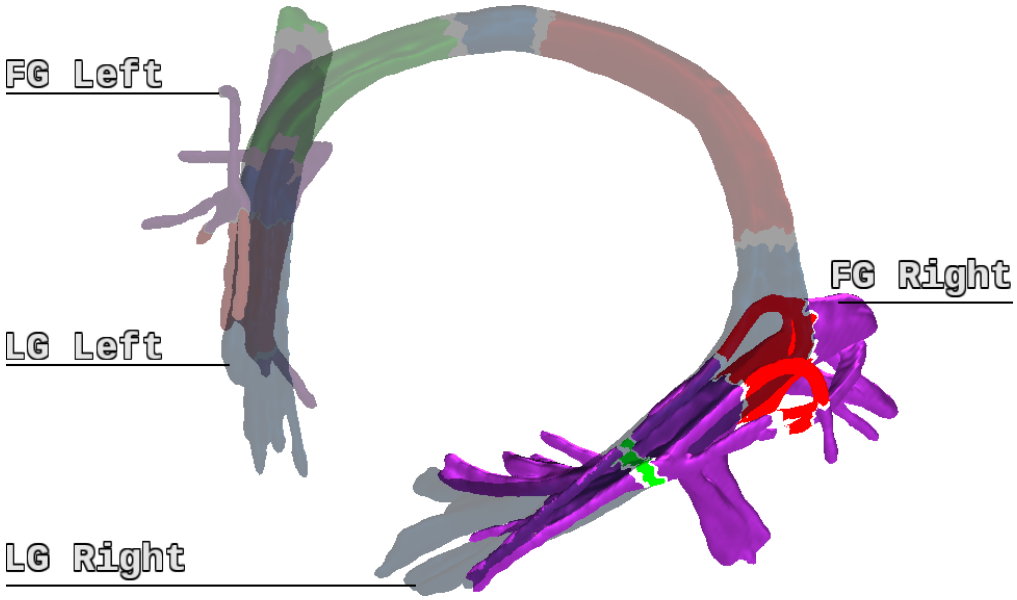


Figure 4.6: *The final rendering of the T1 context and the effective connectivity graph from Figure 4.2. The involved regions are labeled and the fiber tract clusters are colored differently. To properly understand the image, the animation is required. The animation provides the direction and, therefore, the source and target of an "information-package".*



(a) Focusing LG left - LG right



(b) Focusing LG left - FG right

Figure 4.7: *The user can selectively explore the effective connectivity graph, by highlighting the needed parts of the graph. Transparency also helps to explore occluded parts of other fiber tract volumes, or to unveil interpenetrated tracts.*

4.4.1 Data

DCM Data

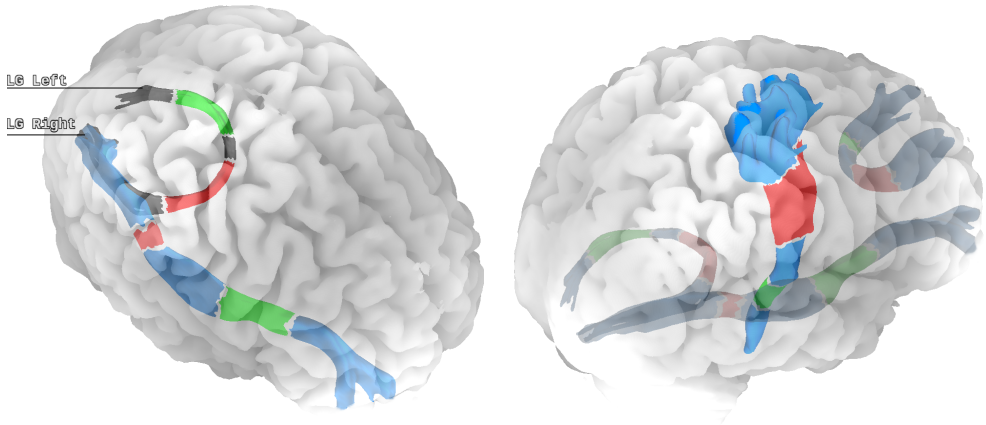
The data was taken from Stephan et al. [194] and [195]. It examined the connection of a region in one hemisphere and the connectivity to their relatively close counterpart in the other hemisphere (see Figure 4.2). The fiber tractography was acquired using a DTI measurement with 60 directions acquired with a three Tesla scanner at the Max Planck Institute for Human Cognitive and Brain Sciences. The DTI image is given as second-order tensor data; $93 \times 116 \times 93$ voxels with a resolution of $1.72mm$. The tracts we use, were computed using the method of Weinstein et al. [217]. The complete set of tracts consists of 74,313 tracts, represented by 5,472,306 vertices. The fusiform gyrus (FG) and the lingual gyrus (LG), left and right, were segmented in a MRI T1 image, measured using the same three Tesla scanner as used for the DTI data. The image, warped into standard space, has a resolution of $1mm$ for $160 \times 200 \times 160$ voxels.

The selection of the fiber tracts connecting the regions yielded only three of the links shown in Figure 4.2. The link between FG left and FG right is missing. This results from the low probability of this connection, together with the parameter setting of our deterministic tracking. Neuroscientists, who perform studies about effective connectivity, however, are able to fit their probabilistic tracking to a deterministic fiber tract dataset properly. Thus, the missing link is not a fault of our method.

Figure 4.7 shows the effective connectivity data in anatomical context. We provide labels for each region and color different tract volumes differently. Our implementation in OpenWalnut (cf. Chapter 3) allows blending out each part of the rendering separately. Working with transparency and de-saturation allows clutter reduction and context preservation at the same time. Figure 4.6 shows this with examples.

Artificial Data

For the artificial example, we took the same tract and T1 anatomy data as above, but selected arbitrary regions. The connectivity for the tract connection between these regions was chosen randomly. This approach allowed us to produce complex, yet expressive connections that help to illustrate our method.



(a) (b)

Figure 4.8: *Artificial test data for illustration: (a) shows the LG left – LG right fiber tract and the tract resulting from using the LG right – FG right regions without cropping the fibers. (b) shows the tracts from (a) and a part of the corticospinal tract, as well as the forceps minor in front of the brain.*

Fiber tract cluster	FPS
Whole connectivity graph (Figure 4.6)	12
Whole connectivity graph, no context (Figure 4.7)	22
Artificial data with context (Figure 4.8(a))	18
Artificial data with context (Figure 4.8(b))	14

Table 4.1: *Performance of the rendering in frames per second (FPS).*

Figure 4.8 shows two examples of artificial data. Even though, these examples are not necessarily realistic, they prove that our method is not only applicable to some special physical connections and effective connectivity graphs.

4.4.2 Performance

The computational effort of the method can be divided into two separate parts: preprocessing (fiber selection and volumization) and rendering. The preprocessing step runs in the order of seconds for all the datasets presented in this paper and will not be much larger for any data of reasonable size. Thus, the effort is in the same range as loading the data and is consequently negligible.

The rendering step is entirely performed on the graphics hardware. This leads to very efficient computation, yielding interactive frame rates. Table 4.1 lists some frame rate measurements and the corresponding figure. For measuring these frame rates, we have used a computer with two AMD Quad-Core

Opteron processors, 32 GB of RAM and NVidia GeForce 8800 GTX graphics hardware. The GPU can be seen as low-end consumer graphics hardware nowadays (2014). As shown in Table 4.1, we provide a visualization with high frame rates, so the neuroscientists can interactively explore their data.

4.4.3 Limitations

The presented method was a first experiment, whether it is possible to transfer effective connectivity models to an anatomically-based visualization with reasonable effort. As such, there are a lot of options to improve the method. Here, we mention two of them.

Regions Our approach strongly relies on segmented data for each of the regions involved into the effective connectivity graph. These segmented regions significantly influence the results of the fiber tract selection process. To cope with this issue, a reasonable next step is to analyse the use of probabilistic tractography and atlas based region masks. Especially probabilistic tractography creates volumetric representations already, allowing our method to skip the fiber tract selection and volume representation steps.

Animation The metaphor of moving information-packages is not well suited for transporting quantitative information on the actual effective connectivity values. It only represents relative quantities and directional information. However, our method to extract the anatomical representation of the effective connectivity data can still be used as a basis for other, quantitative visualizations to depict the actual values.

4.5 Future Work and Conclusion

4.5.1 Future Work

As mentioned above, there are several limitations in our proposed method. Next steps include the exploration of probabilistic tractography for creating the fiber tract volumes and the use of standard information visualization techniques to depict effective connectivity values in the anatomical visualization. Possible would be the use of arrows, whose size or density changes with the effective connectivity value; colormaps; hatching; changing the diameter of the fiber tract volume according to the connectivity – and much more. Holten and

Wijk [82] provides a user study on possible ways to visualize directed edges of graphs.

Even more important than proposing a lot of alternative visualizations is to evaluate them. How do they perform in the daily routine of the neuroscientists and how well do they transport the desired information?

Another important improvement is the application of ambient occlusion techniques for improved spatial perception. Especially Figure 4.6 shows the need for more spatial relations being visible in the rendering. Part II of my work will focus on this kind of computer graphics to improve the visualization results tremendously.

4.5.2 Conclusion

We presented an interactive, animated visualization for ***illustrating effective connectivity*** in the human brain. The method is not mature. There is a lot of potential to optimize the method on a technical level, as well as on the visualization level.

Nevertheless, it provides an ***intuitive and understandable*** visualization of the involved anatomical structures and the corresponding effective connectivities, using the ***metaphor of "information-packages"***. This helps neuroscientists to see and understand the information transfer between the involved regions in the context of their underlying anatomical context.

In complex DCM graphs and networks, the animation can get confusing as many anatomical paths show information-transfer and, therefore, create visual clutter. We lessen the effect of visual clutter by allowing the user to ***selectively view parts*** of the graph and fading out animation on others, thus retaining the other anatomical structures. The incorporation of focus and context principles and interactive selection of parts of the data makes it even more useful for daily use and exploration of data.

5

Visualizing Simulated Electrical Fields from EEG and tDCS: A Comparative Evaluation.

This chapter is based on the following publications:

[P5] – S. EICHELBAUM, M. DANNHAUER, M. HLAWITSCHKA, D. BROOKS, T. R. KNÖSCHE, and G. SCHEUERMANN. **Visualizing Simulated Electrical Fields from Electroencephalography and Transcranial Electric Brain Stimulation: A Comparative Evaluation**. *NeuroImage* 101 (2014), 513–530. ISSN: 1053-8119
Online: http://sebastian-eichelbaum.de/pub14a

[P6] – S. EICHELBAUM, M. DANNHAUER, G. SCHEUERMANN, D. BROOKS, T. R. KNÖSCHE, and M. HLAWITSCHKA. **A Comparative Evaluation of Electrical Field Visualization from EEG/tDCS**. *The 20th Annual Meeting of the Organization for Human Brain Mapping (HBM), Poster 3029*. 2014
Online: http://sebastian-eichelbaum.de/pub14b

5.1 Overview and Background

In this chapter, we show the value of several, common visualization methods using three well chosen and neuroscientifically relevant examples where electrical fields play a significant role. We are convinced that visualization can help to gain deeper insights into volume conduction phenomena. Those phenomena are often only statistically describable, and, at best, investigated by standard visualization techniques. We want to contribute to approach an answer to the question: "What aspects of visualization are helpful regarding electrical fields in neuroscientific research?".

Structure This chapter is structured in sections as following.

1. We introduce noninvasive neuroscientific techniques (electroencephalography (EEG) and transcranial direct current stimulation (tDCS)) that are relevant in this work and discuss visualization in this context. In the current work, tDCS was chosen exemplarily as a representative of a family of electric brain stimulation techniques, like transcranial alternating current stimulation (tACS), transcranial random noise stimulation (tRNS), transcranial electrical stimulation (TES) [147, 163] that employ scalp surface electrodes to inject electric currents.
2. We identify three generic criteria to evaluate visualization techniques in neuroscience, introduce common visualization techniques and explain their basic working principles.
3. We describe three clinically relevant examples to evaluate visualization methods.
4. We present visualization results and discuss the findings for each clinical example. We especially evaluate the mentioned three criteria for each visualization method.
5. We conclude our work and summarize general advantages and disadvantages of standard visualization techniques.

5.1.1 Electroencephalography (EEG)

Noninvasive mapping of neuronal activity is important for a better understanding of human brain function. In clinical practice, for example, the mapping is essential for the diagnosis of neurodegenerative diseases and the identification

of epileptogenic brain tissue [166]. Electroencephalography (EEG) is a noninvasive technique that is directly sensitive to the electrical activity of neuronal populations and, therefore, well suited to observe normal and pathological brain function in humans. Recording electrodes are placed on the head surface and pick up potential differences caused by Ohmic return currents, which are driven by electromotive forces in and around active neuronal areas. Electric flow fields mediate between those neural sources and the measured EEG. These fields are embedded in a very complicated volume conductor, the human head, which features many different structures of varying electrical properties (conductivities). Both, the prediction of measurements from known sources (forward problem) and the estimation of the source locations from measurements (source reconstruction) involve modeling these fields. The accuracy and precision of these estimations depend on the accuracy of the head modeling, which, in the most general case, requires a voxelwise description of inhomogeneous and anisotropic conductivity values as well as a reasonable sampling of the tissue boundaries. For more information concerning head modeling and source reconstruction, refer to the literature [220].

In order to gain insights into the complicated relationship between neural activity and measured EEG, visualization of electrical fields is of great value. It allows assessing, at one glance, which features of the head cause a large influence and, therefore, need to be modeled in greater detail. Visualization can also help to assess the effect of certain modeling errors and simplifications. Moreover, it can show, in a very demonstrative fashion, how pathological anomalies, such as holes in the skull, influence the way EEG reflects brain activity. One important prerequisite for field visualization is that the electrical field is explicitly computed within the three-dimensional head volume, using, for example, the finite element or the finite difference method. We refer the reader to the literature in this complex topic [16, 23, 35, 59, 70, 120, 166, 172, 225].

5.1.2 Transcranial direct current stimulation (tDCS)

Transcranial direct current stimulation (tDCS) is a noninvasive technique to modulate neural brain activity (e.g., [114, 126, 137, 207]) by injecting low amplitude direct currents through surface electrodes. tDCS has been known for over a century, but has recently been rediscovered as a promising tool to support a wide range of clinical applications [20, 26, 56, 100, 138, 175]. Moreover, it has been successfully applied in basic and cognitive neuroscience

research (e.g., [91, 224]). In this technique, frequently, large rectangular patch electrodes are used (normally $25 - 35\ cm^2$, e.g., Nitsche et al. [140]) in experimental settings and placed according to accepted EEG standards (e.g., $10 - 20$). In some rare cases also smaller electrodes are employed in experiments [29, 47]. To study the impact of modeling tDCS for experimental settings, electrical current density is one of the main parameters to determine physiological effects for brain and other head tissues. Visualization of tDCS simulations, like current density plots by Wagner et al. [210], can be helpful for assessing those effects as well as for understanding the way particular brain areas are stimulated depending on electrode montage and design, head geometry (e.g., skull thickness), and other factors.

5.1.3 Visualization of electrical fields

In general, when considering head modeling in EEG/MEG/tDCS analysis, the significance of certain modeling issues or particular features in the biological tissues (e.g., holes in the skull) are mostly assessed by visualizing and quantifying their final consequences, such as changes in surface potentials or mislocalization of dipolar sources (e.g., Dannhauer et al. [35]). These consequences are, however, mediated by the electric flow field in the head. Hence, visualizing the direct effects of above mentioned features in models or real head anatomy in terms of current flows and electrical potentials throughout the head might provide more direct insight into the nature of that relationship.

Generally, the literature on volume current visualization regarding EEG and tDCS [14, 141, 184] is relatively scarce. Often, visualization of electrical current is based on simple voxelwise current density visualizations represented graphically as cones, arrows [171, 182, 210], or as current density magnitudes using colormaps [182, 210]. Visualizations with more advanced techniques, such as streamlines, are rare in the EEG- (e.g., Wolters et al. [226]) or tDCS-related literature [85, 146, 170]. Characterization of visualization methods for local or global examples to evaluate visualization methods and applicability for certain tasks and domains has not yet been analyzed sufficiently. Wolters et al. [226] (for EEG) as well as Bangera et al. [8] (for tDCS) demonstrated the impact of white matter anisotropy and highly conducting cerebrospinal fluid (CSF) onto volume currents by computing streamlines using line integral convolution (LIC, Cabral and Leedom [28] and Stalling and Hege [193]). Very closely related to this paper is the work by Tricoche et al. [203], where several advanced vector field methods are shown in the context of bioelectric fields

for EEG. In most existing publications, volume current visualization is not the main focus, and visualization procedures are not used systematically to investigate the effect of features in real biological tissue (e.g., skull holes), assumptions in volume conductor models (e.g., modeling the CSF or not, taking into account anisotropy), or experimental settings (e.g., electrode montages). Such studies might help to better understand effects that otherwise can be assessed only by their final results, i.e., simulated sensor readings or source localization results [34, 35, 102].

Visualization of electrical flow fields in three dimensions can be based on either the scalar electrical potential or on the vector-valued current flow. In both cases, several principal techniques are available (see Section 5.2). The aim of this work is to demonstrate not only the advantages of certain methods, but also their drawbacks, as the applicability of these methods differ for each case, domain, and desired analysis. To achieve this goal, we will define a set of concise criteria for the usefulness of visualization techniques in the context of neuroscience and apply these to the evaluation of the presented algorithms.

5.2 Visualization Algorithms

In the last decade, visualization made a big step towards interactive and visually appealing methods, fuelled by the rapid development of affordable graphics hardware and computing devices. These developments made advanced visualization available also to neuroscience. It is important to stress that the scientific benefit of using visualization techniques is not just a matter of "pretty images", but lies in the extent to which these methods actually improve the perception, exploration, and interpretation of scientific results. Here, we identify three criteria that convey whether and to what extent a visualization technique is useful to a neuroscientist.

I) **Comparability** - The images produced by one method need to be comparable in a quantitative way over a series of subjects or time series. Colormaps play an important role in this context.

II) **Anatomical Context** - Anatomy plays an important role for exploring and navigating through the data. Without this structural context, visualized functional data loses its anatomical embedding.

III) **Interactivity** - Interactivity represents the interaction of the user with the data and its visualization. Interactivity depends on the latency be-

tween user action and visual feedback. Due to the large amount of data and the required detail of visualization, hardware and software limits can be quickly exceeded.

In this section, we *briefly* present the standard visualization techniques we use for evaluation and describe our particular implementations, which are available in OpenWalnut (cf. Chapter 3). We keep the introductions on each method short on purpose, as they are standard approaches, widely used, and sufficiently well known in the visualization community.

5.2.1 Slice View

The simplest, yet essential way of visualizing volume data is based on mostly orthogonally oriented slices cutting the data domain, often in axial, coronal, and sagittal directions. These slices in three-dimensional space are used to merge multiple colormaps, representing anatomy as well as functional data. This way, comparability in a multi-subject or time-dependent context is ensured and navigation through complicated scenarios is greatly facilitated. It is important to note in this context that an essential prerequisite of comparability is proper image registration (e.g., [112, 188]).

5.2.2 Isosurfaces

In the context of bioelectric fields and their exploration, isosurfaces can help to gain insight into the propagation of the field through head tissues in conjunction with anatomical structures. Isosurfaces can be computed from scalar potential fields, such as electrical potentials. They describe a surface in the field, where the values are equal to an user-defined, so-called isovalue. This concept allows visualization of value distributions inside the three-dimensional data field. Isosurfaces derived from electrical fields are normally used to understand the propagation of the field in a volume.

Many methods are currently available to create isosurface renderings. Most commonly known is the marching cubes algorithm by Lorensen and Cline [113] and its improvement by Nielson and Hamann [136]. The marching cubes algorithm works on the cell grid, which can be seen as the dual grid of the original voxel grid. Each cell is defined by eight neighboring voxels, forming the cell's corners. The algorithm classifies each corner of each cube according to whether the value is smaller or larger than the desired isovalue. This way, the algorithm can check whether a part of the isosurface cuts the cube. If this

is the case, marching cubes draw this surface part, depending on the inside-outside-configuration of each corner of the current cube. However, the native marching cubes algorithm might be too slow to fulfill the interactivity criterion. Therefore, many optimizations have been developed. These optimized methods make use of additional data structures to speed up mesh creation in marching cubes. Well known examples are octrees [223], interval trees [31], and a technique called span-space optimization [111]. By now, many approaches for isosurface rendering are available that exploit the calculation power of modern graphics processing units (GPU) and create isosurface renderings directly by ray-casting on the GPU [94, 95, 211].

Here, we use a ray-casting-based approach in order to ensure interactive frame rates and thereby allow direct modification of the isovalue with surface adaptation in real-time. The underlying principle is to render the bounding box geometry (the so-called proxy geometry) representing the data volume. On this proxy geometry, ray-casting is performed for each rendered pixel on the three-dimensional data domain, which is stored as a three-dimensional memory block. In other words, a ray is shot into the data volume for each pixel. If the ray hits the surface with the desired isovalue, the algorithm stops for the particular pixel and further lighting and coloring can be applied.

5.2.3 Direct Volume Rendering

Another important visualization technique is direct volume rendering (DVR), which is able to reveal features in a three-dimensional context and makes them spatially more perceivable. To achieve the volume rendering, the algorithm first needs a transfer function, which assigns a color and a transparency to each voxel of the dataset. Given this, one of the most common DVR render strategies sends a virtual ray for each pixel on screen into the data volume. Along each ray, the colors of each intersected voxel are composited using the transparencies, provided by the transfer function. This process finally defines the pixel's color on screen. This way, the physical light transport model, the theoretical background of DVR, can be evaluated in a fast and efficient way. An extensive description of this technique and its possible optimizations can be found in the literature, like *Real-time Volume Graphics* by Engel et al. [50]. Additionally, Preim and Botha [153] cover different aspects of DVR in the medical context.

Due to its ability to show whole volumes of interest, the DVR technique is widely used for visualizing three-dimensional imaging data, such as MR and CT images.

One of the greatest challenges of DVR is the transfer function design process, which can be complicated, even for experienced users. Therefore, many automatic and semi-automatic transfer function techniques have been developed (e.g., [116, 164, 236]). In this paper, however, we use manually selected transfer functions.

5.2.4 Streamlines and Explorative Tools

In flow visualization, streamlines play an important role in visualizing directional information. Basically, the streamline describes the trajectory of a particle within a vector field and can be calculated by specifying seed points. From each of those seed points, the vector field values are used to move one step towards the vector direction. This is done in an iterative fashion for each new point until a certain stop-criterion is reached. Usually, advanced step and error estimation techniques are used to achieve numerically accurate streamlines. For a more comprehensive overview, see *Fluid Mechanics* by Granger [63].

In the current work, we calculate streamlines using a fifth-order Runge-Kutta approach (as in shown by Dormand and Prince [46]) with 100,000 random seed points in the entire volume. For validation, we compare results from different runs with randomly initialized seed points. Other seeding schemes, such as spherical seeding around the source, yield similar results in our case because of the properties of the electric flow field, where all paths of the field start and end at field singularities.

For the streamline rendering, we used a combination of quad-strip-based tubes [128] and illuminated lines [117] with proper ambient shading for improved perception of structure. The idea is to render camera-oriented quad-strips instead of line-strips to emulate tubular streamlines. The illusion of a continuous tube can be achieved by adding a quadratic intensity gradient perpendicular to the tangential direction. This approach creates the effect of having cylindrical tubes at each line segment that also reduces computational complexity while having a realistic visual appearance. We combined this approach with per-pixel illumination, which creates an additional cue of line orientation in space. Furthermore, we used directional standard coloring, where the absolute components of tangent vectors are interpreted as red-green-

blue (RGB) color triples (red: left-right, green: back-front; blue: bottom-top). This coloring is common in medical visualization and helps users to grasp the local orientation of the line in space. By adding an additional ambient occlusion shading, we were able to ensure proper spatial and structural perception. We introduced this novel ambient line shading in [P10] and will present this in Chapter 8.

Streamline Selection and Clipping

Dense streamlines generate an unwanted occlusion problem. Selective rendering of streamlines is a common way to overcome this problem. Basically, there are two options: selection and clipping. Selection is a tool that allows removing whole streamlines, which match a certain criterion. This criterion can be defined either automatically or manually. A commonly known selection mechanism involves dynamic queries using multiple regions of interest (ROI) as introduced by Akers et al. [2], which were originally developed for the exploration of white matter pathways in the human brain, where it is possible to logically combine several cuboid regions in order to select white matter pathways. The query describes spatial features, such as “x is in region of interest” and “x is not in region of interest”. This way, a very fine-grained selection of streamlines can, in principle, be accomplished. However, in many cases a complex combination of several ROIs would be needed to get the desired result. Unlike automatic selection methods, ROI-based approaches can potentially be combined with general or patient-specific knowledge about anatomical structures and abnormalities. Thereby, the user can directly explore electric fields for particular anatomical features.

In contrast to selection, clipping removes all occluding parts of a rendered scene to allow direct sight onto otherwise occluded parts of the data. This process is usually accomplished with clipping planes, which can be placed and oriented arbitrarily and cut the space into two half-spaces, one visible and one invisible. Alternatively, it is possible to use anatomical structures as clipping surfaces, such as the cortical or inner bone surface. Clipping surfaces are typically used whenever no useful selection criterion can be defined or too many streamlines occlude the interesting, inner, part of the ROI.

Local Opacity and Coloring

As pointed out above, visualization of all streamlines makes it impossible to understand the complete structure of the electrical field due to occlusion. By using transparency, the occluded parts of the streamlines can also help to attain a more volumetric impression. This technique allows rendering of all streamlines at the same time, which clarifies the three-dimensional structure of the field. Similar to direct volume rendering, a transfer function is needed to map each point on a streamline to its color and transparency values. Again, the design of these transfer functions can be time consuming and application specific. Basically, we found two transfer functions very beneficial for our applications. Firstly, the curvature of the field can be mapped to transparency in a suitable way. Curvature models the angle between two consecutive tangents on the streamline [200]. Using these coloring schemes produces a volumetric impression of the streamlines and emphasizes areas with many local changes (high curvature). Secondly, interesting results can be obtained by using transfer functions, which incorporate anatomical information. In particular, portions of streamlines are highlighted by coloring if they are located within certain anatomical structures of interest, such as the skull or a target region for tDCS.

5.2.5 Line Integral Convolution

Line integral convolution (LIC, Cabral and Leedom [28] and Stalling and Hege [193]) is one of the most widely used techniques in flow visualization. LIC uses a three-dimensional vector field of a flow to create Schlieren-like (i.e., having a streaky, directional texture) patterns on a given surface. The direction that is depicted by the Schlieren-like patterns will always be orthogonal to the direction of isolines, making LIC represent the directions of the largest change in the field.

To generate a LIC rendering, one has to define a two-dimensional domain (i.e., a surface) within the vector field. On this surface, the LIC algorithm initializes random points, yielding a white noise texture. The term “texture” hereby refers to the two- or three-dimensional memory block on a graphics card, which can be used for mapping surface structure to the currently rendered geometry. The LIC algorithm then starts a streamline at each texel (texture pixel) until each texel is either the seed point of a streamline or is intersected by another streamline. With a streamline given on each texel, the LIC renderer smears the original white-noise texture along each streamline

using a rectangular smoothing filter. For a more detailed description, please refer to the literature [28, 193].

Unfortunately, the originally proposed LIC approach can be computationally expensive, which is undesirable for most interactive applications. For highest performance in terms of interactivity, we implemented the LIC approach on the GPU. The technique we employed is similar to other screen space LIC techniques [62, 104, 105] and provides the interactive performance needed for exploring the data, which is not possible with standard implementations. Another advantage of this approach is the ability to map LIC textures to arbitrary surfaces without losing performance. We applied postprocessing to the surface LIC, as described in Chapter 7. In order to compute the Schlieren-effect on the GPU, the vector field is projected to screen space, and so is the initial noise texture on a surface. In the following step, the projected surface and vector field are smeared directly, by using several steps of Euler integration for each pixel. In other words, the GPU-based LIC algorithm does not compute whole streamlines, but uses only fragments of the streamlines. This implementation creates a similar effect as the classical LIC, but is computationally less expensive. A main drawback of LIC is its intrinsic two-dimensionality. In three-dimensional space, LIC-like methods exists [53] but have to deal with occlusion, which might be possible to solve to a certain degree using transparency.

5.3 Application Cases

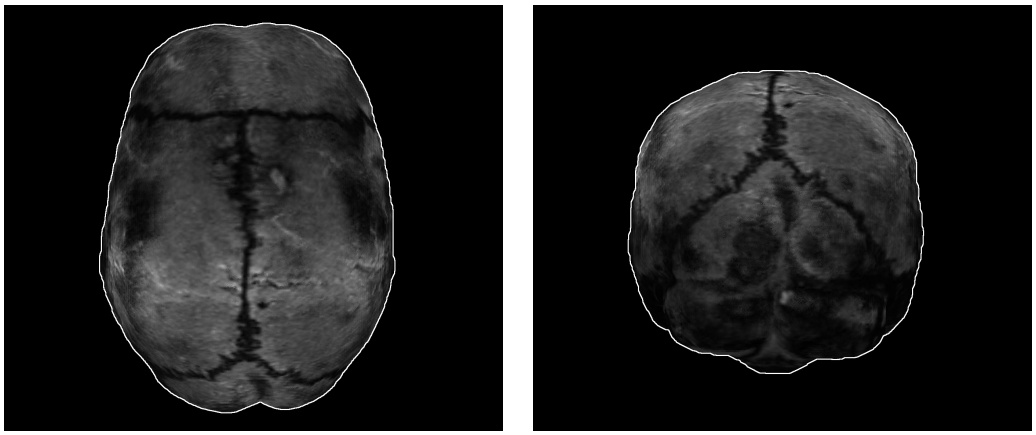
In the following section we will describe three neuroscientifically relevant applications for electrical field visualization in the human head. The first two examples deal with the electrical modeling of the human skull in terms of volume conduction. The skull, with its very low conductivity, is the major obstacle for Ohmic currents on their way between sources and EEG electrodes. Hence, the correct modeling of the skull is of major importance for EEG-based source reconstruction [35] and also for tDCS forward modeling [37, 199, 210]. Visualizing the influence that different aspects of skull modeling have on the electric flow field can provide important insights into the relationship between neural activity and EEG readings, as well as elucidate the impact of errors and simplifications on modeling accuracy [210]. Here, we will first visualize the effect of a hole in the skull, for example due to injury or surgery. For this purpose, we use a finite element model of a human head, introduced by

Lanfer et al. [102]. In the second case, we investigate how the intact skull can be modeled with various levels of detail [35]. Skull modeling has also been of general interest in recent tDCS literature [37, 156]. For all simulated volume currents, in the first to examples (EEG), the Saint Venant source model (linear basis functions, transfer matrix approach, [35, 166]) was used, which is implemented in SimBio/NeuroFEM toolbox [40]. The third application evaluates the visualization of electrical current based on an electrode placement common in tDCS settings. The forward solution for tDCS was computed using software implemented in the SCIRun package by Dannhauer et al. [33].

5.3.1 Modeling a Hole in the Skull

In clinical practice, EEG is a widely used tool to investigate and monitor brain function. It can be utilized, for example, in the treatment of epileptic patients in order to investigate and localize epileptic seizures [166]. The treatment of those patients often involves surgery, where epileptogenic and tumorous brain tissue is removed. In many cases, several surgeries have to be performed to finally remove all epileptogenic tissues, leading to significant differences in volume conduction due to the removed tissue and remaining skull holes. It is still not entirely clear how the EEG, generated by differently oriented and positioned electrical current sources, is affected by skull holes in their vicinity. Therefore, we use all the previously described visualization techniques (previous section) to investigate local and global changes of volume conduction in the presence of a skull hole (denoted Skull-Hole-Model). The impact of the skull hole is evaluated with regard to the direction to which a source near the hole is pointing (Direction 1: perpendicular to skull surface; Direction 2 and Direction 3: tangential). Instead of placing the current source directly underneath the hole, we chose the slightly more interesting case in which the dipole is placed near the hole. One of the two tangential directions (Direction 3) has a larger component pointing towards the hole than the other one. It is well known that the direction of a current source has a major impact on scalp potential distributions - in fact, it is more important than the location of the source. If source directions are known (cortical surface constraint, [110]) from anatomy, e.g., derived from MRI, the solution space can be reduced to improve source localization. Visualization can make a contribution to better constraint dipole locations in source localization problems.

The Skull-Hole-Model introduced by Lanfer et al. [102] comprises 10 tissue types with different isotropic conductivities: scalp, muscle, fat, soft tissue



(a) Superior (b) Posterior

Figure 5.1: *Visualization of skull bone plates from MRI. Human skull bone tissues, 2 mm below the skull surface, based on a T1-weighted magnetic resonance image is shown here. The coronal, sagittal, and lambdoidal sutures appear darker (zig-zag-pattern). The sutures join skull bone plates together. The figure highlights soft bone tissues (brighter areas in figure) within skull plates that are separated by sutures. A white outline is added to clearly show the object boundaries.*

(e.g., eyes), soft bone, hard bone, air, cerebrospinal fluid, gray matter, and white matter. All generated field differences are computed by subtracting the electrical field of the Skull-Hole-Model from the electrical field of the reference model (without hole).

5.3.2 Modeling the Layered Structure of the Skull

In general, head modeling involves certain simplifications. These simplifications are motivated by the need to keep calculations tractable and by the limited availability of information, for example, on tissue conductivities. The skull comprises three layers of different conductivities: two outer layers of hard bone and, sandwiched between them, a layer of soft bone [169] (not always present, see Figure 5.1). This fact can be accounted for by different models. For more details, please refer to the work of Dannhauer et al. [35]. Here we explore the following possibilities: (i) modeling three layers of bone, using measured conductivity values from the literature [3]; (ii) assuming a single homogeneous isotropic conductivity, using a standard value from the literature ($\sigma_{hard/soft\ bone} = 0.0042\ S/m$); (iii) assuming a single homogeneous isotropic conductivity, determined by fitting an optimal isotropic conductivity estimate to the three-layer model ($\sigma_{hard/soft\ bone} = 0.01245\ S/m$) using a bisection method within the range of hard ($\sigma_{hard\ bone} = 0.0064\ S/m$) and soft

bone ($\sigma_{soft\ bone} = 0.0268\ S/m$) conductivity (more details in [35] and review subject 3, IH model). The terms soft and hard skull bone are also known in the literature as *spongy* and *compact* bone [35]. The skull modeling using an isotropic conductivity of $\sigma_{hard/soft\ bone} = 0.0042\ S/m$ has been common practice for decades. Dannhauer et al. [35], in accordance with earlier work by Oostendorp et al. [143], could show that a value of $0.01\ S/m$ is more appropriate. Since $0.0042\ S/m$ still appears sporadically in default settings in EEG [109] and older software packages for source localization, we compared its effect in a qualitative manner. The rest of the head, both inside and outside the skull, was modeled as homogeneous compartments (skin: $0.43\ S/m$, brain: $0.33\ S/m$). For this model (referred to as the 3-Layer-Model), we demonstrate the use of the LIC and streamline approaches.

5.3.3 Stimulating of Brain Tissue using tDCS

Up to this point it has not been well understood how experimentally applied tDCS affects tissues of the human head. In consequence, the exact impact of electrode montages, parameterization of electrical stimulation, and volume conductor properties in tDCS is still subject to research (for more details see below). In clinical environments stimulation parameters are often based on examples taken from the literature and might not be always ideal for individual subjects Datta et al. [38] and Minhas et al. [130]. Furthermore, information from literature is limited to certain stimulation setups and, therefore, new experimental protocols are difficult to establish without having knowledge of their impact on head tissues. Visualization of simulation results can make a real contribution to help to understand general effects of tDCS to the human head and especially to brain tissues.

In order to evaluate the implemented visualization algorithms we performed tDCS simulations using a realistic head model. The model is composed of 8 tissues (skin, skull, cerebrospinal fluid (CSF), gray matter, white matter, eyes, internal air, electrode material), which were derived from a multimodal integration approach. Skin, skull, and internal air were derived from a computed tomography (CT) dataset (GE CT Scanner, General Electrics, Fairfield, United States; 1 mm isotropic voxel resolution). Gray and white matter as well as eyeballs were derived from a MRI dataset (1 mm isotropic voxel resolution) acquired with a 1.5 T Magnetom Symphony (Siemens Healthcare). We used the tool BrainK [108] to combine the data acquired from different imaging modalities in order to integrate them into the tissue segmentation. An automated pro-

cedure implemented in BrainK was used to extract and, if necessary, manually correct, the different tissue segmentations. Furthermore, the tissues, such as eyeball, etc., could be extracted based on the available MRI contrast and modeled as homogeneous segmentation masks. Two patch electrodes (surface area: $50 \times 50\ mm$, $5\ mm$ height) were placed on the head using a C3-Fp3 ($10-20$ system) electrode montage to target the primary and secondary motor cortex. Based on the tissue segmentation, a tetrahedral mesh (43.7 million elements, 7.7 million element nodes) of the head and electrodes was generated using a novel meshing package (cleaver V1.5.4, [24]) that preserves conformal mesh boundaries and guarantees a certain mesh quality (dihedral angles 4.7-159.1). Isotropic conductivity tensors Dannhauer et al. [33] were assigned to each of the tetrahedral elements depending on tissue type: skin ($0.43\ S/m$, [35]), skull ($0.01\ S/m$, [35]), cerebrospinal fluid (CSF, $1.79\ S/m$, [11]), gray matter ($0.33\ S/m$, [35]), white matter ($0.142\ S/m$, [72]), eyes ($0.4\ S/m$, [36]), internal air ($1e-15\ S/m$, [36]), electrode material ($1.4\ S/m$, [36]). A stiffness matrix was computed for the resulting FEM model using the SCIRun environment [33]. For the two current injecting patch electrodes, the electrical boundary conditions were considered using the complete electrode model by Polydorides and Lionheart [152] and Somersalo et al. [190], considering an electrode-skin impedance of $5\ k\Omega$.

To study the effects of volume conductor modeling for EEG and tDCS stimulation, we performed careful simulations. Our modeling efforts naturally contain modeling simplifications (e.g., no white matter anisotropic conductivity modeling) with respect to realistic conditions. However, we believe that our head models capture important features of volume conduction and, therefore, results as well as the drawn conclusions are helpful to understand better specific effects in EEG and tDCS. Experimental validation in clinical settings is still an indispensable issue. Only a few studies in the literature have focused primarily on experimental validation of current injection. In an early animal study, Hayes [73] investigated current injection in vivo using anaesthetised spider monkeys, injected $58mA$ through surface electrodes and measured voltages at intracerebral probe sites. The author was able to estimate different tissue resistivities (scalp, skull, brain) to investigate their effects on the current flow through the monkey's head. To obtain results for human physiology, Rush and Driscoll [167] used data from an electrolytic tank that contained a half-skull structure with attached surface point electrodes. Currents were injected throughout the surface electrodes at different locations and electrical potentials

were measured, its attenuation was depicted with respect to the skull center and resistivities were estimated. For a human volume conductor model, and finite tDCS electrodes, Datta et al. [39] validated their simulations with experimental electrode readings (errors for potentials between 5-20%) conducted using a whole head electrode array and low amplitude current injection ($1mA$). Besides empirical evidence supporting the effects of tDCS-like technologies in a broad range of medical applications (see above for more details) in human, there are numerous studies investigating cortical excitability and activity alterations induced via tDCS (for more details see e.g. [25, 26, 126, 137, 139, 148, 192]). For example, Caparelli-Daquer et al. [29] as well as Edwards et al. [47] used event-related potentials (EEG) to prove the ability of focal stimulation of the motor cortex using tDCS. The used volume conductor models in the current work, 3-Layer-Model and Skull-Hole-Model [35, 102], are based on segmentations from structural MRI contrasts similar to many studies in the literature [33, 34, 36, 37, 38, 39, 70, 85, 107, 130, 166, 170, 210, 226]. However, the head model used for tDCS in this work represent a more novel type that incorporates multimodal imaging data (MRI, CT, cf. Section 5.3.3) for more realistic modeling of scalp, skull, and internal air cavities [132]. It also features a more advanced current injection formulation (complete electrode model, [190]) that is frequently used in electrical impedance tomography [152]. For all three applications cases, the volume conductor models were parameterized with respect to tissue conductivities (see above for more details) widely applied in recent literature.

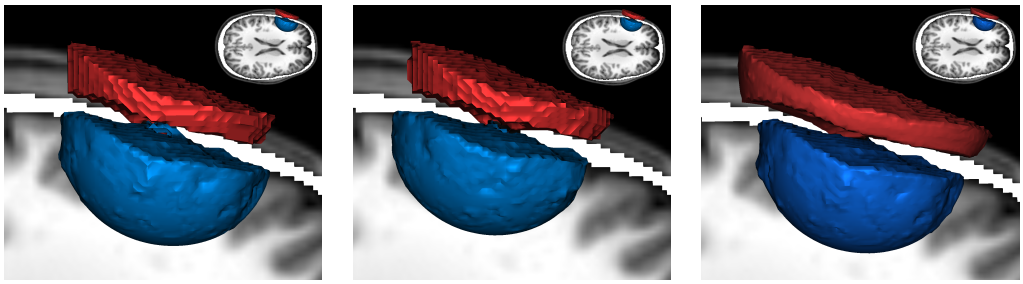
5.4 Results and Discussion

We have applied the methods from Section 5.2 to all three application cases. In this section, we evaluate and review the usefulness of the visualization methods for the chosen applications with respect to the three criteria described above: comparability, anatomical context, and interactivity (see Section 5.2).

5.4.1 Surfaces and Direct Volume Rendering

Isosurfaces

We applied the interactive isosurface ray-tracer to the Skull-Hole-Model data and visualized the scalar electrical potential as the difference between modeling approaches. Figure 5.2 shows isosurfaces (red for $+0.2\ \mu V$; blue for $-0.2\ \mu V$)



(a) Direction 1 (radial) (b) Direction 2 (first tangential) (c) Direction 3 (second tangential)

Figure 5.2: *Isosurface renderings for the Skull-Hole-Model. These isosurfaces show, for each source direction, the potential differences (red for $+0.2\ \mu V$; blue for $-0.2\ \mu V$) between the Skull-Hole-Model and the reference model. For the positions and orientations of the dipoles, see Figure 5.12. These surfaces denote the border between the volume with an absolute difference of more than $0.2\ \mu V$ on the inside and less than $0.2\ \mu V$ on the outside. Directly comparing the colormaps of the reference field and the skull-hole fields does not allow a quantitative rating of differences between the two fields. Using the difference field instead unveils the structural difference caused by the skull-hole very explicitly. The images demonstrate that the influence of the skull hole is different (more wide-spread) for the second tangential source orientation (Direction 3). It is clearly shown that the skull-hole only influences the area around the hole and that the difference of reference model and Skull-Hole-Model on the remaining field is rather low.*

generated from a source located near the skull hole, in difference to the reference model without hole. The rendered isosurfaces represent the boundary of a spatial domain, where the absolute potential difference between the models exceeds a value of $0.2\ \mu V$. These rendering clearly show that the skull-hole influences the electrical field only near the hole itself. Note that, while the visualization of an isosurface of the potential difference is useful, as it renders a volume within which significant differences occur, isosurfaces of the potentials in either condition are far less useful, as the potential value depends on a reference (so, one would render a volume, where the potential is close to the one at the reference electrode).

Comparability - In general, isosurfaces allow a high degree of comparability, and proper lighting can support a direct comparison of local shape and structure. Additionally, colormaps are useful in order to give cues about the surface potential or current density magnitude, which in turn increases comparability. Note that comparability is ensured only if the range of the values in all datasets is the same. Thus, normalization might be needed.

Anatomical Context - The isosurface approach has some significant advantages with respect to its anatomical embedding. Firstly, isosurfaces can be

rendered in combination with other objects, such as slices or surfaces. Secondly, isosurfaces can be combined with anatomical information, e.g., from magnetic resonance imaging (MRI). Naturally, anatomical context can help to increase comparability. However, combining anatomy and colors could also create confusing renderings, if too much information is combined into one color. A possible solution to overcome this problem is to use orthogonal slices for anatomy, as shown in Figure 5.2.

Interactivity - Since the isosurface renderer is implemented on the GPU, the interaction with surface renderings and surface modifications can be done without a significant loss of performance. For example, the modification of isovalues allows for a direct real-time exploration of the potential field and its propagation inside the head, just by pulling a slider.

Skull-Hole-Model Isosurfaces used to render electrical fields and differences between electrical fields can help to interactively explore these fields. In Figure 5.2, the electrical field difference between the Skull-Hole-Model and its reference model (same, but without hole) for all three source orientations is rendered. It can be seen that all three source orientations lead to similar difference renderings. With closer inspection, the radial direction (Direction 1) and the first tangential source orientation (Direction 2) have a more similar appearance than the second tangential direction (Direction 3). It appears that the second tangential direction (Direction 3) is more influenced by the presence of the skull hole. This is expected, as this direction is pointing towards the center of the hole. With the help of LIC, this can be shown more clearly, as done in section 5.4.3.

3-Layer-Model For the 3-Layer-Model, isosurfaces are not very useful since model differences are diverse and inhomogeneously distributed in the skull. Hence, it was difficult to define meaningful surfaces based on isovalues for this particular application.

tDCS In Figure 5.3, the current density magnitude is depicted (without isosurface truncation) on orthogonal slices cutting through all materials modeled in the volume conductor for the tDCS example. It can be seen that the highest current density magnitudes seem to be located on the electrode sponge-scalp interface [191]. Further, the impact of high conducting CSF can be clearly seen with higher current density magnitudes values close to the injecting electrodes. The current density magnitude is almost zero in the air-filled cavities

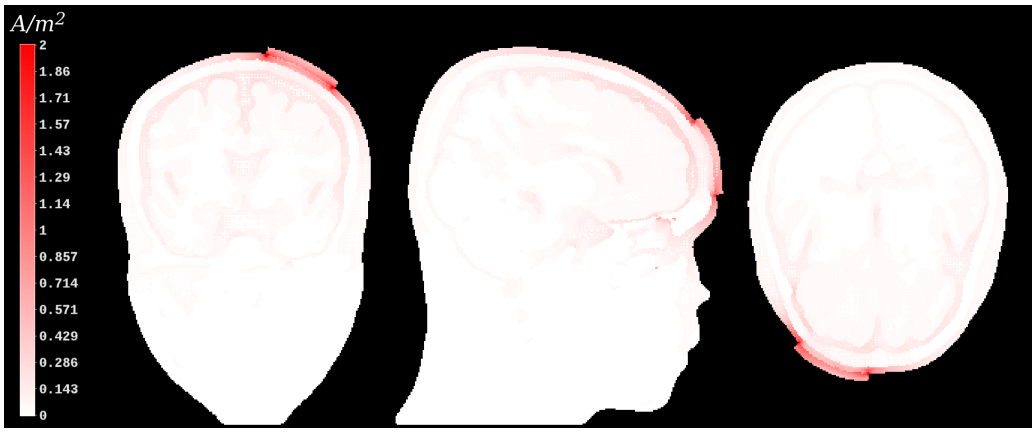


Figure 5.3: *Current density magnitude plot for tDCS example on cutting plane. A coronal, sagittal, and axial view of the volume conductor, where current density magnitudes (white-to-red colormap) are mapped. High current density concentrations are present at the electrode sponge-scalp boundaries as well as in CSF. Although the current density around the electrode sponge-scalp boundaries was maximally up to* 4.2 A/m^2*, we have chosen a windowing interval of* $[0, 2]$ A/m^2*. This way, we are able to show the rapidly decreasing current density in vicinity of the sponge-scalp boundaries, which, otherwise, would not be seen as their value would be mapped to a nearly white color.*

and small in the skull tissue. Furthermore, in Figure 5.4, the current density magnitude is mapped onto material surfaces: scalp, skull, and brain. The visualization clearly shows the impact of the different conductive materials on the current density. As also implied in Figure 5.3, the increased current densities are concentrated around the edges of the electrode sponge, with the highest values near the corners. The current density on the skull surface is only slightly smeared out since the skin is just 2-3 *mm* thick and skin resistance is not very high compared to other materials (skull, air). However, the current density on the brain surface is very broadly distributed due to the low conductivity of skull tissue and the high conductivity of CSF. Another important point to mention here is the window-function used to map a certain current density magnitude interval to a color intensity interval. In Figure 5.4, the values on each tissue are mapped to the full white-red interval using a different window for each tissue. This windowing is motivated by the rapidly decreasing maximum magnitude when moving from the head surface towards the brain. Without the windowing, the colormapping on the brain would be nearly white.

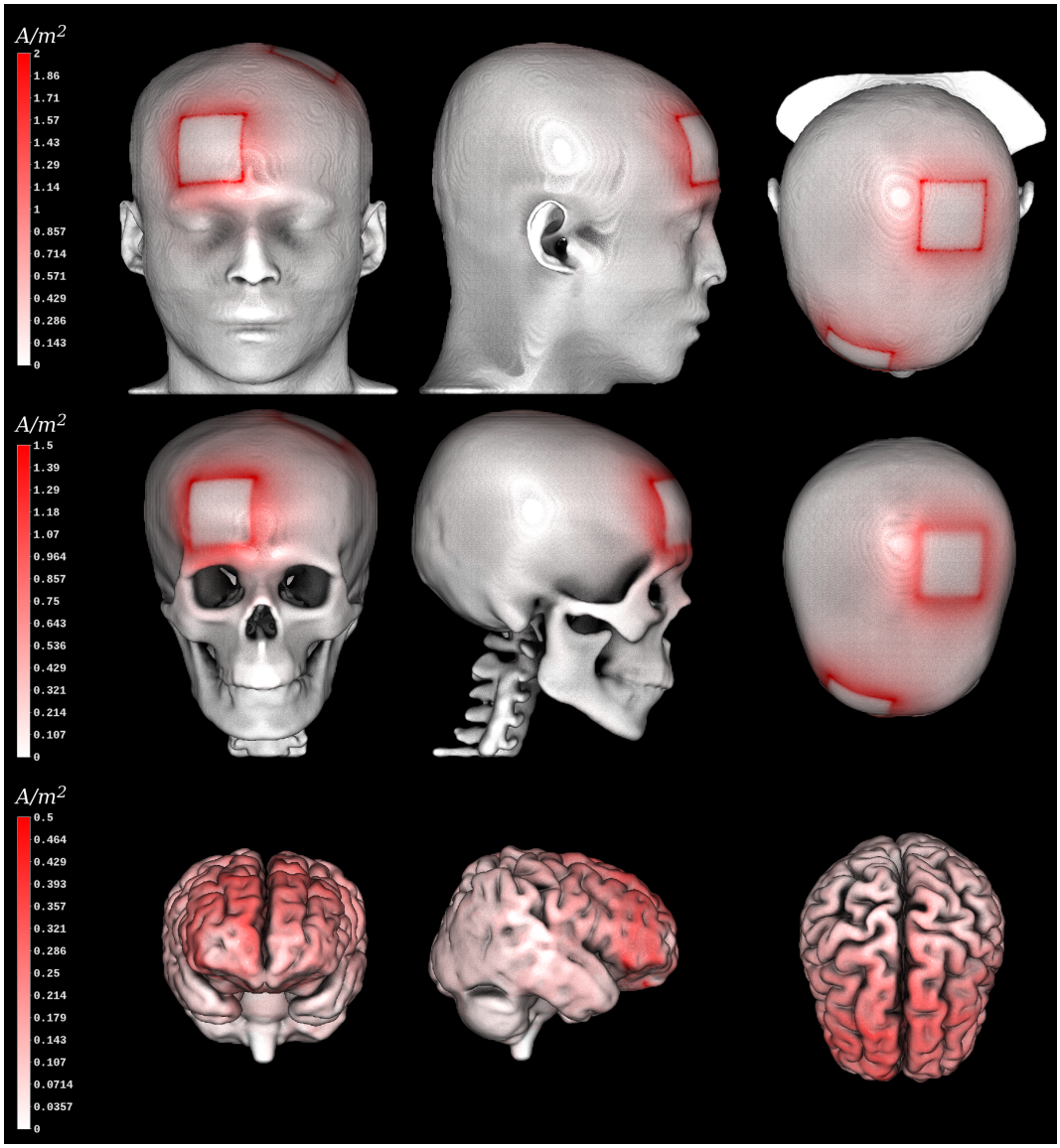
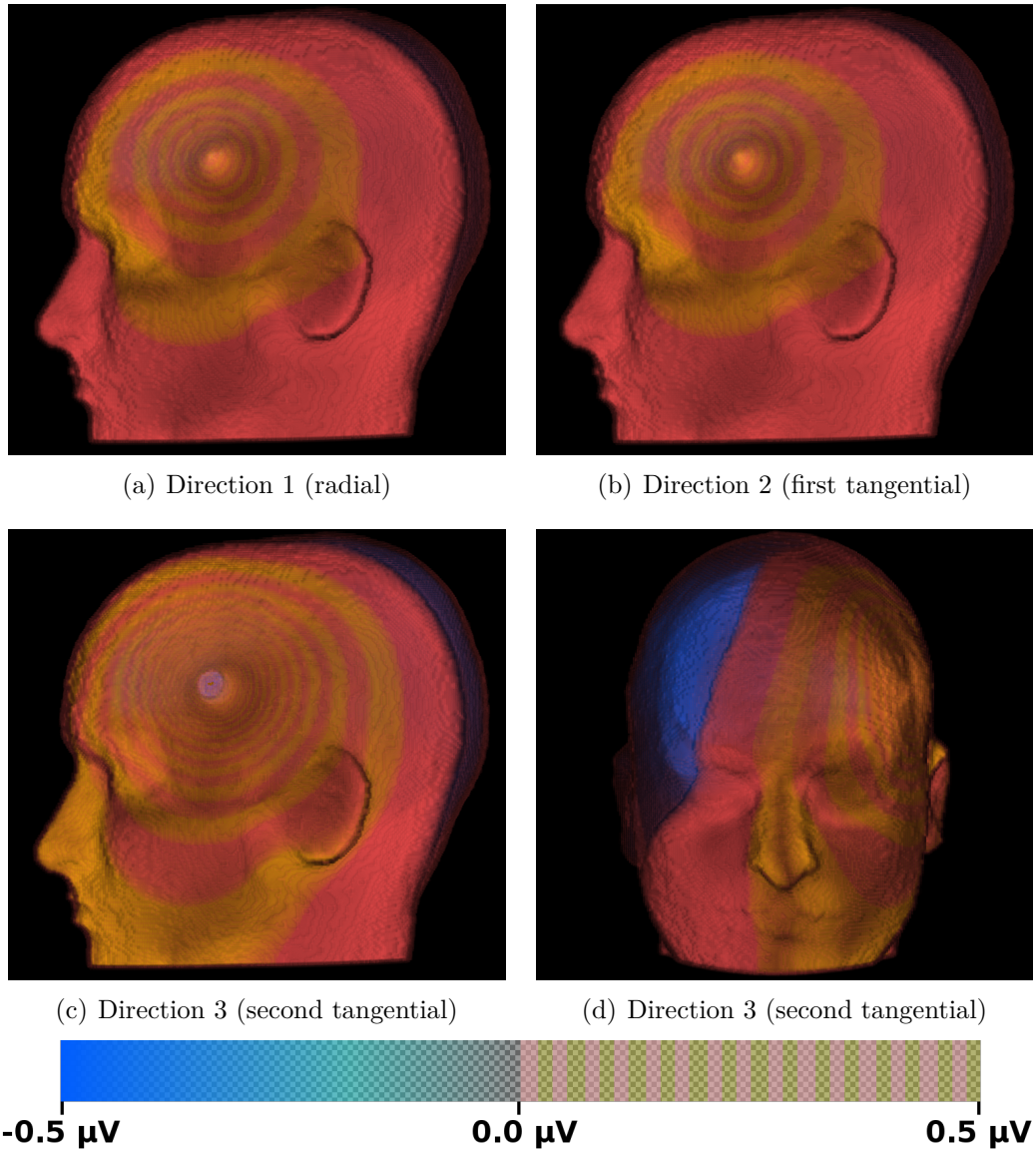


Figure 5.4: *Current density magnitude (white-to-red colormap) computed for a standard tDCS electrode setting displayed on tissue boundaries: scalp ($[0, 2]$ A/m^2), skull ($[0, 1.5]$ A/m^2), and brain surface ($[0, 0.5]$ A/m^2). We have used different windowing intervals for each tissue boundary to cope with the rapidly decreasing current density. This way, we avoid that the maxima on the skin influence the coloring on inner tissues. It can be seen that the conductivity profile of the modeled materials has different effects on the current density distribution.*



(a) Direction 1 (radial)

(b) Direction 2 (first tangential)

(c) Direction 3 (second tangential)

(d) Direction 3 (second tangential)

Figure 5.5: *Direct Volume Rendering (DVR) for the Skull-Hole-Model. DVR for the potential difference fields for each source orientation in the Skull-Hole-Model. As Figure 5.2 indicated, the skull hole has the strongest influence on the field simulated from the second tangential direction. The used transfer function shows the spreading potential difference between the Skull-Hole-Model data and the corresponding reference field. The transfer function maps negative potential differences to blue and positive differences up to* $0.5\ \mu V$ *to a color pattern switching from red to yellow every* $0.033\ \mu V$*. This way, the spreading structure can be visualized in an intuitive way using direct volume rendering and is conceptually similar to isolines but has the advantage of also showing the spatial extend of intervals. For the positions and orientations of the dipoles, see Figure 5.12.*

Direct Volume Rendering

Similar to isosurface renderings, we applied a red-blue colormap to denote positive and negative potential differences for the Skull-Hole-Model. Figure 5.5 depicts a volume rendering, with a specific transfer function. This transfer function was designed to specifically emphasize the gradient of the potential difference outside the skull hole, rather than its absolute values. For this purpose, we stippled the positive part of the transfer function to map the positive potential difference to alternating colors (red and yellow in this case). The negative part is a fading blue, to show the negative potential difference inside the skull. This is conceptually similar to isolines, but has the advantage of also providing information on the spatial extent of a certain value interval within the data.

Comparability - Like isosurfaces, DVR can provide a high comparability, if transfer function and data range stay the same over all datasets. Transfer functions, which were designed to unveil certain features or value distributions in the data, can provide a particularly high degree of comparability (e.g., Figure 5.5). Admittedly, this is not true in general and depends on the transfer function and the data. Focus and context techniques, like the magic volume lens [212], can help to selectively compare certain areas of the volume. However, unlike isosurfaces, DVR suffers from a lack of clear and crisp surfaces. Local illumination can additionally help to create surface-like effects, which influence the colormap. Overlap and high transparency in the transfer function further complicate comparisons over multiple renderings as they falsify the coloring of certain features or structures.

Anatomical Context - The combination of DVR and anatomical structures is a difficult problem. The additional use of orthogonal slices with anatomical colormaps is a difficult task as well. Figure 5.5 shows a feature-enhancing transfer function, where the shape of the head happens to be reflected quite well by the shape of the potential field.

Interactivity - Modern GPU implementations of DVR are able to perform high-quality volume renderings in real-time with interactive transfer function design. The interactive modification of transfer functions with an easy-to-use interface is important to allow neuroscientists to explore datasets with different parameters quickly and intuitively.

Skull-Hole-Model In Figure 5.5, a DVR of electrical field differences is shown for all three source orientations. To emphasize specific changes of positive

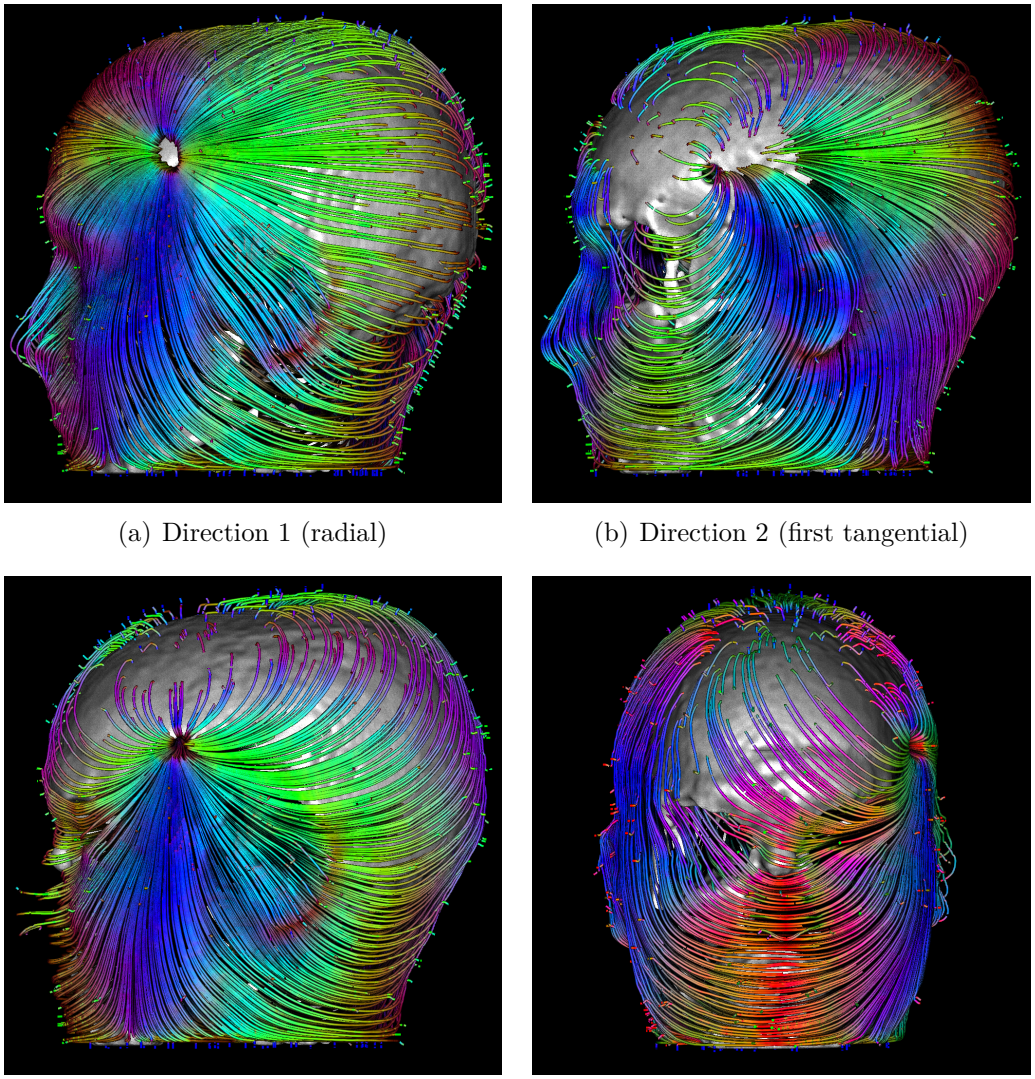
potential differences, the transfer function includes an alternating red-yellow colormap (see Figure 5.5). For the negative potential differences, the transfer function uses a blue-transparency fading. It can be seen that positive potential differences are present in outer parts of the head (mainly in skin tissue). The negative range of the potential differences is primarily present inside the skull (in the brain tissues), whereas the biggest differences are close to the skull hole. In comparison to isosurfaces, we obtain similar results. DVR results for Directions 1 and 2 appear similar in contrast to Direction 3. Even though Direction 1 and Direction 2 look similar, there are potential differences, mainly in the brain tissue. It is also apparent that the potential gradients point radially towards the center of the hole, but their strengths are modified by the head shape and clearly differ for Direction 3 as compared to the other two directions. Another interesting finding is visualized by the different spatial frequencies of the circular structure. This pattern is different for Direction 1 and Direction 2 as compared to Direction 3, which has a much higher spatial frequency. This frequency indicates that the potential differences in Direction 3 increase much faster around the skull hole. The higher spatial frequency also proves that for this particular source orientation, the skull hole has the biggest effect. This information could not be conveyed by just one isosurface. DVR provides a simple way to represent multiple value ranges, which spatially overlap.

3-Layer-Model Similar to using isosurfaces, it is difficult to gain any benefits and new insights into volume conduction from using DVR for the 3-Layer-Model due to the very local effect, confined to the skull compartment. It is hard to model a proper transfer function, which would be able to provide the needed resolution for seeing local details without the inherent occlusion.

tDCS Also for the chosen tDCS example, it appeared difficult to design a proper transfer function to highlight the mostly local effects. The situation is further complicated by the fact that similar ranges of current density magnitude values are present in skin and CSF tissue, which would lead to significant occlusion effects.

5.4.2 Streamlines and Explorative Tools

In this section, we explore streamlines and streamline rendering methods in all three application cases. We calculated streamlines for all model variants.



(a) Direction 1 (radial)

(b) Direction 2 (first tangential)

(c) Direction 3 (second tangential)

(d) Direction 3 (second tangential)

Figure 5.6: *Streamlines depict the electrical flow field in the Skull-Hole-Model. The skull mask, including the hole, has been added to provide anatomical context. As already seen in Figures 5.2 and 5.5, the influence of the skull hole seems to be nearly identical for source orientations Direction 1 and Direction 2. With the second tangentially oriented source (Direction 3), the field leaves the skull through the hole and enters it again through the eyes and foramen magnum due to the higher conductivity there. The streamlines use tangential coloring. This coloring can make the local orientation of each point of the streamline in three dimensions more visible, without the need to rotate the scene. For the positions and orientations of the dipoles, see Figure 5.12.*

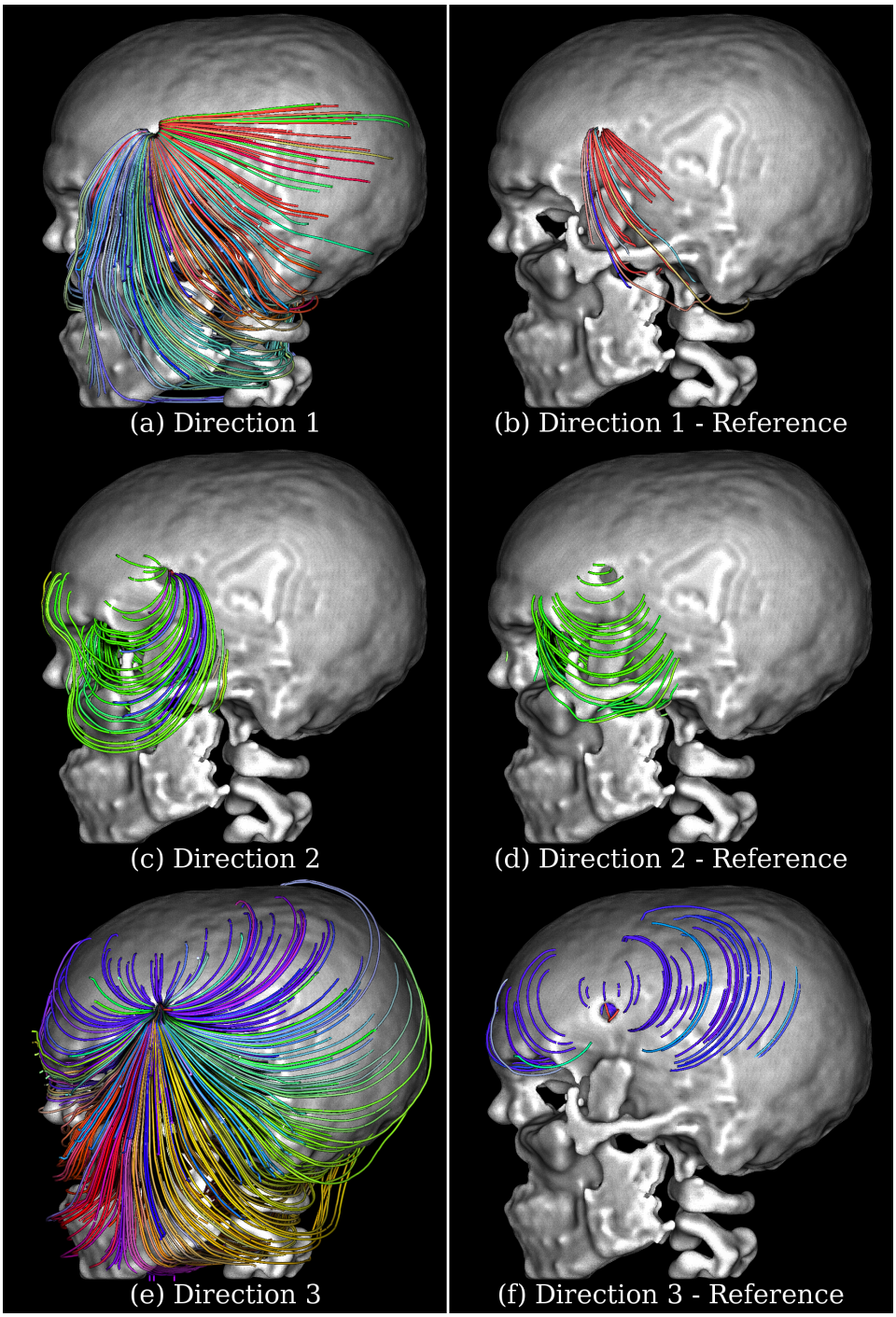
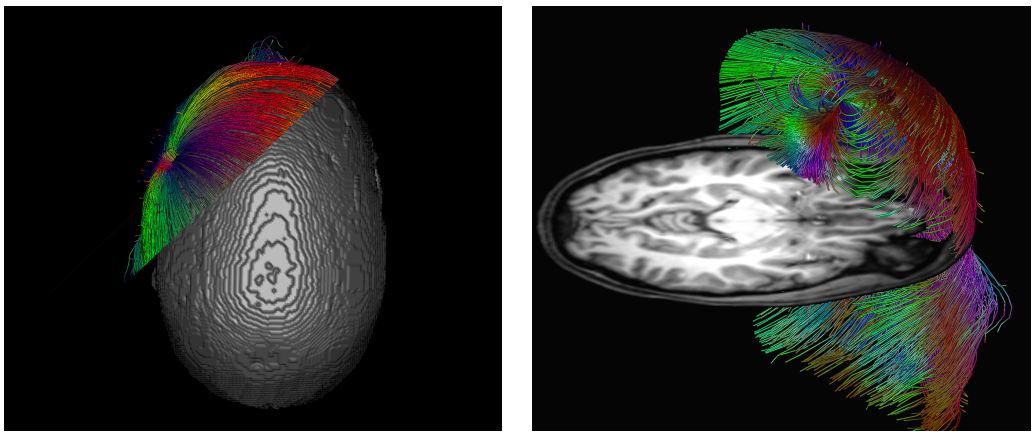


Figure 5.7: *Streamlines depict differences of the electrical flow fields. Direct comparison of the directionally (global) colored streamlines computed for each source orientation and both models: reference model (without hole) on the right and Skull-Hole-Model on the left. Shown are those streamlines, which are running through the skull hole (also for the reference case without the hole). Unlike Figure 5.6, this figure shows that the field of the radial source is also influenced by the skull hole. However, Direction 3 is most strongly influenced, which can also be seen in direct volume rendering results (Figure 5.5). In comparison to the previously described methods, this technique offers a detailed view. For the positions and orientations of the dipoles, see Figure 5.12. The skull mask is a binary mask that has been processed in an automatic pipeline. It is of suboptimal quality and contains several processing artifacts. Mesh refinemend and smoothing does not help to get rid of the artifacts.*



(a) (b)

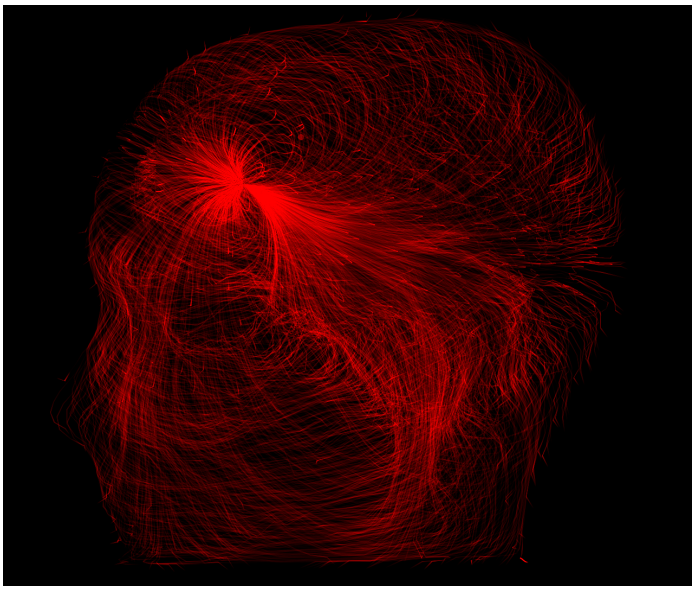
Figure 5.8: *Clipping planes used for streamlines with anatomical context in the Skull-Hole-Model. The plane is placed in the radially oriented source. With such a clipping plane (or a combination of planes), it is possible to select a certain fraction of the streamlines. Part (a) shows a top view of the applied clipping plane. As the isosurface prohibits the direct view onto the dipole, it is often more useful to combine interactive selection tools with orthogonal anatomy slices for orientation. In (b), such an axial slice helps to improve orientation and allows an unhindered view to the dipole. For the positions and orientations of the dipoles, see Figure 5.12.*

If not stated otherwise, the streamlines are colored according to their local tangent direction.

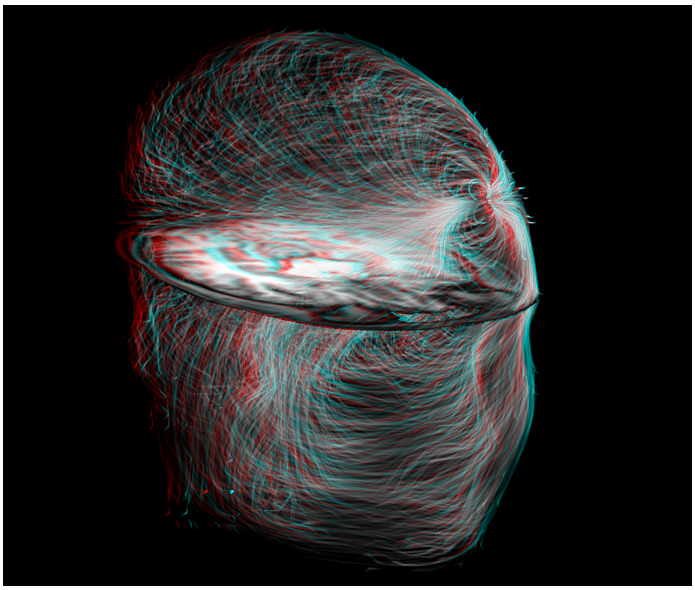
Comparability - A quantitative comparison between several streamlines is not reasonably possible. In Figures 5.6 and 5.7, global differences in streamlines generated from different models can be judged subjectively by the user. The user can directly compare density, orientation, and number of streamlines among several images. For a comparison, it is important to provide the same coloring and value ranges for colormaps throughout the models.

Anatomical Context - Embedding of anatomical context with streamlines can be a problem. In very dense areas near the source (or in deeper brain regions), occlusion becomes a serious problem and can prohibit the direct sight to anatomy. This problem can be solved to a certain degree by utilizing clipping surfaces or transparency, such as in Figures 5.8 and 5.9.

Interactivity - The streamline calculation process itself cannot be performed in real-time. However, rendering large numbers of precomputed streamlines is possible in real-time. The selection and coloring using transfer functions can also be done interactively, which is required for efficient exploration of the data, with the possibility to display details on demand.



(a) Curvature only



(b) With anatomy in 3D

Figure 5.9: *Perception of streamlines in 3D. Electrical flow field of the Skull-Hole-Model in combination with transparency and a curvature-based transfer function. The transparency, which is defined by the line curvature at each point, highlights the shape of the electric field deeper inside the brain. Curvature is a common measure to describe how much a streamline deviates from being straight. In (a), no anatomy is provided, rendering spatial relations difficult to see. Due to the missing depth cue, these types of renderings are useful only if the viewer interacts with the scene, allowing perception of spatial relations and structure of the field inside the head. The image shown in (b) uses stereoscopic (anaglyph three dimensional) rendering to add a spatial cue and, thus, allows perceiving the spatial relation of the field structures towards a given anatomical cue. For the positions and orientations of the dipoles, see Figure 5.12.*

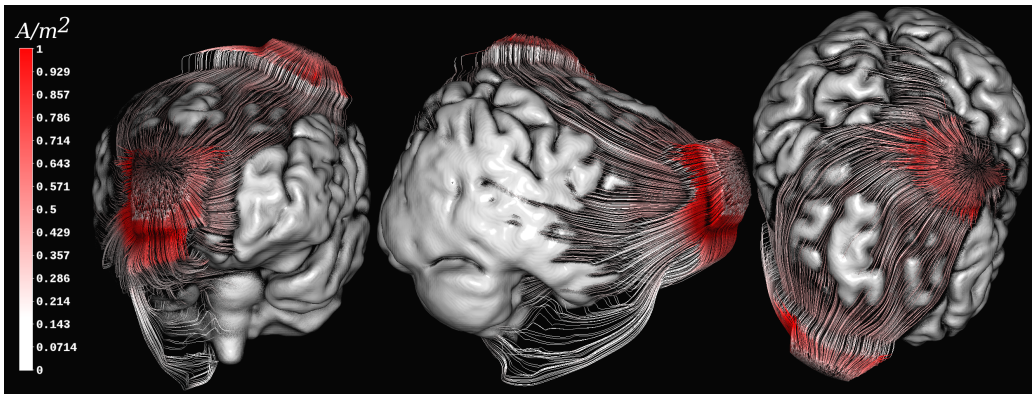


Figure 5.10: *Streamlines through volume conductor. Streamlines show results of a tDCS simulation with respect to the brain surface while using a colormap to encode current density magnitude (white-to-red colormap,* $[0, 1]$ A/m^2*).*

Skull-Hole-Model In Figure 5.6, the streamline tracking results are shown for all three source orientations. Further, all streamlines outside the skull (mainly in skin tissue) are running more or less tangentially to the skin surface. For different source orientations, the impact of the skull hole is very different. For Direction 3, the impact of the skull hole is most apparent since a huge number of streamlines are passing through it. This result is quite interesting, because Direction 3 is a tangentially-oriented source, which, however, has a relatively large component pointing towards the center of the hole. The source is located slightly superior and anterior to the skull hole (see radially oriented Direction 1 for reference). Furthermore, besides the impact of the skull hole, some other effects are visible. Firstly, the high tissue conductivity of the eyes evidently diverts some of the streamlines (i.e., electrical current) and makes them pass through the natural skull openings (e.g., for optical nerves) at these locations. Secondly, a similar behavior is apparent at the foramen magnum. This behavior is generally expected at locations where the skull is not closed or a conductivity bridge (through low-conductance skull tissue) can be established, for example at surgery holes, sutures, etc.

tDCS Figure 5.10 displays streamline tracking results in relation to the brain surface. The depicted streamlines indicate that electrical current enters the skull tissue radially close to the injecting electrodes. As in the Skull-Hole-Model, the streamlines are strongly bent when flowing through a natural skull opening (foramen magnum).

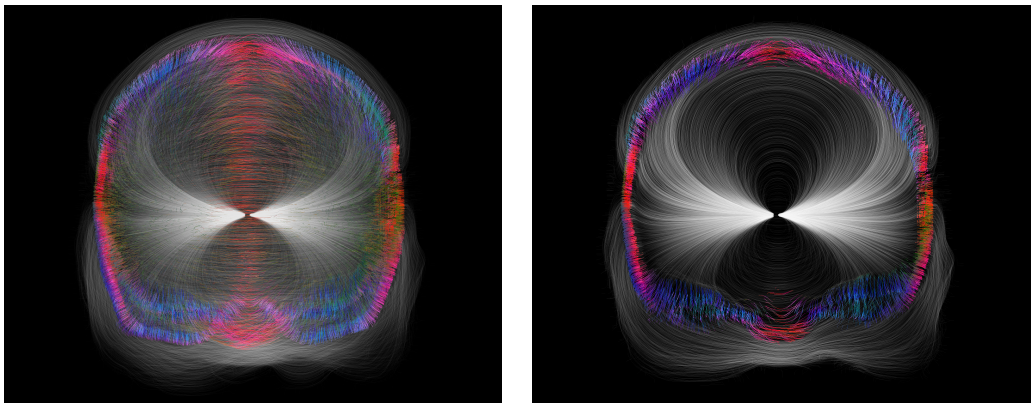
Streamline Selection and Clipping

Skull-Hole-Model In Figure 5.7, the particular effect of the skull hole was investigated by visualizing streamlines running through the hole (or the site of the hole for the reference model). A ROI box was used that approximately covers the hole (shown in cyan), thus selecting only streamlines that actually pass through the hole. For comparison, the streamlines for the reference model regarding the same source orientation are depicted. It can be seen that, for all three source directions (Direction 1, Direction 2, and Direction 3), there appears to be a clear difference in volume conduction. With respect to the absence of the skull hole, the number of the outgoing streamlines in the reference model is much smaller. Again, the biggest difference between the models can be seen for Direction 3. In Figure 5.8, another selection tool, the clipping plane approach, is shown. With such a clipping plane or a combination of planes, it is possible to select a certain fraction of the streamlines. In combination with anatomical slices, interesting areas, e.g., the source singularity, can be investigated more precisely.

tDCS In tDCS, the streamline algorithm always creates streamlines starting and ending at the injecting electrodes, independently from where the seed points were placed. This means that a ROI box covering the motor area underneath the anodal electrode (C3) selects the majority of streamlines running through the target brain tissue (motor cortex) as shown in in Figure 5.10. The advantage of using the selection tool is to exclude those streamlines, which run through the skin and, thus, would otherwise occlude the view onto the much more interesting streamlines through the target region. All in all, the tDCS and the Skull-Hole-Model share the same advantages and disadvantages for the respective methods. In both examples, streamlines are adequate for showing the global structure of the electrical field, but are limited when it comes to local details.

Local Opacity and Coloring

Skull-Hole-Model In Figure 5.9, a curvature-based transfer function in combination with the streamline approach is shown. The curvature-based rendering accentuates areas with high streamline curvature, which correspond to tissue conductivity jumps or gradients based on large differences in potentials of adjacent nodes. This rendering makes it possible to see interesting details (such



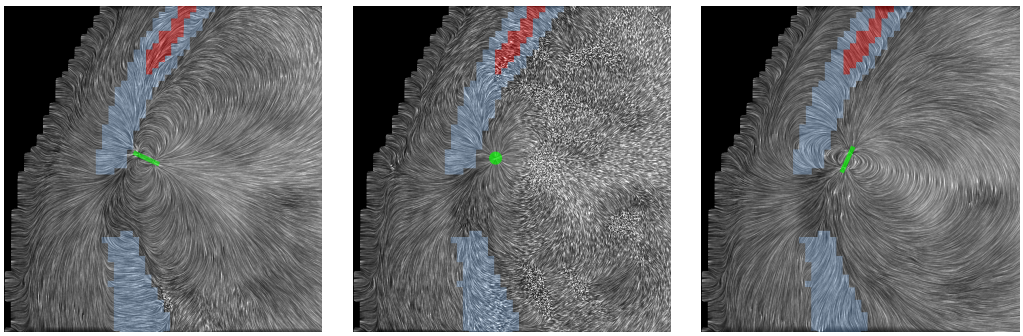
(a) (b)

Figure 5.11: *Streamlines through the 3-Layer-Model data. Coronal view of the 3-Layer-Model, where a source is placed near the thalamus. The streamlines were made opaque inside the skull and slightly transparent elsewhere. The color of each streamline inside the skull reflects its local direction (tangential coloring). Due to the coloring inside the skull, the field lines clearly undergo different degrees of diversion, depending on the angle at which they enter the skull: (a) volumetric rendering of all streamlines, (b) rendering of streamlines within a slab (thickness* 10 *mm) around a coronal slice passing through the thalamus, which removes the occlusion problem and unveils the streamline structure inside the slab.*

as the mainly affected streamlines) inside the model without the need of explicitly selecting them. It is important to note that the full benefit of this technique is only achieved in combination with modern display techniques, such as interactivity (the user can turn around the object in real-time) and 3D display using modern display devices (see Figure 5.9(b)).

3-Layer-Model Figure 5.11 shows a streamline rendering of a source located in the human thalamus. Since all areas inside the skull are modeled isotropically (with a brain conductivity of $\sigma_{brain} = 0.33\ S/m$), the streamlines are smooth (due to the absence of conductivity jumps). However, the skull is modeled inhomogeneously as done by Dannhauer et al. [35] with much lower conductivities for soft and hard bone, as compared to isotropic skin and brain conductivity. Therefore, the streamlines, being representations of the electrical current, are bent at boundaries between tissues.

tDCS In Figure 5.10, generated streamlines are colored with current density magnitudes using a white to red colormap. Clearly, the corners of the electrode sponges touching the skin surface have the highest current densities. The current densities inside the skull are significantly smaller compared to the



(a) Direction 1 (radial) (b) Direction 2 (first tangential) (c) Direction 3 (second tangential)

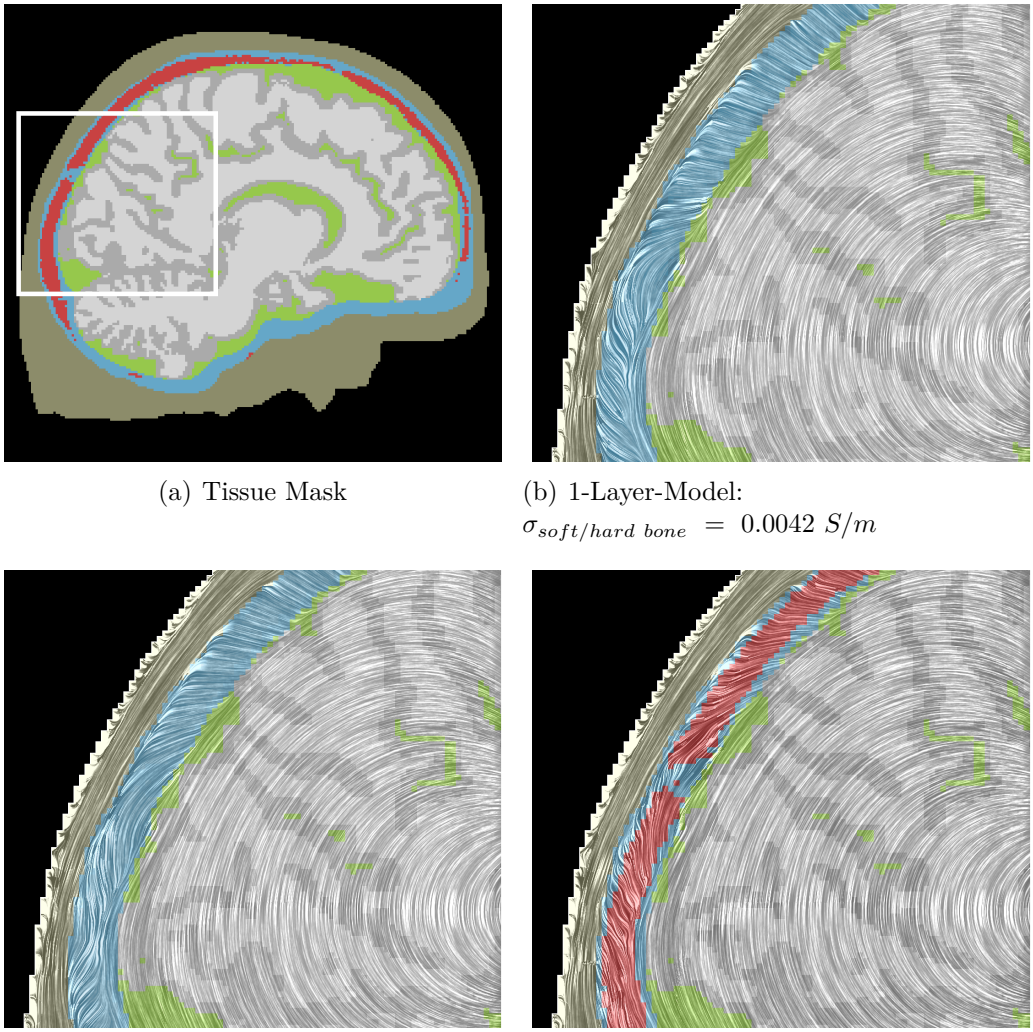
Figure 5.12: *Line Integral Convolution (LIC) images on a coronal slice through the hole combined with the electric fields for all three source orientations in the Skull-Hole-Model. Small differences in current flow between the source orientations can be seen. However, a direct quantitative comparison is not reasonably possible with LIC. The green bar in each image indicates the source orientation and position. The soft bone tissue is colored in red, the hard bone tissue in blue, and the remaining head tissues in gray.*

skin. However, current densities magnitudes appear to be higher in CSF even though they are more distant from current injecting sites most likely, because of the high conductivity ($\sigma_{CSF} = 1.79\ S/m$) of CSF compared to surrounding materials. Similar to the Skull-Hole example, a curvature-based coloring of the streamlines would be possible. This coloring could help to find conductivity bridges indicating problems that have been overlooked during tissue segmentation.

5.4.3 Line Integral Convolution

We applied LIC to all three application cases (see Figures 5.12, 5.13 and 5.14) on orthogonally oriented slices. A skull mask was used as a colormap for the Skull-Hole-Model (see Figure 5.12). Furthermore, we combined tissue masks (from tissue segmentation) as an additional colormap for the different bone layer models (see Figure 5.13) and tDCS-Model (see Figure 5.14). Additionally, we applied LIC on the surface of the skull and the brain in the tDCS example to demonstrate the possibilities and problems of surface LIC (Figures 5.15 and 5.16).

Comparability - LIC provides a global overview of the electrical field as well as specific local details. Both aspects can be compared between models and to other visualization techniques. Unlike colormapping, quantitative comparisons with LIC are not reasonably possible - only the local direction



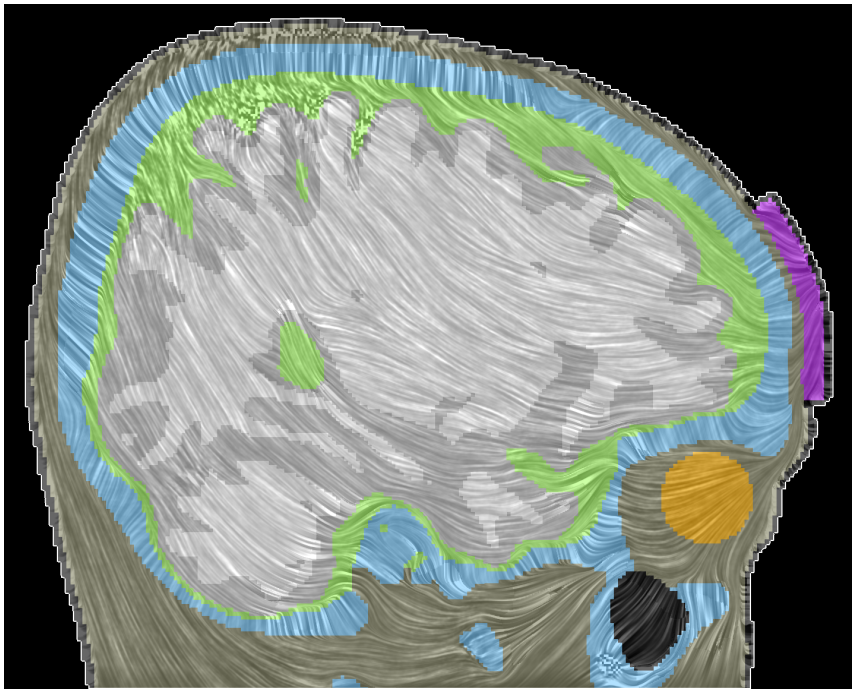
(a) Tissue Mask

(b) 1-Layer-Model:
$\sigma_{soft/hard\ bone} = 0.0042\ S/m$

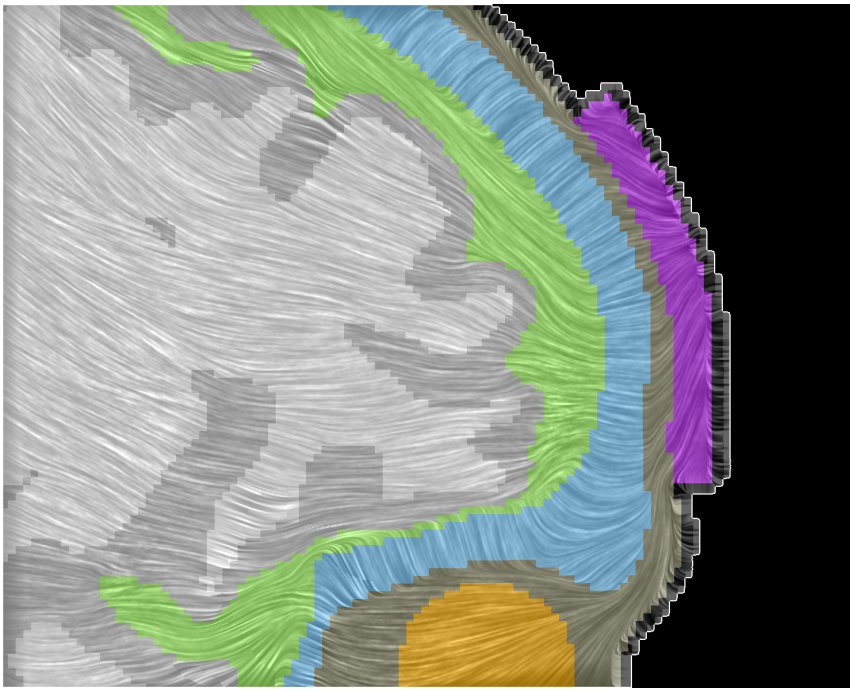
(c) 1-Layer-Model:
$\sigma_{soft/hard\ bone} = 0.01245\ S/m$

(d) 3-Layer-Model:
$\sigma_{soft\ bone} = 0.0287\ S/m$ and
$\sigma_{hard\ bone} = 0.0064\ S/m$

Figure 5.13: *Line Integral Convolution (LIC) for the 1- and 3-Layer-Model. In (a), the different tissue types are visualized (skin in beige, CSF in green, gray matter in gray, white matter in light-gray, hard bone in blue, and soft bone tissue red). CSF, gray and white matter are modeled electrically using an isotropic conductivity of* $0.33\ S/m$. *The zoomed images use LIC to show the influence of the occipital fontanel regarding the electric flow field, for different values and bone conductivity models. The source is located in the thalamus for the 3-Layer-Model. (c) shows the best matching isotropic model, which seems to be a good approximation of the 3-Layer-Model (d).*

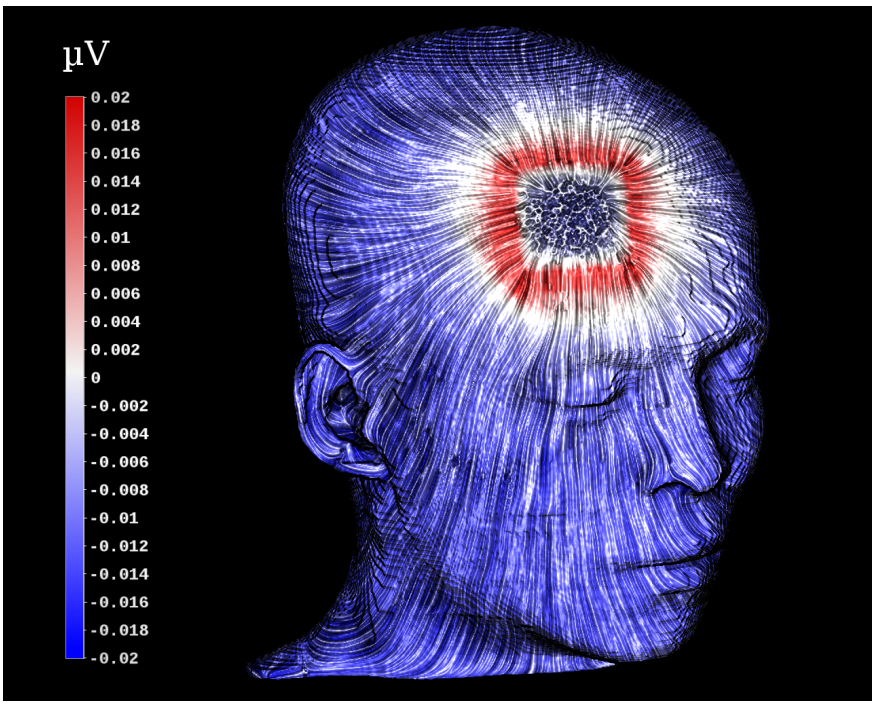


(a) Sagittal Slice

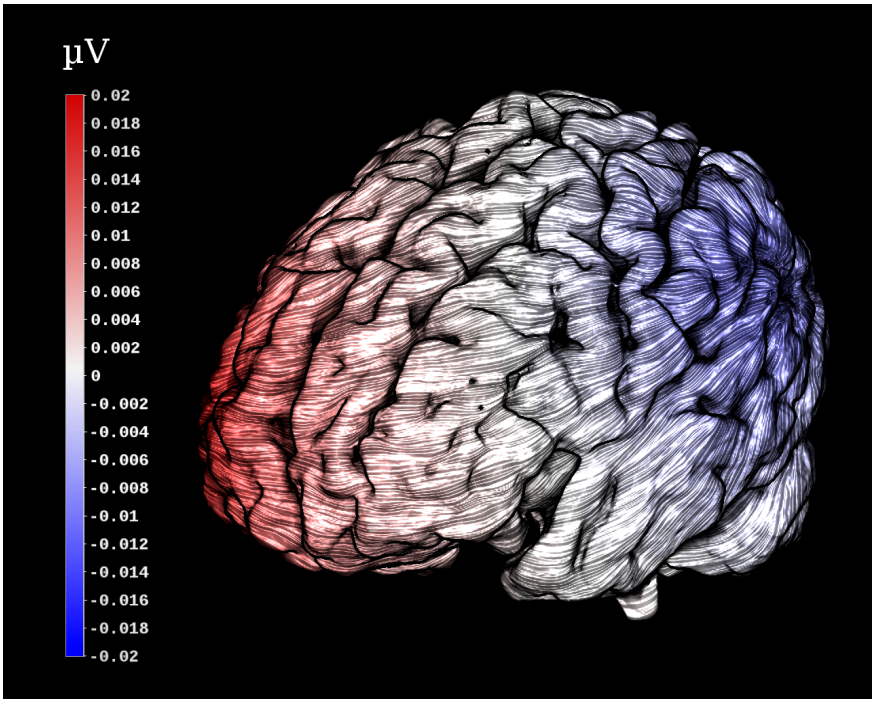


(b) Zoomed In

Figure 5.14: *Line Integral Convolution (LIC) images mapped on a sagittal slice (right panel: zoomed) through volume conductor shows results of a tDCS simulation in combination with a colored background based on tissue labels. The different tissue types are visualized using a colormap similar to the one in Figure 5.13: skin in beige, CSF in green, gray matter in gray, white matter in light-gray, skull in blue, the eyeball in yellow, and the tDCS electrode sponge in purple.*

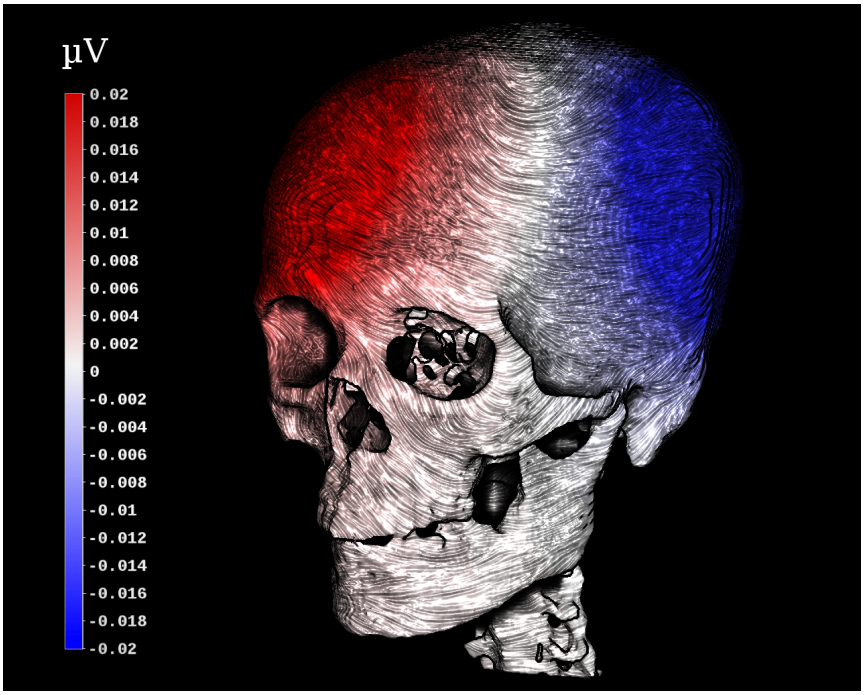


(a) tDCS field on the skin.

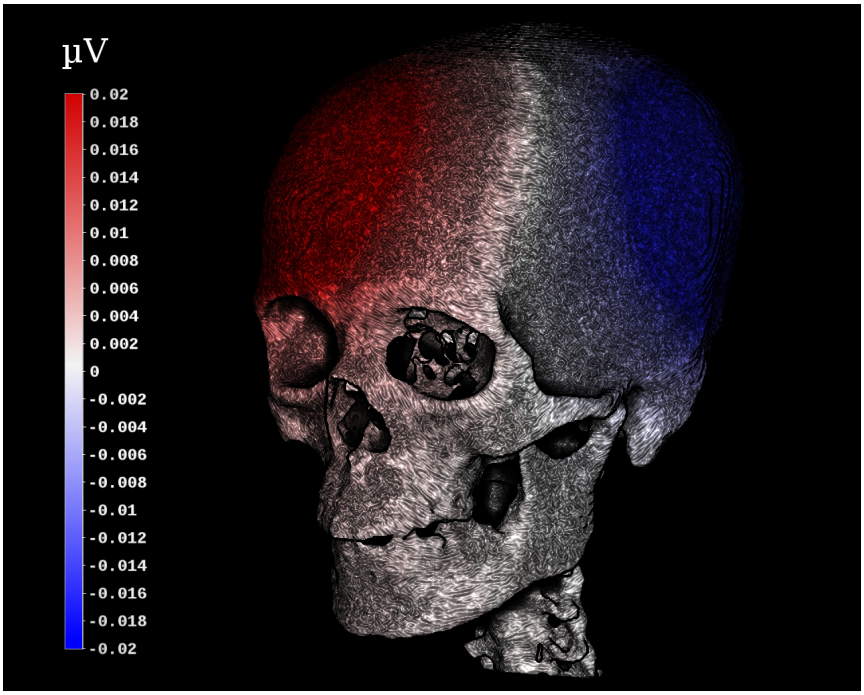


(b) tDCS field on the brain.

Figure 5.15: *Line Integral Convolution (LIC) and colormapping of the potential field on the skin and brain masks. Both images show the possibilities of LIC in surfaces. Especially interesting are anatomical surfaces, whose conductivity profiles modify the behaviour of the field. The top image (a) shows that the field enters the skin nearly perpendicular below the tDCS sponge. It flows around the brain (in the cerebrospinal fluid), as depicted in (b).*



(a) Projection of nearly perpendicular vectors.



(b) Cutoff angle of 45 degree.

Figure 5.16: *Line Integral Convolution (LIC) and colormapping of the potential field on the skull mask with two different cutoffs for the vector projection. In (a), even nearly perpendicular vectors are projected to the surface. In (b), vectors with an angle towards the surface of more than 45 degree were removed. The difference between both images is obvious and demonstrates the influence of visualization parameters on the interpretation and expressiveness. (a) shows electrical flow, which is not there at all, as seen in (b).*

of the current flow can be inspected qualitatively. In combination with colormaps, comparability can be enhanced, since colormaps allow the combination of the flow direction with other details (such as the strength of local potential changes). In terms of visibility, the contrast between colormaps and LIC may be a limiting factor. Moreover, LIC textures modify color intensities, which can lead to misinterpretation of the colormap. Again, similar to the other methods, it is important to make sure that the same algorithm parameters are used throughout the whole series for comparison.

Anatomical Context - As mentioned above, a combination of colormaps with LIC is possible (although not without limitations). Another option is to use geometric information derived from anatomical data (isosurfaces) for LIC. The LIC effect can be applied to the surface, serving as an anatomical cue and can easily be combined with orthogonal slices showing the anatomy. Figures 5.15 and 5.16 show this for the tDCS electrical field data on the skin, skull, and brain. Figure 5.16 also shows a downside of surface-based LIC. The projection of the electrical flow vectors to the surface needs to be cutoff for a certain vector-to-surface angle. When choosing a too steep angle, the projection of the vector yields questionable results. This is, again, caused by the inability to represent the length of the vector as quantitative information in the Schlieren-pattern.

Interactivity - Usually, the standard LIC implementation is too slow for interactive modification and exploration. In contrast, our GPU-based approach does allow rendering at interactive frame rates.

Skull-Hole-Model In Figure 5.12, LIC textures are shown on a coronal slice, for a source near the skull hole (see mask), for the three current directions (Direction 1, Direction 2, Direction 3). It can be seen in all three LIC images that some currents flow through the skull hole. However, as mentioned above, this effect cannot be quantified. The seemingly "noisy" parts of the texture indicate flow directions perpendicular to the depicted slice.

3-Layer-Model Figure 5.13 shows LIC results for the different ways of skull modeling. The figure shows the area around the occipital suture, whereas the only difference between Figures 5.13(b)-5.13(d) is the applied conductivity profile of the skull. In Figure 5.13(b), the skull is modeled with the traditionally used isotropic conductivity ($\sigma_{hard/soft\ bone} = 0.0042\ S/m$). Previous work [35] showed that the isotropic conductivity must be much higher in a realistic setting (Figure 5.13(c)). For that model, the isotropic conductivity was fitted

(see section 5.3.2) for more details) to the 3-Layer-Model, yielding an isotropic value of $\sigma_{hard/soft\ bone} = 0.01245\ S/m$. In Figure 5.13(d), the LIC result for the reference model is shown. The reference model uses experimentally measured conductivities for soft (red) and hard bone (blue). Soft and hard bone distribution was estimated by skull segmentation based on a T1-weighted MR image. The LIC approach allows detailed insight into flow features and structures inside the differently modeled bones and emphasizes their difference. It can be seen that Figures 5.13(c) and 5.13(d) are much more similar than Figures 5.13(b) and 5.13(d). Furthermore, LIC streamlines that are, due to the presence of soft bone, diverted tangentially with respect to the skull surface, can be clearly identified (Figure 5.13(d) compared to Figure 5.13(c)). For more details about the approximation of the three-layered skull structure using a global isotropic conductivity model please refer to Dannhauer et al. [35].

tDCS Figure 5.14 depicts LIC streamlines of a sagittal slice passing through the frontal electrode (Fp2) combined with a colormap helping to perceive material boundaries. Similar to Figure 5.13 and more detailed as in Figure 5.10 the dominance of a radially-oriented electrical current is strikingly apparent.

Wagner et al. [210] investigated the impact of homogeneous and inhomogeneous skull modeling for tDCS in which they varied conductivity ratios of soft and hard bone within ranges that were experimentally determined as described by Akhtari et al. [3]. They depicted the results as cones having normalized length. Based on their visualizations they concluded that currents mainly flow radially through isotropically modeled skull tissue. Their investigations contained inhomogeneous skull models in which they stepwise increased the hard-to-soft bone conductivity ratio (nominally soft bone conductivity) from averaged [3] to ratios that led to mainly tangential current flow within soft bone structures. They claimed that for higher hard-to-soft bone conductivity ratios their chosen target regions were significantly affected by those changes, depending on their location. Additionally, they used similar cone plots to investigate changes in current flow direction in the case of including CSF, differentiating between brain tissues (gray and white matter) in the volume conductor model and using color maps to point out the impact of white matter conductivity anisotropy. Our results confirm the results reported by Wagner et al. [210] for tDCS but also for EEG as shown in Figure 5.13 and 5.14, respectively. Figure 5.15 further substantiates this. The field enters the skin nearly perpen-

dicularly below the tDCS sponge, while Figure 5.15(b) shows the tangential flow around the brain surface.

5.5 Future Work and Conclusion

In the previous sections, we have ***highlighted advantages and disadvantages of several standard visualization techniques*** exemplarily for three interesting models regarding the influence of the human skull and tDCS stimulation on bioelectric field simulations. We used visualization methods to create an intuitive understanding of volume conduction effects, which otherwise can be described only in a rather counter-intuitive way by numerical measures [35]. Most importantly, ***we assessed all algorithms*** in all examples with respect to clearly defined criteria: (1) the quantitative comparability between datasets, (2) the possibility to provide anatomical context, and (3) the feasibility of interactive use. In particular, the latter point is often underestimated in the written literature with its unavoidably static images. The possibility to interactively change parameters or to rotate the image in three dimensions can often provide more insight than very sophisticated renderings trying to pack as much information as possible into static images. In our work, we did not go into much detail for this point, since we ensured that each used method works interactively, by utilizing the vast computational power of today's graphics hardware. But of course, this is not necessarily possible in general.

Isosurfaces and Direct Volume Renderings provide a quick overview of the data and the influence of anatomical structures on the field propagation. These methods were especially fruitful for the visualization of global features of the field in the Skull-Hole-Model. The local features of the 3-Layer-Model could not be sufficiently captured. In the chosen tDCS example, isosurfaces were especially helpful to visualize the current density magnitude on anatomical structures. Unfortunately, DVR suffers from the problem of complicated and time-consuming designs of useful transfer functions and hence is the subject of further research.

The visualization using streamlines provides more detail on the structure of the actual electrical field, especially the influence of the skull hole and current flow properties in tDCS stimulation, which can be seen very clearly together with filter and selection tools. The selection mechanisms allow for simple exploration and comparison of the field in conjunction with anatomy and model-specific regions. As with DVR and Isosurfaces, the prime benefit

of this method is the exploration of features within a global scope. For the 3-Layer-Model, local effects are hard to interpret with streamlines as the interesting areas are small and cluttered inside the skull tissue. The same is true for the tDCS example, where dense streamlines occlude the more interesting, local stream features in certain tissue types. Selection mechanisms can help to filter out uninteresting streamlines to avoid intense visual clutter.

Finally, LIC proved ideal for exploring the interesting local details in the 3-Layer-Model and tDCS. It provides a qualitative explanation for local effects of different skull models and their statistically measured similarities and dissimilarities. Unfortunately, quantification is difficult with LIC. Especially for the Skull-Hole-Model and tDCS, the combination of LIC with colormaps is difficult, as LIC directly influences the brightness of the underlying colormap, which can lead to misinterpretation. LIC is an interesting option, as it provides local details otherwise invisible with streamlines. Its limitation to surfaces and slices prohibits the fast volumetric perception of the field. Volumetric LIC (3D-LIC, [53]) methods could help if a proper importance-function could be defined, which might be difficult and very application-dependent. We have shown the influence of parameter selection on the resulting images using a surface LIC. A parameter is used as criterion for deciding whether the vector still contains valuable information *on* the surface or not. Generally, parameter tuning is a necessary evil in a lot of visualization techniques. Parameters allow flexibility, but at a cost.

Altogether, ***visualization provides a tremendous insight*** into volume conduction and helps to ***understand the underlying models*** and the influence of their parameters. Visualization allows to qualitatively explain features in bioelectric fields, even if they are only indirectly detectable using quantitative error measures. A myriad of visualization techniques is available, all with their own benefits and drawbacks. The selection of the proper method mainly ***depends on the specific application*** and the kind of features that need to be explored. In addition, neuroscience and other life sciences have very specific visualization requirements. Besides the three main requirements postulated in this work (comparability, context, and interactivity), acceptance of a method mainly depends upon its ability to reveal information and to allow its intuitive interpretation. We found that an interactive, intuitive, and adapted tool is often more important than nice-looking images, created with methods that require multiple parameters. The latter often lead to error-prone methods, requiring a great deal of manual fine-tuning. Even if they provide

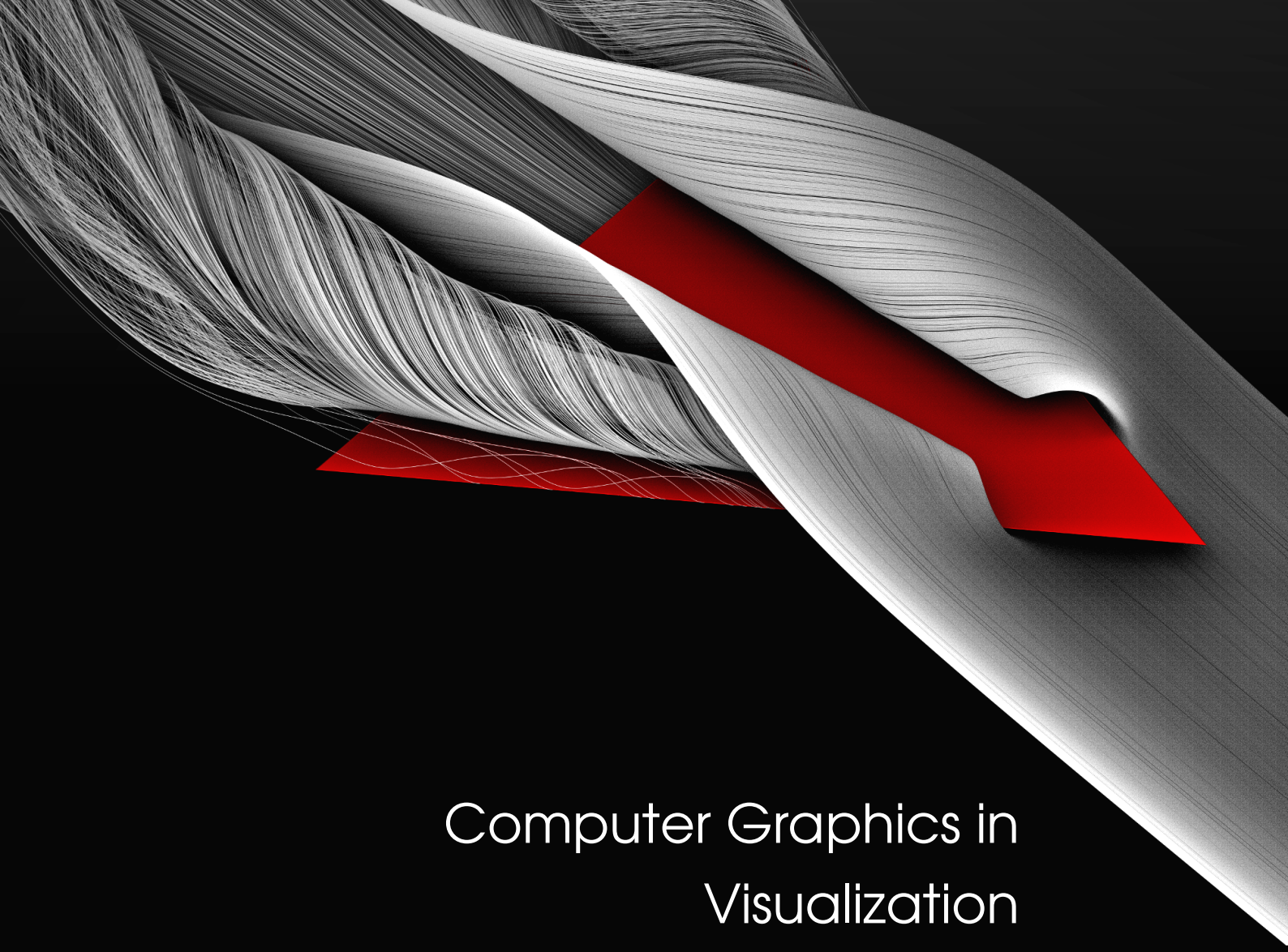
subjectively impressive images, they do not necessarily transport the needed information. Table 5.1 gives an overview on the general advantages and disadvantages of the methods used in this chapter. The actual value of a method heavily depends on the domain and the features to investigate.

5.5.1 Future Work

Future directions of this type of application-specific visualization research should involve experimental and clinical validation. In this context, other neuroscientific techniques and aspects of volume conduction might be interesting to explore such as induced neuronal activity by transcranial magnetic stimulation (TMS), reconstruction of current flow measured by intracranial EEG (iEEG), and modeling the specific volume conductor properties, e.g., skull modeling in children [107]. In general, we aim at more application-specific techniques, including automated transfer function design and estimation of parameters from the data. In our experience, a lot of parameters of the used visualization techniques can be estimated automatically, rather than asking the user for their exact value. This would provide the scientists with useful defaults, but still allows fine-tuning the visualization.

Pros	Cons
Isosurfaces	
• Insights into spatial distribution of scalar fields. • Easy embedding of anatomical context.	• Only shows a part of volumetric structure (choose isovalue properly; consider meaning of "volume" and "distance" in renderings). • Prone to noise and sampling artifacts.
Most useful in the context of selectively showing global features and behavior.	
Direct Volume Rendering (DVR)	
• Insights into spatial structure and distribution of scalar fields in the entire volume. • Avoids occlusion problems.	• Transfer function (TF) design is very domain- and case-specific. • Anatomical context is hard to embed.
Most useful in the context of catching multiple, global features in the entire volume.	
Streamlines	
• Insights into directional structures at globally in 3D	• Occlusion problem (partially solvable by transfer functions and line filters).
Most useful in the context of grasping major directional structures in 3D.	
Line Integral Convolution (LIC)	
• Insight into directional structures locally (focus on details). • Good qualitative comparison among multiple images.	• Only depicts directional information; quantification difficult. • Combination with colormaps can lead to misinterpretation.
Most useful in the context of analyzing local and small-scale directional structures.	

Table 5.1: *Comparison of the general advantages and disadvantages of the shown visualization methods.*



Computer Graphics in Visualization

In the last decade, a multitude of computer graphics methods were developed to improve plasticity and realism of computer generated images. These methods aim at a physically accurate replication of the real world. Especially the computer game industry pushes the development of techniques that achieve this ambitious goal phenomenologically.

Unfortunately, these developments attract only limited attention in visualization. This might be due to the common notion of "*A visualization is not a computer game. There is no need for pretty looking images.*" or simply because most of these computer game methods cannot be applied to a typical visualization scene directly. Most of these methods rely on high-quality triangle meshes, precalculated normal maps or simply handle transformational effect-coherency in a rather sloppy way; things that are different in visualization. However, improved structural perception, improved spatiality, and better visual detection of relations in the data are only some of the advantages one gets when reasonably utilizing modern computer graphics in visualization.

To achieve this, one has to adapt and re-invent computer graphics methods towards the specific needs of visualization and the specific characteristics of a given rendering/visualization method. In this part, three methods are shown, which contribute to this rather sparsely investigated area of visualization and provide tremendous perceptional improvements to existing and commonly used visualization techniques. These improvements help domain scientists to grasp local details as well as global structures and relations in their data, which is crucial to understand complex, three-dimensional scientific data.

Part II

6 Background

Visualization is a very complex process. It involves substantial processing to extract information and features from a raw column of numbers, which is incomprehensible by the human brain. Regardless of the specific visualization technique, it always begins with input data and it always ends with a graphical representation – two cornerstones of visualization. The graphical representation serves as a transfer-medium between the data and the mental image of the data in a viewer's brain – and computer graphics is the driver. Hence, the importance of computer graphics (CG) in visualization cannot be ranked high enough.

In recent years, CG methods evolved that aim at a maximum of realism and plasticity in computer generated images. This development was mainly driven by the computer game industry, but found only limited attention in visualization. Thereby, modern computer graphics methods can tremendously improve the perception of space and structure in still images; something that is a major demand in visualization.

In this part, I will introduce our work on computer graphics methods to improve the visual quality, structural, and spatial perception in visualization. To understand these techniques, it is crucial to have a basic understanding of the modern graphics pipeline and the screen space rendering approach.

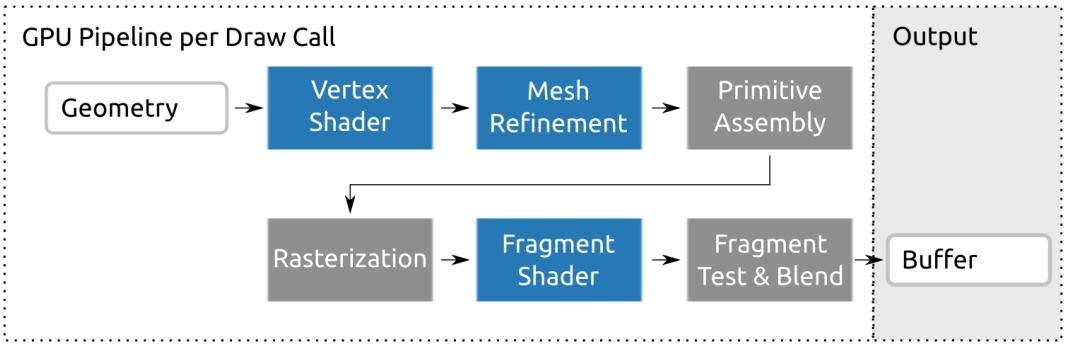


Figure 6.1: *The OpenGL 4 rendering pipeline. The blue boxes denote the freely programmable stages of the pipeline. The gray boxes are defined by the GPU. Conceptionally, the pipeline can be split into two parts. The upper row shows the vertex processing stages and the lower row the fragment processing stages. For the sake of simplicity, the geometry shading stage and the tessellation unit were simplified into "Mesh Refinement". The final fragment is written to the output buffer, which usually is associated with the screen.*

6.1 The Modern Graphics Processor

The architecture and capabilities of graphics processors (GPU – graphics processing unit) has changed tremendously over the last ten to fifteen years. The fixed function pipeline has been loosened up in favor of more and more programmable stages in the rendering pipeline. In this thesis, I completely rely on OpenGL. Hence, the following descriptions are centered about OpenGL and its structure, but can be applied to other graphics interfaces as well.

6.1.1 The Graphics Pipeline

The OpenGL 4 pipeline is shown in Figure 6.1. The rendering software starts with defining the different programs for the programmable stages. These programs are usually called "shaders". The rendering software finally issues a "draw call", as OpenGL calls them. The geometry to render is associated with the draw call. It consists of a list of vertices and a list of primitives, mapping the vertices to the primitives. Additional attributes can be defined too. These attributes are usually texture coordinates or normals, but can be of any type and meaning.

Vertex Shader The first step on the GPU is to issue the *vertex shader*. As the name implies, it processes each vertex of the input geometry. A core concept behind the vertex shader is its strict locality and scope to the vertex level. At this level, the shader has access to the data making up the vertex

it currently processes. It sees its current vertex only, it is not allowed to access neighbouring vertices, and the vertex shader does not even know what kind of primitive the vertex belongs to. Vertex shaders are used to apply arbitrary transformations to the data and usually to finally project the vertex to screen space. They also define the final texture coordinates of the vertex. Texture coordinates are usually not transformed and are given by the rendering software as additional attribute per vertex. They define which part of the bound textures should be mapped to the surface of the primitive. Additionally, these shaders are allowed to do calculations for the vertex and save the results in variables that are, again, associated with the vertex.

Optional Mesh Refinement After the vertices were processed by the vertex shader, the GPU allows to use the tessellation unit and the geometry shader. Although both are different stages, they both allow to increase the amount of vertices and primitives on the GPU with certain limitations. As these stages are not needed in the following chapters, I skip them for the sake of simplicity. For further details, please refer to the "OpenGL Red Book" [185].

Primitive Assembly After the vertex shader or the optional refinement stages have finished and created the projected vertices, the GPU automatically reconstructs the primitive from the stream of vertices and checks their visibility against the camera setup (clipping). The GPU applies the perspective division and transforms the primitives to the viewport coordinates.

Rasterization As all (partially) visible primitives now reside in the coordinate system defined by the output buffer (usually the screen), the rasterizer can do the scan conversion into fragments. These fragments relate to a pixel on the output buffer, but are not yet known to be visible. They might be overwritten by a primitive in front of the current one. This is the reason why they are called fragments instead of pixels. The rasterizer also interpolates all variables associated with the vertices on the rasterized surface. These variables can be set by the vertex shader. This is the usual way to forward-communicate from the vertex shader to the fragment shader.

At this point, the pipeline has left the vertex processing part of the pipeline. The next steps are centered around the processing of the fragments created by the rasterizer.

Fragment Shader The fragment shader is finally called for each generated fragment. Similar to the vertex shader, it has to obey to the locality and scope rules of the GPU. This means, the fragment shader has no read-access to any of the vertices or mesh data. It cannot read information of neighbouring fragments and is only allowed to write to its own location in the output buffer. However, the fragment shader has global read-access to the textures bound to the currently processed draw call. After the rasterizer created the fragments, the GPU loads the bound textures into local memory of the fragment shader. The GPU knows the exact part of the textures to load, since the rasterizer interpolated the texture coordinates for the fragment already. Fragment programs are usually used to define color and depth of a fragment.

Fragment Testing and Blending As the fragment shader has written the color and other optional information, the GPU runs several tests on the fragment. These tests define whether it is visible or not. Typically, this includes the depth test. The depth of a fragment is compared to a depth buffer. If the fragment is in front of the previous fragment, if any, it is written to the output buffer and mixed with the previous color in the output buffer, if blending is active. The depth of the new fragment is written to the depth buffer for later comparison with other fragments at this specific pixel position.

Pipeline Summary Of course, the graphics pipeline is much more complex, but the description given here is detailed enough to understand the following chapters and the concepts of the modern graphics pipeline. I intentionally left out several details, like compute shaders or the possibility to loosen up the locality constraint, using OpenGL extensions to read/write locations other than the shader's own. These details are not needed here and would bloat up the descriptions tremendously. The "OpenGL Red Book" [185] provides more and detailed information if needed.

It is important to understand that the GPU allows to program different stages of the rendering pipeline and that these programs work locally and strictly scoped. A vertex shader is called for a specific vertex and only knows about this vertex and the data associated with it. Similarly, the fragment shader has no access to neighbouring fragments, nor is it allowed to write to a different pixel location in the output buffer. The vertex shader can output variables for a vertex, which are automatically interpolated between the vertices making up the rasterized surface. The interpolated values are available to the fragment shader.

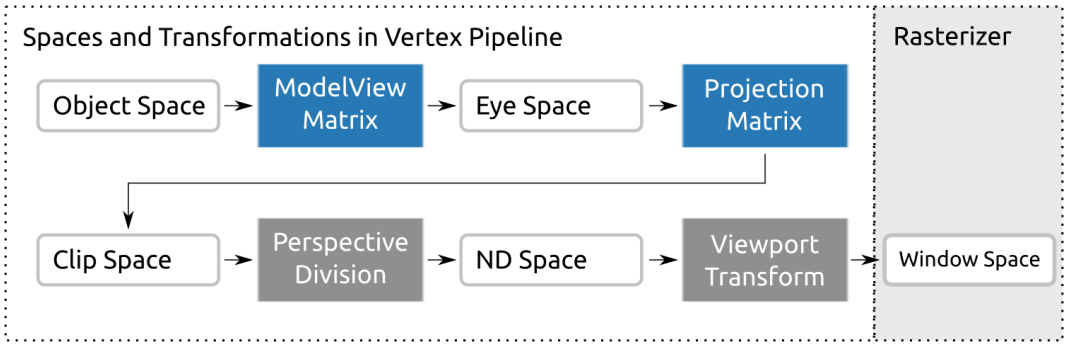


Figure 6.2: *The OpenGL coordinate spaces. The white boxes represent the coordinate spaces and the blue and gray boxes represent the transformation steps. The blue boxes show the programmable part of the transformation pipeline, as they are done by the vertex shader or the geometry shader of the mesh refinement stage. Perspective division and viewport transformation are a fixed function of the GPU. The term ND space abbreviates the "normalized device coordinate space".*

6.1.2 Coordinate Spaces in OpenGL

To understand the screen space principle it is important to first understand the different coordinate systems the OpenGL rendering pipeline operates in. When using the Figure 6.1 as reference, all the transformations to the different coordinate spaces happen in the upper row; the vertex processing part of the pipeline. At this point, it is important to note that all vertex coordinates are given as homogeneous vectors and contain a homogeneous coordinate $w = 1$. All transformations up to the perspective division work with homogeneous coordinates.

Object Space This is the initial coordinate system, where the vertex coordinates reside in. We denote every vertex vector as

$$\begin{pmatrix} x_{object} \\ y_{object} \\ z_{object} \\ w_{object} \end{pmatrix}. \tag{6.1}$$

Eye Space When having a look at Figure 6.1 again, it is clear that the first step after geometry upload is the vertex shader and the mesh refinement stage. As mentioned above, we skip the mesh refinement here and focus on the vertex shader. Its task is to output the clip coordinates of an object space input vertex. This task involves transforming the object space vertex to eye space first. In OpenGL, this is done by multiplying the vector with the *ModelView*

matrix, yielding the eye space coordinates. The ModelView matrix is given by the rendering software and contains the transformations of the object to render and the camera position, rotation and scaling (zoom). The eye space coordinate is then defined as

$$\begin{pmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{pmatrix} = M_{ModelView} \begin{pmatrix} x_{object} \\ y_{object} \\ z_{object} \\ w_{object} \end{pmatrix} = M_{View} M_{Model} \begin{pmatrix} x_{object} \\ y_{object} \\ z_{object} \\ w_{object} \end{pmatrix}. \tag{6.2}$$

OpenGL typically does not differentiate between model transformation and view transformation. Other systems first transform the object space vector to world space using M_{Model} and from world space to eye space using the view transform M_{View}. Often, the terms "world space" and "eye space" are used interchangeably.

Clip Space After the shader program has calculated the eye space coordinates, it calculates the clip space coordinates and passes the result to the GPU. The used matrix is the projection matrix. The rendering software defines the orthographic or perspective viewing frustum and passes the matrix to the GPU. The vertex shader now projects the eye space coordinate as

$$\begin{pmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{pmatrix} = M_{Projection} \begin{pmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{pmatrix}. \tag{6.3}$$

Each component of this vector is in the range $[-w_{clip}, w_{clip}]$ if it is inside the view frustum. That is the reason for calling them "clip coordinates". OpenGL uses them to decide whether a primitive can be removed from the processing stream or not.

Normalized Device Space – ND Space After projecting the eye space vector, the vertex shader passes control over the vertex back to the GPU. The GPU now performs perspective division. This transforms the homogeneous

coordinate back to Cartesian coordinates. This is called de-homogenization and can be expressed by

$$\begin{pmatrix} x_{ND} \\ y_{ND} \\ z_{ND} \end{pmatrix} = \begin{pmatrix} \frac{x_{clip}}{w_{clip}} \\ \frac{y_{clip}}{w_{clip}} \\ \frac{z_{clip}}{w_{clip}} \end{pmatrix}. \tag{6.4}$$

The resulting vector defines a point in the normalized rendering area, thus all components of the vector are in the interval $[-1, 1]$. The normalized rendering area simply is the window where OpenGL renders to, but not yet scaled to pixel coordinates. For example, $x_{ND} = -1/x_{ND} = 1$ refers to the left/right border, $y_{ND} = -1/y_{ND} = -1$ to the bottom/top border, and $z_{ND} = -1/z_{ND} = 1$ to the near and far clipping planes.

Window Space Finally, the normalized coordinates can be transformed to pixel coordinates with respect to the current rendering area. OpenGL calls the rendering area "viewport". It is defined by the origin in pixels (v_x, v_y), a width v_{width} and a height v_{height}. The z_{ND}-component represents the depth of a pixel and can be stored in the depth buffer. The range of the buffer is $[0, 1]$, where 0 is the nearest depth and 1 represents the furthest point. Accordingly, the z-component needs to be translated to this range too. OpenGL allows to define the range as f for the far value and n for the nearest. The final pixel coordinate can then be written as

$$\begin{pmatrix} x_{window} \\ y_{window} \\ z_{window} \end{pmatrix} = \begin{pmatrix} \frac{v_w}{2} x_{ND} + (v_x + \frac{v_w}{2}) \\ \frac{v_w}{2} y_{ND} + (v_y + \frac{v_h}{2}) \\ \frac{f-n}{2} z_{ND} + \frac{f+n}{2} \end{pmatrix}. \tag{6.5}$$

So basically, this is a scaling to the size of the viewport and offsetting it by the origin. This is also true for the depth buffer value z_{window}.

The GPU now rasterizes the primitive by using the window coordinates and the resulting fragments can be processed by the fragment shader. After processing, the GPU can use z_{window} to test whether the fragment is in front or behind a previously rendered pixel at this window coordinate.

Coordinate System Summary Knowing the OpenGL spaces and the general rendering pipeline, one has a solid understanding of how OpenGL renders geometry to the output buffer and how the modern pipeline allows to customize the rendering process on the GPU. The coordinate space in OpenGL can be

transformed in both directions using the provided matrices or their inverses respectively. The whole transformation pipeline up to clip space is programmable by the vertex shader and uses homogeneous coordinates. De-homogenization and scaling to the actual pixels on screen is done by the GPU.

6.2 Screen Space Rendering

Using the above knowledge, this section finally describes the scheme of a whole class of rendering approaches: the screen space techniques. Often, this class of technique is called "image space technique". Both terms are equivalent.

In general, screen space methods stand out as they postprocess three-dimensional scenes in two dimensions. They can be understood as advanced image filters, applied after the original rendering has been done. This is the reason why this class of approaches is usually referred to as "image space" or "screen space".

6.2.1 Concept

Original Render Pass Each screen space approach works as a multi-pass approach. Multi-pass means that the method needs multiple rendering cycles (cf. Section 6.1) to create the final image. Screen space methods usually begin by rendering some arbitrary geometry. A typical example would be an isosurface rendering or a glyph ray-tracer [76]. It does not matter how the method works, but instead of rendering the scene to the screen, it is rendered to an invisible buffer for later use. Typically, this is a texture with the same size as the screen. This first rendering pass yields the color output and the depth of each pixel on screen.

Screen Space Render Pass The rendering software now sends another draw call to the GPU (cf. Section 6.1). Instead of rendering the geometry again, it

- renders a quad at the size of the screen,
- attaches the original output texture to the quad,
- spans the texture to the whole quad, and
- sets a fragment shader.

When reviewing the rendering pipeline and the coordinate spaces again, it gets clear that this way, each texture-pixel (often called "texel") is mapped to its

original pixel on screen. Even more important is the fact that the rasterizer generates fragments that exactly match the mapped texture pixels. These texels are the pixels of the original rendering. This means that the generated fragments exactly match the original pixels on screen, if the rendering would have been done to screen in the first place.

This procedure yields several major advantages:

1. Avoiding fragment locality: The fragment shader has access to the *whole* original result texture. This way, a fragment shader can read neighbouring information, which was not possible during the original rendering pass.
2. Processing visible pixels only: Complexity of screen space approaches is decoupled from data complexity. It only depends on screen resolution and pixel coverage.
3. Cascading render passes: the screen space render passes can output their results to a screen-size texture again. This means, it is possible to cascade multiple offscreen render passes that create different output textures that, in turn, can be used by other screen space passes again.
4. No changes to the original rendering method: The rerouting of outputs can be done using the rendering system only. The original method can be left untouched. This makes it easy to add further processing to arbitrary, already existing methods. No matter how they work, even if they are screen space or multi-pass approaches for themselves.

The last screen space render pass can now render to screen again, instead of a texture. It also writes the original depth values to the depth buffer and thus seamlessly merges with the complete scene.

Data Transfer As indicated above, the transfer of information between the different render passes is done using screen-size textures. The original render pass provides the standard color output as texture and its depth buffer. But usually this is not enough. One requires more information. Maybe the normals on the original surface, tangents or visualized data on the surface, like vectors, tensors, or scalar field information. Unfortunately, this requires a change in the fragment shader of the original rendering method, but is usually not critical or overwhelmingly complex.

As a fragment shader is not limited to writing to one output buffer only, it is possible to attach more output textures at once. This is called “multiple

render targets" in the jargon of OpenGL. It is possible to use single-precision, floating point textures as render target too. This provides a lot of flexibility to transfer additional data to consecutive steps, especially in scientific computing. The fragment shader can write at its specific position to the output textures. This limitation still holds due to its scope and locality limitation.

An important point when transferring vectorial data is to transform it to the right coordinate space before writing it to the output texture. Normals, for example, reside in object space. To use them in the screen space approach, it has to be transformed to screen space too. The question is, which coordinate space is meant here. For vectorial data whose length is not of importance (like normals), this is the clip space – the coordinate space after applying the projection matrix. When perspective scaling is needed, then the Cartesian ND space is used.

In homogeneous projective geometry, vectors can be seen as points on the infinitively far away plane. This means, their homogeneous coordinate is 0. This can be interpreted in another way: vectors are invariant on translation. In GPU programming, this is often used to differentiate points (like vertices) from vectors (like normals). The transformation of vectors to clip or ND space was already introduces in Section 6.1.2.

Pixels and the Texture Space This basically is a matter of terminology. As described, the texture has the same size and resolution as the screen. The texture space is required for resolution-independent access to textures in a shader program. For a two-dimensional texture, the lower-left corner of the texture is at $(0, 0)$ in texture space. The upper-right is at $(1, 1)$. To access a certain pixel P with coordinates (P_x, P_y) in texture space, one has to scale P by the resolution of the texture, yielding the texture space coordinate $(\frac{P_x}{v_w - v_x}, \frac{P_y}{v_h - v_y})$, with v being defined by the viewport as shown in Equation (6.5).

In the remaining work, we always refer to pixels. We do the scaling to texture space when reading the texture. This is not stated explicitly all the time.

Implementation In OpenGL and other rendering systems, multi-pass approaches like these are usually done by using so-called "framebuffer objects", or FBO for short. Their purpose is to redirect the rendering output of a draw call to texture on the GPU. Once activated, everything written by the fragment shader of a draw call is written to a texture, which can be reused by another draw call.

Everything else is trivial on the practical side and works exactly as described above. The only limitation to consider is the amount of texture memory and whether single-precision floating point output is required or 8-bit output is sufficient.

Concept Summary Screen space approaches render the original scene to a texture instead of the screen buffer. The texture is used by another render pass to do further processing via a fragment shader. The fragment shader in screen space has read-access to neighbouring pixels in the provided textures. Each render cycle can output multiple textures and can be cascaded. Textures in general are used to transfer data, but one has to take care of the coordinate space in which transferred vectorial data resides in.

6.2.2 Things to Consider and Potential Pitfalls

There are several limitations and potential pitfalls to be aware of when using screen space rendering. This section provides a short overview on them.

Sub- and Super-Sampling Depending on the scaling of the scene and the point of view, data provided in object space might be sub-sampled or super-sampled during rendering. Figuratively speaking, this is caused when per-vertex data needs to be interpolated. This is the case whenever the rasterized primitive is larger than the pixels representing the vertices. Or, the other way around, when data is lost due to several vertices being mapped to one pixel. This has an influence on the maximum frequency in the data that can be displayed and might pose a challenge to the used interpolation of multivariate data like tensors. Depending on the specific screen space approach and displayed data, this needs to be considered.

Storage and Calculation Precision The calculations done in shader programs are carried out in half-precision mode. It is possible to activate single- or double-precision mode in shaders, but this usually yields a performance penalty. A similar problem exists for data transfer via textures. Usually, a color channel of a texture is eight bit wide. Modern GPU support half- and single-precision floating point textures at the cost of texture memory and automatic interpolation being disabled for these kind of textures. So one has to trade-off numerical precision versus memory and speed.

Locality and Memory Access The GPU works as SIMD machine. This means, it processes **M**ultiple **D**ata using a **S**ingle **I**nstruction synchronized over multiple processing units (PU). Additionally, these groups of PU have their own local memory and can access the global, shared memory. The local memory is small but fast. Accessing it is possible with a very low latency. In contrast, the global memory, accessible by all groups, imposes a high latency on access, due to the type of memory and the shared bus.

The local memory is sufficient to hold the geometry and texture data that is probably needed by the current group of shaders. This holds true as long as the shader reads only local data. In terms of textures, local data refers to the texel assigned to a fragment via texture mapping.

As already mentioned, a fragment shader has read access to the whole textures bound to the current draw call. When a fragment shader reads a texel, which is not cached in the local memory, it causes a cache miss. The GPU then reads from global memory. This will take some time, since the memory is shared and connected via a shared bus. So, the memory fetch instruction of a single PU can cause long wait cycles in a whole group, simply because their instruction flow is synchronized (SIMD idea). In later chapters, we will show that this is an issue for the methods we introduce. In general, this is one of the typical reasons for sudden frame rate drops.

For more details on the GPU architecture, please refer to "*Programming Massively Parallel Processors: A Hands-on Approach*" by Kirk and Hwu [93].

Branching As mentioned above, the GPU works according to the SIMD principle. This means multiple process units (PU) in a group synchronously process a single instruction after the other on multiple fragments (multiple data). If a shader program now contains an if-statement and branches the flow according to it, all PUs whose if-condition failed, have to wait until the other PUs have processed their if-block. This causes tremendous amounts of idle PUs, yielding a severe performance penalty. This is especially true, if the if-block tend to be long or runtime intensive.

Although modern GPU get faster and even implement branch-prediction techniques known in CPU architectures, branching is still a point to keep in mind. Especially in screen space approaches with data dependent branching, it sometimes is better to do a calculation for all fragments and discarding the result later on to avoid idling PUs.

6.3 Summary and Outlook

This chapter provided an overview on the modern GPU and rendering pipeline. It introduced the different coordinate spaces that will play a role in the next chapters and described the general screen space scheme at a glance.

The next three chapters will use these principles in three concrete screen space methods. They improve the structural and spatial perception in different kinds of visualization and contribute to the field of applied computer graphics in visualization.

7

Improved Structure Perception in a Fabric-like Visualization of Tensor Fields

This chapter is based on the following publications:

[P7] – S. EICHELBAUM, M. HLAWITSCHKA, B. HAMANN, and G. SCHEUERMANN. **Fabric-like Visualization of Tensor Field Data on Arbitrary Surfaces in Image Space**. *New Developments in the Visualization and Processing of Tensor Fields*. Ed. by D. H. Laidlaw and A. Vilanova. *Mathematics and Visualization*. 2012, 71–92
Online: http://sebastian-eichelbaum.de/pub10a

[P8] – S. EICHELBAUM, M. HLAWITSCHKA, B. HAMANN, and G. SCHEUERMANN. **Image-space Tensor Field Visualization Using a LIC-like Method**. *Visualization in Medicine and Life Sciences 2*. Ed. by L. Linsen, B. Hamann, H. Hagen, and H.-C. Hege. *Mathematics and Visualization*. 2012, 193–210
Online: http://sebastian-eichelbaum.de/pub10b

7.1 Overview

The Data: Symmetric Second-Order Tensors Tensors are of great interest to many applications in engineering, mechanics, optics, electromagnetism, and medical imaging. They describe the behaviour of several physical effects like material stress, field strength or diffusion in organic tissues.

As this chapter is about computer graphics and the improvement of an existing visualization method for second-order tensor fields, the following paragraphs will provide a rather short background on tensors and second-order tensors in particular. A profound introduction of this mathematical construct and its calculus can be found in *Tensor Analysis and Nonlinear Tensor Functions* [44]. Alternatively, the work by Hagen and Garth [67] provides the necessary fundamentals.

For our purposes, it is sufficient to understand a tensor as a basis-independent geometric object. One of the common definitions of a tensor is using multilinear maps. Multilinear maps can be seen as a function that depends on multiple variables and that is linear with respect to each variable separately.

Definition 7.1 (Tensor as Multilinear Map) *Given a vector space V and its dual space V^*, a (r, s)* **tensor** *can be written as a multilinear map*

$$T : \underbrace{V \times \cdots \times V}_{r \text{ times}} \times \underbrace{V^* \times \cdots \times V^*}_{s \text{ times}} \to \mathbb{R}.$$

The **order** *of a tensor is defined as $r + s$, and its dimension is defined by the used vector space.*

The dual vector space V^* has the same dimension as V and can be seen as the set of all linear forms on V, whereas linear forms are linear maps (functions that obey to additivity and degree 1 homogenity) from V to $\mathbb{R}$.

Definition 7.2 (Secod Order Tensor) *Using the notion of the order of a tensor, one can now define second-order tensors as tensors, where $2 = r + s$; namely $(2, 0)$, $(0, 2)$, and $(1, 1)$ tensors.*

As we work in a Cartesian coordinate system, we can re-formulate the above definitions, because the difference between V and V^* vanishes.

Definition 7.3 (Second-Order Tensor as Matrix) *In Cartesian space of dimension n, a second-order tensor can be written as a $n \times n$ matrix T_{ij}.*

Definition 7.4 (Symmetric Second-Order Tensor as Matrix) *A second-order tensor T_{ij} is called symmetric if and only if $T = T^T$.*

This allows us to understand the second-order tensor as a linear transformation from a Cartesian space to itself. With the above definitions, we are now able to define the second-order tensor field. It represents the association of a linear transformation for each point in the field.

Definition 7.5 (Second-Order Tensor Field) *Let $S \subseteq \mathbb{R}^n$ be a subset in Cartesian space of dimension n and $\mathcal{T}^2$ the set of all second-order tensors with the same dimension in Cartesian space. Then, the mapping*

$$f : S \to \mathcal{T}^2$$

is called second-order tensor field of dimension n.

According to the above definitions, we can apply all rules and properties of quadratic matrices to second-order tensors too. This being said, the $n-th$ dimensional symmetric second-order tensor can be represented by

$$T = R \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix} R^T, \tag{7.1}$$

where $\lambda_1 \dots \lambda_n$ are called eigenvalues. As convention the eigenvalues are sorted: $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$. As we work in a Cartesian system, the rotation matrix R is column-wise defined by the n eigenvectors. For a more detailed introduction and theoretical background on tensors, we refer the reader to the literature [44, 67].

For our purpose, it is important to understand the meaning of the eigenvectors and eigenvalues of the given tensorial data. In many application areas,

the eigenvectors and eigenvalues describe a physical property, which is aimed to be visualized. For diffusion MRI, the eigenvectors and values describe water diffusion direction and intensity; for mechanical application they describe a physical deformation direction and intensity at each point; in electromagnetism, it can describe the electrical field. The tensor is a very flexible and widely used concept in many scientific fields and its visualization is crucial to understand the underlying data.

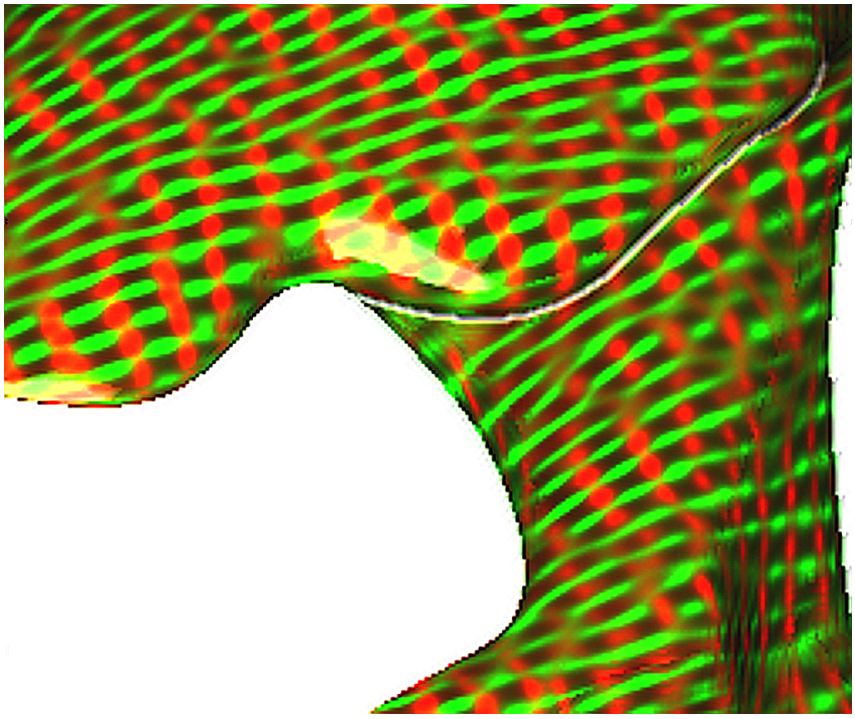
From a visualization point of view, second-order tensor fields are usually seen as a lattice in space with associated tensors at each vertex of the lattice. The multivariate nature of tensors make them especially challenging to visualize. The next paragraphs show an excerpt of possible tensor field visualization methods.

Visualization of Symmetric Second-Order Tensor Data Since the introduction of tensor lines and hyper streamlines [41], there have been many research efforts directed at the continuous representation of tensor fields, including research on tensor field topology [74, 202, 204]. Zheng and Pang introduced HyperLIC [235], which makes it possible to display a single eigendirection of a tensor field in a continuous manner by smoothing a noise texture along integral lines, while neglecting secondary directions. Recent approaches by Hlawatsch et al. [75] and Hlawitschka et al. [77] to visualize Lagrangian structures on tensor fields provide information on one chosen tensor direction and are especially useful for diffusion tensor data, where the main tensor direction can be correlated to neural fibers or muscular structures, whereas the secondary direction only plays a minor role. In 2009, an interactive approach to visualize a volumetric tensor field for implant planning was introduced by Dick et al. [43]. In 2009, Zhang et al. [233] showed the use of evenly spaced tensor lines on surfaces to represent the eigenvector structure.

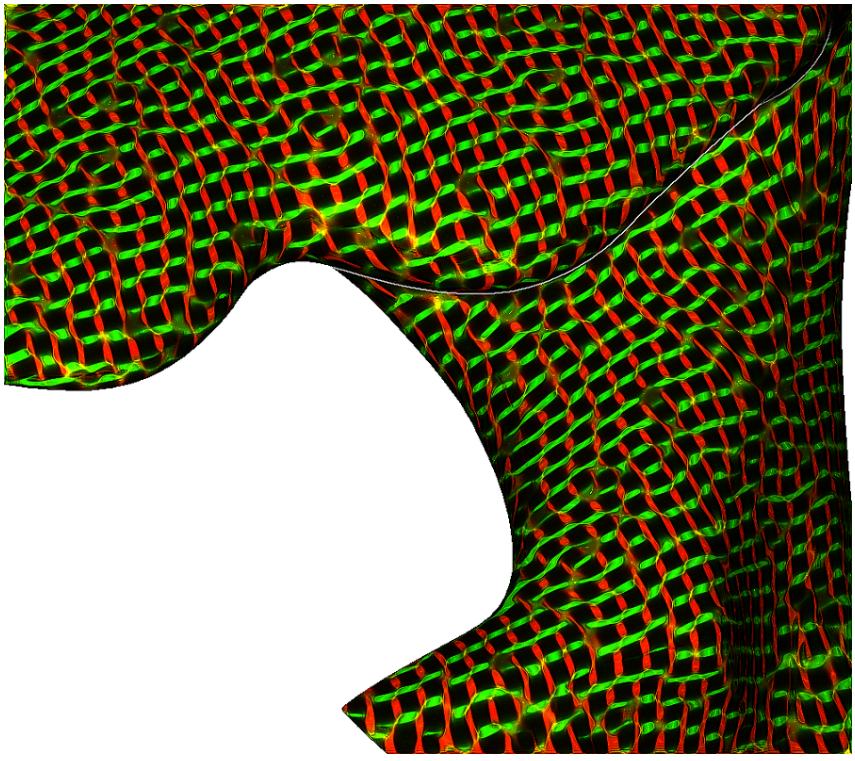
Hotz et al. [83] introduced Physically Based Methods (PBM) for tensor field visualization in 2004 as a means to visualize stress and strain tensors arising in geomechanics. A positive-definite metric that has the same topological structure as the tensor field is defined and visualized using a texture-based approach resembling LIC [28]. Besides other information, eigenvalues of the metric can be encoded by free parameters of the texture definition, such as the remaining color space. Whereas the method's implementation for parameterizable surfaces that are topologically equivalent to discs or spheres is straightforward, implementations for arbitrary surfaces remains computa-

tionally challenging. In 2009, Hotz et al. [84] enhanced their approach to isosurfaces in three-dimensional tensor fields. A three-dimensional noise texture is computed in the dataset and a convolution is performed along integral lines tangential to the eigenvector field. LIC has been used in vector field visualization methods to imitate *Schlieren* patterns on surfaces that are generated in experiments, where a thin film of oil is applied to surfaces, which show patterns caused by the air flow. In vector field visualization, screen space LIC is a method to compute *Schlieren*-like textures in screen space [62, 104, 221, 222], intended for large and non-parameterized geometries. Besides the non-trivial application of screen space LIC to tensor data, screen space LIC has certain other drawbacks. Mainly because the noise pattern is defined in screen space, it does not follow the movement of the surface and, therefore, during user interaction, 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 [218] solved this problem for vector field visualization by generating a three-dimensional texture that is scaled appropriately in physical space.

In contrast to other real-time tensor field visualizations like [232], we developed and implemented an algorithm similar to the original PBM but for arbitrary non-intersecting surfaces in screen space. We call this method "Tensor-Mesh" and it was first published in my diploma thesis [48] (Online: `http://www.sebastian-eichelbaum.de/pub09`). It is explicitly not a part of this thesis. Instead, this chapter focuses on the extensions of the original approach and its visual improvement by postprocessing the resulting images. Tensor-Mesh was able to create a mesh-like structure on arbitrary surfaces that represents the underlying tensor structure. The algorithm was able to perform at interactive frame rates for large datasets. We have overcome the drawbacks present in several screen space LIC implementations by defining a fixed parameterization on the surface. Thus, we did not require a three-dimensional noise texture representation defined at sub-voxel resolution for the dataset. Our approach was capable of maintaining local coherence of the texture pattern between frames when (1) transforming, rotating, or scaling the visualization, (2) changing the surface by, e.g., changing isovalues or sweeping the surface through space, and (3) changing the level of detail. In addition, we implemented special application-dependent modes to ensure our method integrates well with existing techniques.



(a) Original TensorMesh



(b) Improved TensorMesh

Figure 7.1: *Comparison of the original TensorMesh with our improved version. (b) shows the same rendering as above, with a more crisp mesh and a higher mesh resolution. Using our computer graphics methods, we were able to improve the visual quality and the structural perception of the original method tremendously.*

The Problem As the TensorMesh technique visualizes the tensorial data as mesh structure on surfaces, the structural perception of these meshes is crucial to understand the tensor data. Unfortunately, the resulting images were rather blurry and were not optimized towards structural perception.

Our Solution: Shape from Shading The importance of proper shading on the perception of shape and structure has been shown in the past by Ramachandran [155], Langer and Bülthoff [103], and Wanger et al. [215]. In this chapter we use the principle "Shape from Shading" [155] and show that computer graphics allows to improve the perception of data in visualization tremendously.

To improve the structural perception in the TensorMesh approach, we provide two different screen space postprocessings. One that uses an adapted version of the well known bump mapping approach and a streamtube reconstruction that creates a tubular effect for each strand on the surface. Both postprocessings work in ***real-time*** and improve the ***contrast*** of the mesh to create ***crisp and clean*** mesh structures. This way, we are able to provide a ***hugely improved structural perception*** of the underlying tensor mesh.

The next section, will introduce the working principle of the original TensorMesh approach. We will then go on extending this method, by adding a postprocessing step. We explain our modified bump mapping approach and how we achieve tube-like rendering of the TensorMesh. The chapter closes with several visualizations of second-order tensor data and a detailed discussion.

7.2 Background

In this chapter, we enhance a technique we call *TensorMesh*. I introduced it in my diploma thesis [48] and as such, it is not part of this thesis. I introduce the basic working principles here and would like to refer the reader to the publications [P7, P8] for further details.

The TensorMesh technique itself is a screen space method and this section explains its basic working principle, which is also depicted in Figure 7.2. There is only one preliminary step needed on the CPU before running TensorMesh on GPU: the initial noise texture.

Noise Texture Generation Before third requirement of the projection step is a two-dimensional noise texture. In contrast to standard LIC approaches, high-frequency noise textures, such as white noise, are not suitable for the composit-

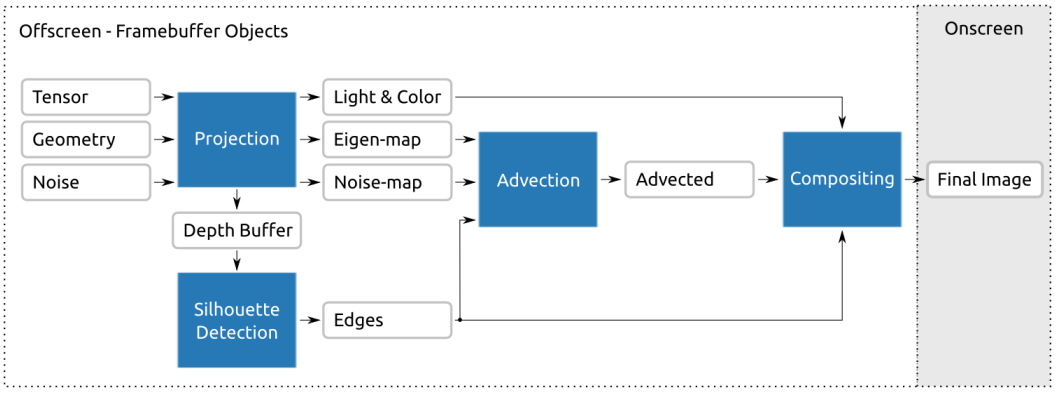


Figure 7.2: *Flowchart indicating the four major steps (blue boxes) of the algorithm:* ***Projection****, which transforms the dataset into a screen space representation and maps the noise texture onto the surface of the rendered geometry;* ***Silhouette Detection*** *uses the depth buffer to reconstruct the visible object borders;* ***Advection****, now uses the two projected major eigenvectors to advect the projected noise along the eigenvector fields until it reaches a geometry edge; finally, the* ***Compositing*** *step combines both advected noise textures, adds original coloring, and outputs the result to the screen.*

ing of multiple, advected textures. Their high-frequency details would not create the mesh-like result as intended with TensorMesh. Therefore, we compute the initial noise texture using the reaction diffusion scheme first introduced by Turing [205]. It simulates the mixture of two reacting chemicals, which leads to larger, but smooth "spots" that are randomly and almost uniquely distributed (cf. Figure 7.3, right). This can be precomputed on the CPU once. The created texture can then be used for all consecutive frames. For the discrete case, we define three discrete, two-dimensional images $a_{i,j}$, $b_{i,j}$, and $\beta_{i,j}$. Each pixel can be accessed by the indexing parameters i and j. The field β gets initialized with white noise, whereas a and b are initialized with 0.5 for each i, j. This can be seen as the initial concentration of the two chemicals. The iterative change of each value in a and b for a single iteration is defined as

$$\begin{aligned} \Delta a_{i,j} &= F(i,j) + D_a \cdot (a_{i+1,j} + a_{i-1,j} + a_{i,j+1} + a_{i,j-1} - 4 \cdot a_{i,j}), \\ \Delta b_{i,j} &= G(i,j) + D_b \cdot (b_{i+1,j} + b_{i-1,j} + b_{i,j+1} + b_{i,j-1} - 4 \cdot b_{i,j}), \text{where} \\ F(i,j) &= s(16 - a_{i,j} \cdot b_{i,j}) \text{ and } G(i,j) = s(a_{i,j} \cdot b_{i,j} - b_{i,j} - \beta_{i,j}). \end{aligned} \tag{7.2}$$

Here, we assume continuous boundary conditions to obtain a seamless texture in both directions. The scalar s allows one to control 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. We gained both constants empirically. They directly influence the shape and density of the created spots.

Both images a and b converge, depending on the used parameters, relatively fast. TensorMesh then takes image a and uses it as input for the projection step. Figure 7.3(a) shows such a generated and tiled input texture.

7.2.1 Step 1: Projection to Screen Space

As with every screen space method, the TensorMesh method starts by rendering the arbitrary geometry. The geometry additionally contains the second-order tensor associated with each vertex of the geometry. In practice, this can be done by storing the six needed components of the symmetric 3×3-matrix as two three-dimensional texture coordinates, as generic vertex attributes, or as three-dimensional textures (cf. Section 6.1). When rendering the scene, the method uses a vertex/fragment shader pair for processing the geometry and each possible pixel (fragment). The results get written to four output textures:

- Light and Color: the standard geometry color and lighting, as if the geometry would be rendered to screen without any further processing. This is provided by the GPU automatically.
- Eigen-map: the two major eigenvectors, projected to the geometry surface.
- Noise-map: the noise used during advection along the eigenvectors. This noise was projected to the surface of the geometry too.
- Depth buffer: finally, the standard depth buffer of the scene. This is provided by the GPU automatically.

The next paragraphs summarize the creation of the eigen-map and the noise-map.

The Eigen-map At this point, the vertex shader has access to the tensor and the geometry normal of the currently processed vertex. Using this, it can project the tensor to the surface of the geometry as

$$T' = P \cdot T \cdot P^T, \tag{7.3}$$

with a matrix P defined using the surface normal n as

$$P = \begin{pmatrix} 1 - n_x^2 & -n_y n_x & -n_z n_x \\ -n_x n_y & 1 - n_y^2 & -n_z n_y \\ -n_x n_z & -n_y n_z & 1 - n_z^2 \end{pmatrix}. \tag{7.4}$$

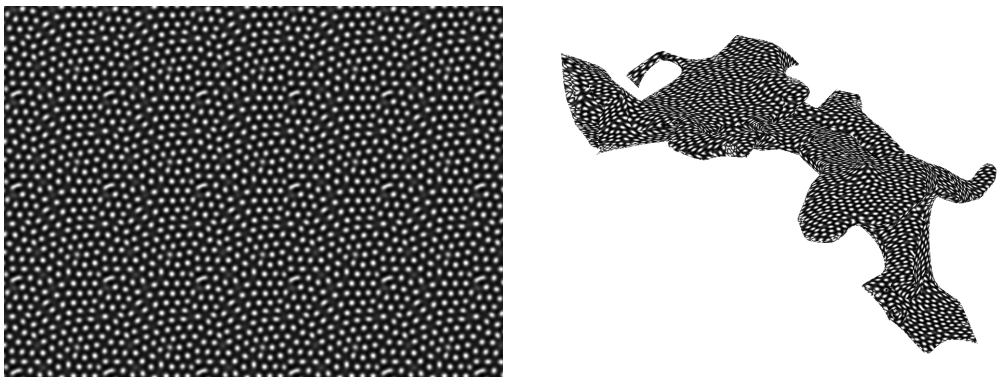
The surface-projected tensor T' is now stored in a per-vertex variable. Per-vertex variables are interpolated by the GPU during rasterization. The interpolated value will be available during each run of the fragment shader. The component-wise interpolated tensor is then decomposed into the eigenvector/eigenvalue representation using a method derived from the one presented by Hasan et al. [71], only using iteration-free math functions. This causes a tremendous acceleration on the GPU. With this method, we calculate the three real-valued, orthogonal eigenvectors $v_{\lambda_{1-3}}$ and the corresponding eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3$. In our method, we are only using the first two eigenvectors, showing the two main directions. The eigenvectors, still defined in object space, are projected into screen space using the same projection matrices $M_{ModelView}$ and $M_{Projection}$ used for projecting the geometry to screen space (cf. Section 6.1.2). These usually are the standard *modelview* and *projection* matrices OpenGL offers:

$$v'_{\lambda_i} = M_{Projection} \times M_{ModelView} \times v_{\lambda_i}, \text{ with } i \in \{1, 2\}. \tag{7.5}$$

After the projection, the two major eigenvectors can be stored in the corresponding pixel of the eigen-map.

The Noise-map Mapping the initial texture to the geometry can be a difficult 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. [87, 154]) and, in general, require additional and often time-consuming preprocessing. Another solution, proposed by Turk [206], calculates the reaction diffusion texture directly on the surface. A major 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 preprocessing time tremendously.

For our purpose, a simple, yet fast and flexible mapping strategy is used. Informally spoken, we classify each surface point P of the geometry to belong



(a) Turing noise (b) Mapped to geometry

Figure 7.3: *Illustration of the reaction diffusion texture used (left) and the noise texture mapped to geometry (right). For a higher resolution in the projection step, we tiled the noise texture multiple times. The appearing repetition artifacts do not play any role after projection.*

to one certain voxel of a virtually defined 3D grid. This can be interpreted as discretization of the surface with the help of the implicit voxels. The normal at P on the surface is then used to find the most similar side of the voxel associated with P. Once the side-plane is found, the point's P texture coordinates can be determined by:

Side-normal	Texture coordinates
$(1,0,0)$ or $(-1,0,0)$	(p_y, p_z)
$(0,1,0)$ or $(0,-1,0)$	(p_x, p_z)
$(0,0,1)$ or $(0,0,-1)$	(p_x, p_y)

Please note that we assume the texture coordinates to be defined in a wrapped and continuously defined coordinate system, which is common in OpenGL. This allows the seamless tiling of the input noise texture on each voxel surface, which then is mapped to the surface. This can be interpreted as an orthographic projection of the voxel side plane onto the surface along the plane's normal vector. By changing the size of voxels during the calculation, different frequencies of patterns can easily be produced and projected onto the geometry. This capability allows one to change the resolution of the texture as required for automatic texture refinement when zooming. In practice, this can be done in the fragment shader easily. It automatically is called for every visible surface point P and can now use the object space normal of P to calculate the texture coordinate:

Listing 7.1: *Map the input vertex coordinate to a texture coordinate in the 2D noise texture.*

```
float noise( vec3 P,      // surface point in object space
             vec3 normal // P's normal in object space
           )             // return the noise value for P
{
    // Allow scaling the virtual voxel space. The variable
    // bbSize contains the maximum dimension of the
    // geometry bounding box. This normalizes P to a
    // texture space interval of of [0,1]. This represents
    // one virtual voxel per bounding box. nbVoxels then
    // allows to increase this.
    pScaled = ( P * nbVoxels ) / bbSize;

    // The default case if the normal points exactly
    // towards the voxel corners.
    vec2 noiseTexCoords = vec2( pScaled.x, pScaled.y );

    // Find the side of the virtual voxel to which
    // the object space normal of P points to.
    if( abs( normal.x ) >= max( abs( normal.y ),
                                abs( normal.z ) )
      )
    {
        noiseTexCoords = vec2( pScaled.y, pScaled.z );
    }
    else if( abs( normal.y ) >= max( abs( normal.x ),
                                     abs( normal.z ) )
           )
    {
        noiseTexCoords = vec2( pScaled.x, pScaled.z );
    }
    else if( abs( normal.z ) >= max( abs( normal.x ),
                                     abs( normal.y ) )
           )
    {
        noiseTexCoords = vec2( pScaled.x, pScaled.y );
    }

    // Get the value from the input noise texture,
    // represented by noiseTexture.
    return texture2D( noiseTexture, noiseTexCoords ).r;
}
```

Regardless of its simplicity, this method supports a fast and flexible parameterization of the surface space that only introduces irrelevant distortions (cf. Figure 7.3), which vanish during the advection step. Additionally it creates a

consistent visual impression of the noise on the surface when interacting with the scene. Figure 7.3(b) shows the noise mapped on an arbitrary geometry.

7.2.2 Step 2: Silhouette Detection

With the eigen- and noise-map, the advection step could now follow the streamlines on the eigen-map images. To avoid advecting over geometry borders, we require the geometry edges first. These edges are calculated in the silhouette detection step. We use the depth buffer of the geometry and apply a Laplace filter on it. As this second step already works in screen space, this is done for each pixel of the input depth buffer in a single fragment shader. The result is written to an output texture for later use.

Although the Laplace filter could be applied on demand during the next step, we precalculate these edges. This is useful as the same pixel might be visited multiple times during the advection step, causing the edge to be calculated multiple times. A counter-argument to precalculation would be to assume that a large fraction of the pixels do not belong to the rendered surface, hence making edge detection on these pixels superfluous. In this case, the precalculation might be skipped. However, the definition of *large fraction* is difficult and a gain in performance also depends on the actual vector fields and the length of integration in the next step.

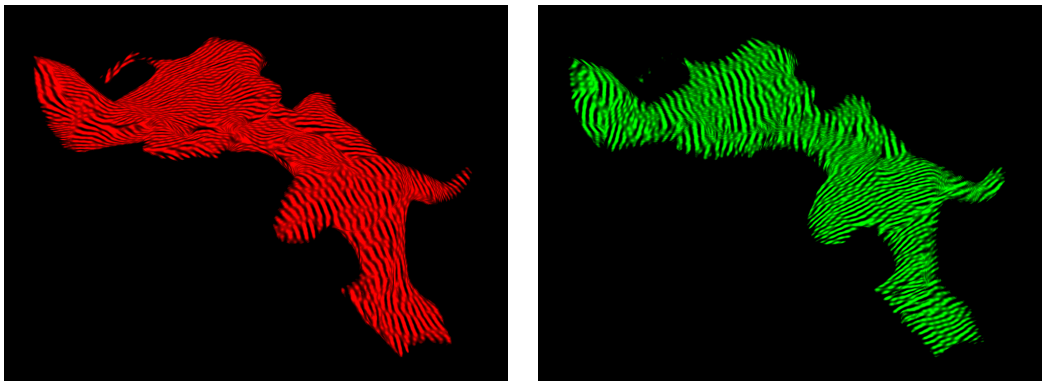
7.2.3 Step 3: Advection

We have discussed how to project the geometry and the corresponding tensor field to screen space. With the prepared screen space eigenvectors and the input noise texture on the geometry, the advection can be done for both of the eigenvector fields.

The advection step is conceptually very similar to the line integral convolution (LIC) approach, but in our case, we do not calculate streamlines at each position of both vector fields. As a fragment shader has no write-access to other pixels, it is not able to convolute the noise along the streamlines. That is the reason why we use Euler integration to follow the vector field from the current pixel in both directions and combine the noise value from each vector field sample using a weighted sum. This way, we write only to the originating pixel and achieve a visually similar result. The length of the vector, used for each step during integration, is one pixel to ensure that no detail is missed.

Listing 7.2: *Advection of the noise texture acording to the given input eigenvector field. This GLSL-like pseudo code demonstrates the advection at a given pixel P in a fragment shader.*

```
float advect( vec2 P,           // the pixel
              int fieldIndex,   // vector field to use
              int numIterations // number of iterations
            )                   // return advected noise
{
    // Iterate along the field, in both directions.
    vec2 lastPos1 = P; vec2 lastPos2 = P;
    // Keep track if an edge was crossed
    bool isOutside1 = false; bool isOutside2 = false;
    // How much iterations have contributed?
    int numContributions = 0;

    // Iterate
    float sum = 0.0;
    for( int i = 0; i < numIterations; ++i )
    {
        // Calculate new positions, in forward and backward
        // directions. In the case of Euler integration and
        // a step size of 1, those functions can be written
        // as lastPos1 + getVector( fieldIndex, lastPos );
        vec2 newPos1 = getNextPos( fieldIndex, lastPos1 );
        vec2 newPos2 = getPrevPos( fieldIndex, lastPos2 );

        // Get the edge information from step 2. 1 if the
        // pixel is an edge or outside, 0 otherwise.
        isOutside1 = isOutside1 || isEdge( newPos1 );
        isOutside2 = isOutside2 || isEdge( newPos2 );

        // Let the sample's noise contribute. We found that
        // an unweighted contribution yields the best
        // results. It is also possible to scale using a
        // geometric progression, distance dependent, ...
        // Note: only contribute if the positions did not
        //       cross an edge.
        sum += float(!isOutside1) * getNoise( newPos1 );
        sum += float(!isOutside2) * getNoise( newPos2 );

        // Keep track
        lastPos1 = newPos1;
        lastPos2 = newPos2;
        numContributions += int(!isOutside2) +
                            int(!isOutside1);
    }

    // The sum needs to be scaled to [0,1] again.
    return = sum / numContributions;
}
```



(a) Advection along v'_{λ_1} (b) Advection along v'_{λ_2}

Figure 7.4: *Advection texture after ten integration steps. Left: red channel containing advected noise along the eigenvectors v'_{λ_1}; Right: green channel containing the advected noise along the second eigenvectors v'_{λ_2}.*

Integration is stopped in two cases:

1. when reaching a given maximum integration length,
2. or when an edge is reached (cf. Edges-texture in Figure 7.2).

We also considered using other, higher order streamline integration schemes, but the view-dependent super- and sub-sampling during projection to screen space voids the advantages over the Euler method. The GLSL-like pseudo code in Listing 7.2 shows the basic idea, as done for each pixel P in this step's fragment shader.

In our originally published articles [P7, P8], we used the so-called ping-pong approach for advection. There, we advected the incoming noise texture only along one step of the vector fields. The resulting texture was then used as input for another advection render pass, which renders another single-step advection to a different output texture. This new texture is then used as input for the next advection step, whereas the old input is used as the output, hence the name "ping pong". This was faster on older GPU architectures, but is obsolete today due to the increased local shader unit memory and shader unit processing power.

The result of the advection step with ten integration steps is shown in Figure 7.4. For later reference, we call the advected images A_{λ_1} and A_{λ_2}.

7.2.4 Step 4: Compositing

In a subsequent rendering pass, a fabric-like texture is composed. The compositing step combines the advection results for A_{λ_1} and A_{λ_2} into one image as RGB triple:

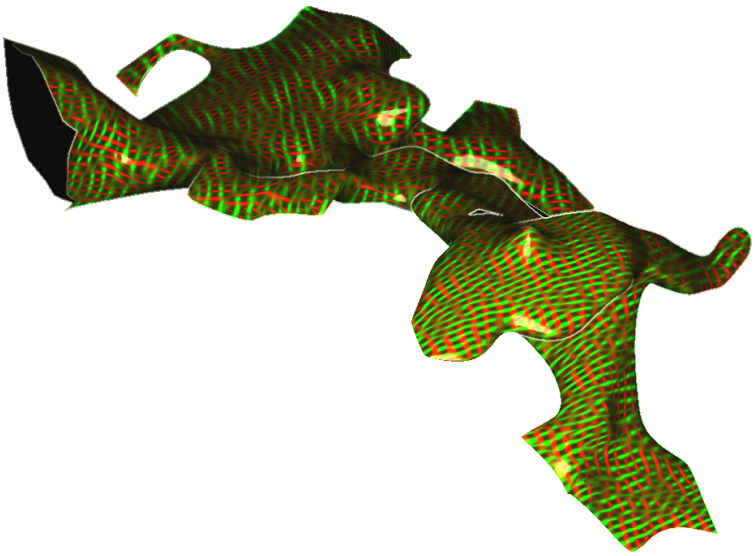
$$\begin{aligned} R &= \frac{r \cdot A_{\lambda_2}}{8 \cdot (A_{\lambda_1})^2}, \\ G &= \frac{(1-r) \cdot A_{\lambda_1}}{8 \cdot (A_{\lambda_2})^2}, \\ B &= 0. \end{aligned} \tag{7.6}$$

Equation (7.6) is a weighting function between the two advected images for both eigenvectors. The scalar factor r is used to blend between the two tensor directions. If both directions are equally important, a value of 0.5 ensures an equal blending of both directions. To explain the above compositing scheme, we are using the red component as an example. The red color should represent the main tensor direction. We therefore reduce the intensity of the second eigenvector image A_{λ_2} using the over-emphasized first eigenvector image A_{λ_1}. To furthermore emphasize the influence of a high intensity in the advected image for the first eigenvector, the denominator is squared. This way, pixels with a high intensity in the first eigenvector direction get a high red intensity. This is done vice versa for the green channel. The compositing implicitly utilizes the clamping to $[0, 1]$ which is done for colors on the GPU. Additionally, it is possible to blend in the edges or the original coloring and lighting of the geometry, as shown in Figure 7.5. The final image is then shown on screen again.

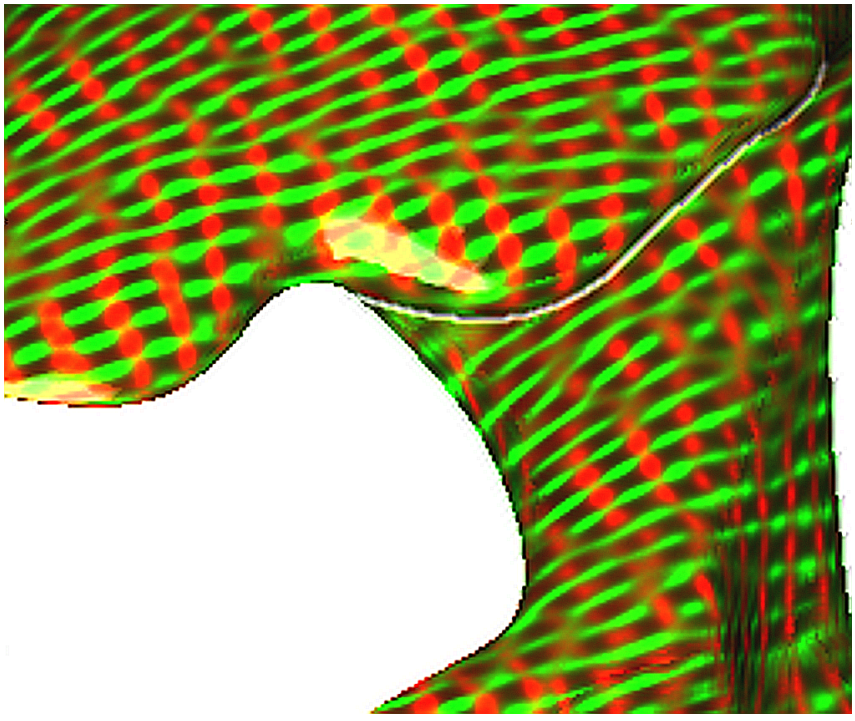
Using this offscreen pipeline, we were able to reproduce the mesh-like structure of a second-order tensor field on arbitrary geometry, independent of the algorithm creating the geometry or the source of the tensorial data. But as Figure 7.5 showed, the results are rather blurry and do not represent a clean mesh structure perfectly.

7.3 Method

In the previous section, we introduced the original TensorMesh method. Right after publishing my diploma thesis [48], we thought about how to improve the visual quality of the method and published the results [P7]. These improvements are the subject of this section.



(a) TensorMesh



(b) Zoomed in

Figure 7.5: *The composited image produced by the compositing shader. (a): the whole geometry. (b): a zoomed part of the geometry to show the still blurry fabric structure on the surface. The chosen geometry has no further meaning. We have chosen an arbitrary, rather unshaped surface inside a brain's DTI image to show how well TensorMesh works on these kind of surfaces. Especially (b) shows that our method properly detects and handles the borders of the geometry.*

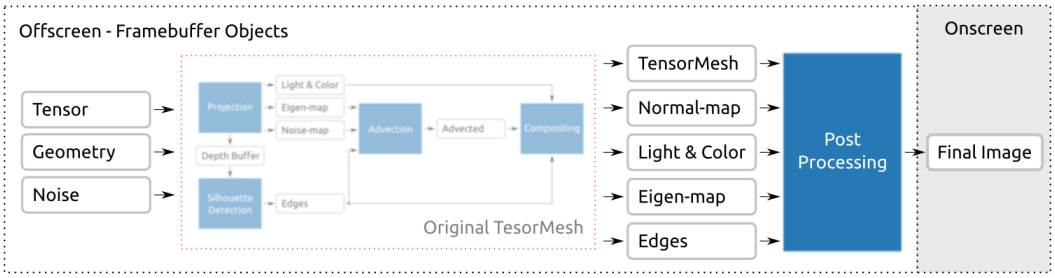


Figure 7.6: *Flowchart indicating the additional postprocessing step in the context of the original TensorMesh method. Instead of rendering the formerly final image to screen, it is now stored in a texture. The postprocessing step can then utilize the image and improve its visual quality. Additionally, the original projection step now also outputs the surface normals in screen space to a texture. Besides this, the other outputs of the projection and silhouette detection steps are reused. They have been calculated by the original method, and we bind them as input textures to the postprocessor to achieve the effects described in this chapter.*

As the original TensorMesh method completely works in screen space, it was easy to extend it with another, fifth offscreen processing step: the postprocessing. Figure 7.6 shows the updated method schematically. We modified TensorMesh to output the image to another texture, which then can be processed by the postprocessor.

7.3.1 Postprocessing

Figure 7.5 shows the result of Equation (7.6) combined with Blinn-Phong shading. Even though Blinn-Phong shading [18] provides 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 [30]. These techniques can be incorporated in our compositing scheme easily, but provide a perception improvement for the shape of the geometry only. The results still look blurry and justify the need for additional postprocessing to improve the structural detail *on* the surface. We have implemented two postprocessings which reduce blur and create crisp and appealing images, which improve the perception of the represented structures tremendously.

At this point, please keep in mind that the following calculations work in screen space and need to be done for each visible pixel P. For the sake of simplicity, we left out the explicit reference to P in each of the following formulae, although they are dependent on the current P.

Bump Mapping

In computer graphics, triangles are usually used to represent complex geometric objects. The surface of these objects can be lit in many different ways, for example with Blinn-Phong shading [18] or Cook-Torrance shading [32] to mention only two. Whatever method is chosen, they have one thing in common: the surface normal. It defines the surface's orientation and significantly influences the shading of it – and the shading defines the local structure and how the human vision percepts the surface and its shape. For details on the perception of shape, structure and spatiality, we refer the reader to the work of Ramachandran [155], Langer and Bülthoff [103], and Wanger et al. [215].

The surface normal is usually defined per triangle or vertex and is linearly interpolated in between. To increase the surface detail, one has to tremendously increase the triangle mesh complexity. This is not feasible as the graphics processor will reach its limits sooner or later. To circumvent this issue, bump mapping was introduced by Blinn [19] to simulate three-dimensionality via shading in otherwise planar surfaces. The idea is to define a normal-map on a planar surface, like a triangle. This additional map allows assigning different normals to each rasterized point of the surface. This way, the amount of normals on a single triangle can be increased tremendously. Accordingly, the shading can reproduce more structural detail on it. The bump mapping technique is quite widely used in computer games to simulate high detailed surface structures on a few triangles only. A typical example would be the rough stone texture on a coarsely modelled rock. We wanted to achieve a similar effect for the TensorMesh rendering. This is why we use the basic idea of bump mapping to improve the structural perception of the fabric mesh on the rendered surface.

The original bump mapping method relies on a given normal-map and precalculated surface tangents. The normal-map is usually created by a model designer to match the requirements of the specific object. With the normal and a tangent at each point, bump mapping is able to define the so-called *tangent space*. With the help of this, all lighting calculations, independent of the actual light model used, can be done in tangent space. We do not use this method, since we do not have the tangent space available in screen space anymore. Instead we rely on the *conceptual idea* of bump mapping, namely using a specific normal for lighting at each rasterized point. For specific implementation details on the original bump mapping method, we would like

to refer the reader to the book *Mathematics for 3D Game Programming and Computer Graphics*, Section 7.8 on bump mapping by Lengyel [106].

To achieve a similar effect in screen space, we modify the original approach and extend it to use normals implied by the mesh structure on the surface. To calculate a normal at a pixel P, we first calculate the two-dimensional gradient of the intensities of the original TensorMesh result texture. We call this

$$g = ||\nabla(R + G)||. \tag{7.7}$$

Using $R + G$, we denote the intensity of a pixel. According to Equation (7.6), we only use the red and green channel. The blue channel can be ignored as the original TensorMesh only produces output in the red and green channel. In practice, gradient calculation on the GPU can be done fast by calculating the difference quotient in x and y directions, with a step size of one pixel.

Additionally, we modified the projection step of the original method to provide screen space surface normals in a texture. Figure 7.6 shows the normal-map as an additional outcome of the TensorMesh algorithm. The projection step calculates this normal-map by projecting the eye space normal to screen space. This is done by using the projection matrix of the rendering system, as explained in Section 6.1.2. Please note that the normal still is three-dimensional.

With the gradient vector, we are now able to deviate the screen space normal n in direction of g. The new normal

$$n' = n + (g_x, g_y, 0)^T \tag{7.8}$$

can then be used in the Blinn-Phong equations to calculate the overall light intensity. In our case, we only use the diffuse and specular parts. Ambient light is not used, as it is not depending on the normal and, hence, does not contribute to the bump-mapping effect. This yields the amount of reflected light of a single light source i as

$$\begin{aligned} \mathcal{B}_i = & I_i^{in} k_{diffuse} < L_i, n' > + \\ & I_i^{in} k_{specular} < ||L_i + V||, n' >^s \end{aligned} \tag{7.9}$$

and for all light sources

$$\mathcal{B} = \sum_{i \in lights} \mathcal{B}_i. \tag{7.10}$$

The vectors L_i and V are the normalized screen space vectors pointing from P to the light source (L) and the camera point (V) respectively. The material properties $k_{diffuse}$, $k_{specualr}$, and s are usually defined by the rendering system, but in our case, we set $k_{diffuse} = 1$, $k_{specular} = 1$, and $s = 200$. They represent the material's diffuse and specular reflectance as well as its shininess. We have chosen these values as we are interested in the reflection factors only. We cannot tell anything about the material properties, nor do we want to implement a physically accurate lighting model. Our focus is on a light-based shading of the mesh structure for improved structural perception. This is also the reason for setting the incoming light intensity $I_i^{in} = 1$. The high shininess value s ensures subtle specular highlights on the mesh structure. We additionally do not calculate the reflection factor $\mathcal{B}$ separately for each color channel. This is because we do not want to stain the color of the rendering due to non-white light sources. This way, we keep the original red-green coloring of the TensorMesh.

$\mathcal{B}$ can now be combined with the original colors and the edges e at P that have been calculated by the original method. The pixel's RGB triple is then defined as

$$\begin{aligned} R_{bump} &= \mathcal{B} \cdot R + e, \\ G_{bump} &= \mathcal{B} \cdot G + e, \\ B_{bump} &= e. \end{aligned} \tag{7.11}$$

This yields the rendering Figure 7.7(a). When adding a contrast enhancement to the red and green channels as

$$\begin{aligned} R_{bump} &= \mathcal{B} \cdot (R \cdot G + R^2) + e, \\ G_{bump} &= \mathcal{B} \cdot (R \cdot G + G^2) + e, \\ B_{bump} &= e, \end{aligned} \tag{7.12}$$

we are able to improve the crispness of the rendered image as shown in Figure 7.7(b), compared to Figure 7.7(a). Keep in mind that we are using the automatic clamping of each color channel to the interval $[0, 1]$. This way, the silhouette of the geometry is always white and we do not need to take care that the contrast-enhanced colors are larger than one. The Equations (7.11) and (7.12) also allow combining the enhanced TensorMesh with other effects and colorings, calculated earlier. For example, we combined a colormap of the

original data with the mesh, as shown in Figure 7.10. This information can be transferred using the "Color & Light" texture from Figure 7.6.

Streamtube Effect

With the help of bump mapping, we achieve a better spatial impression of the fabric-like pattern. A different visual improvement can be achieved by interpreting the structure on the surface as streamtubes along the surface. The idea of representing trajectories and streamlines using tubes is not new. Zhang et al. [234] used streamtubes to visualize DTI data of the human brain. The approaches of Merhof et al. [128] and Schirski et al. [174] enabled the use of large amounts of tubes without performance penalty due to the increased amount of geometry, when compared to simple lines. We also create the visual effect of streamtubes on the geometry's surface, without actually creating tubes to replicate the mesh effect.

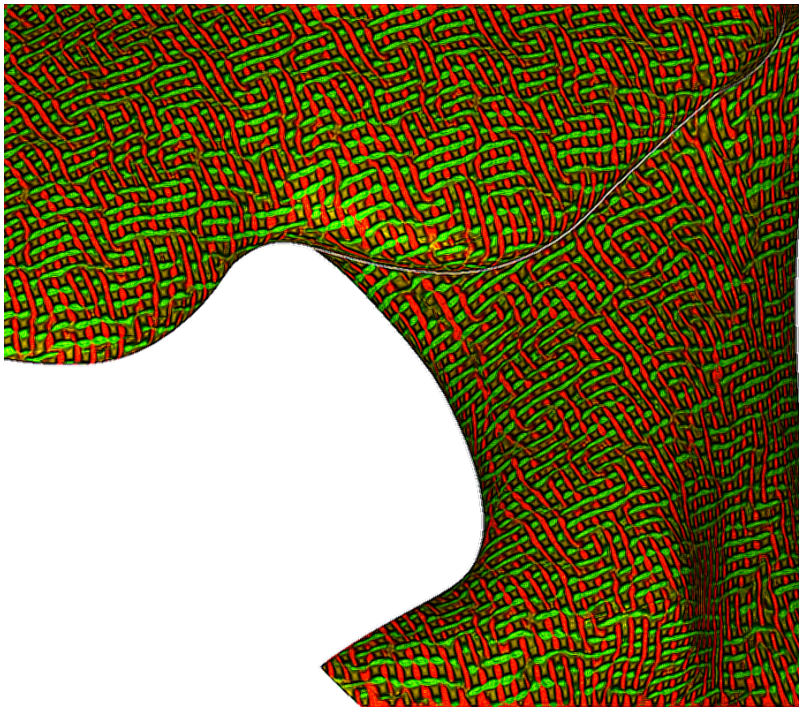
The first test to do before doing any further calculations is to check whether the pixel P currently belongs to one tube of the original TensorMesh. This can be done by checking the value of the red and green channel of the input TensorMesh texture at P. If the red or green channel is larger than a given threashold, 0.2 in our case, then it belongs to a tube. If not, the pixel can be skipped and rendered in black.

Next, we need to have a tangential coordinate system, similar to the one needed for the original bump mapping. The screen space eigenvectors v'_{λ_1} and v'_{λ_2} from Equation (7.5), transferred using the Eigen-map texture, are interpreted as the tube tangents for the first and second eigenvector field. These tangents denote the direction of the tube along the surface and, together with the trivial surface normal of $(0,0,1)^T$, define the bi-normal vectors for each eigenvector field. When thinking of each tube as cylinder that runs parallel to the screen plane, and its direction is defined by the eigenvector on the screen plane, we can define the bi-normals as radial vector b for each eigenvector field to be

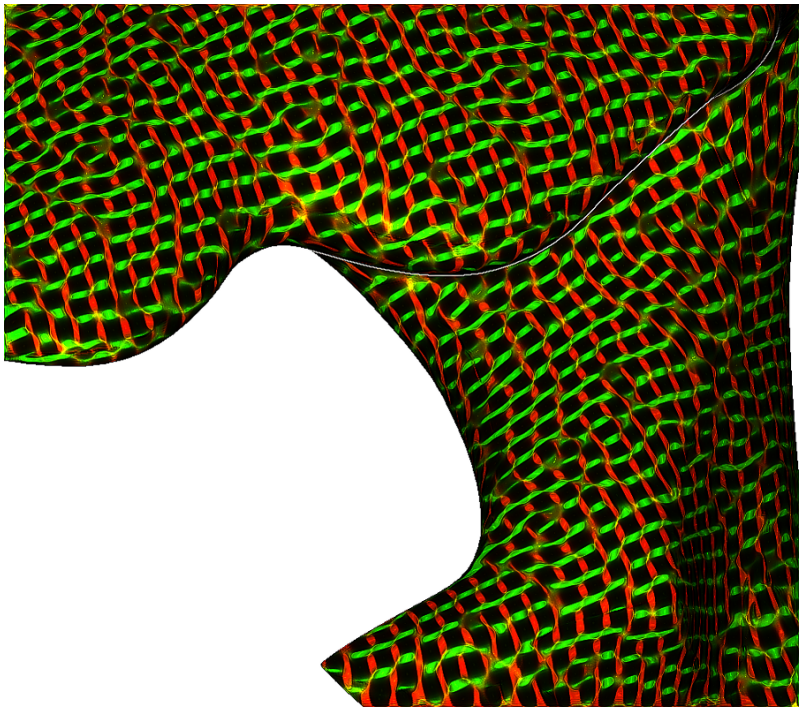
$$b_{\lambda_1} = ||(0,0,1)^T \times v'_{\lambda_1}|| \text{ and } b_{\lambda_2} = ||(0,0,1)^T \times v'_{\lambda_2}||. \tag{7.13}$$

To simplify the further descriptions, we describe the next steps for rendering the tubes of v'_{λ_1}. These steps need to be done for the second eigenvector too.

The bi-normal b_{λ_1} is now used to find the correct normal on the tube-surface at the current pixel for lighting. As we already know that a pixel is on a tube, we search the boundaries of the tube in radial direction. Therefore, we sample



(a) Bump mapping only



(b) Bump mapping with enhanced contrast

Figure 7.7: *The final image produced by the postprocessing shader in combination with bump mapping and combined edges. Top: standard bump mapping. Bottom: the same zoomed part of the original geometry to show the effect of additional contrast enhancement of Equation (7.12). This approach creates a more fabric-like impression and looks like rotating ribbons similar to stream ribbons.*

in direction of the bi-normal, to find the pixel whose red color value is smaller than 0.2. The step size is one pixel and the value 0.2 is the threshold we used earlier to check whether a pixel belongs to a tube or not. In other words, we search for the smallest, positive scaling factors $a^p_{\lambda_1}$ and $a^n_{\lambda_1}$, which scale the bi-normal b_{λ_1} and $-b_{\lambda_1}$ to point to the nearest pixel with value < 0.2 in the red channel. The width of the tube passing the current pixel is then defined by $a^p_{\lambda_1} + a^n_{\lambda_1}$ and

$$p_{\lambda_1} = 2 \cdot \left(0.5 - \frac{a^p_{\lambda_1}}{a^p_{\lambda_1} + a^n_{\lambda_1}}\right) \in [-1, 1] \tag{7.14}$$

defines the relative position of the current pixel on this tube regarding the (normalized) bi-normal, where 0 is the middle of the tube. Also note that $a^p_{\lambda_1}$ and $a^n_{\lambda_1}$ are both positive.

This information is enough to define a diffuse shaded surface. However, we want proper per-pixel Blinn-Phong shading and therefore need to use this to calculate the tube normal at the current pixel:

$$n^{tube}_{\lambda_1} = (1 - p^2_{\lambda_1})(0, 0, 1)^T + p^2_{\lambda_1} b_{\lambda_1}. \tag{7.15}$$

The value of p is additionally squared to achieve the effect of a round surface. The normal $n^{tube}_{\lambda_1}$ is then used to calculate the Blinn-Phong shading on the surface similar to Equation (7.9) and yields $\mathcal{B}_{\lambda_1}$.

When doing the above steps for both eigenvector directions, one gets the shading factors $\mathcal{B}_{\lambda_1}$ and $\mathcal{B}_{\lambda_2}$. In combination with the silhouette e, they allow the definition of the final RGB triple of the pixel P under consideration of the $[0, 1]$ interval-clamp of the GPU as

$$\begin{aligned} R_{tube} &= \mathcal{B}_{\lambda_1} + e, \\ G_{tube} &= \mathcal{B}_{\lambda_2} + e, \\ B_{tube} &= e. \end{aligned} \tag{7.16}$$

This produces the tube-like effect with proper structural impression on the surface as shown in Figure 7.8.

7.3.2 Implementation

The implementation of the pipeline shown in Figure 7.2 is straight forward and can be done as indicated in Section 6.2. The figure clearly shows the

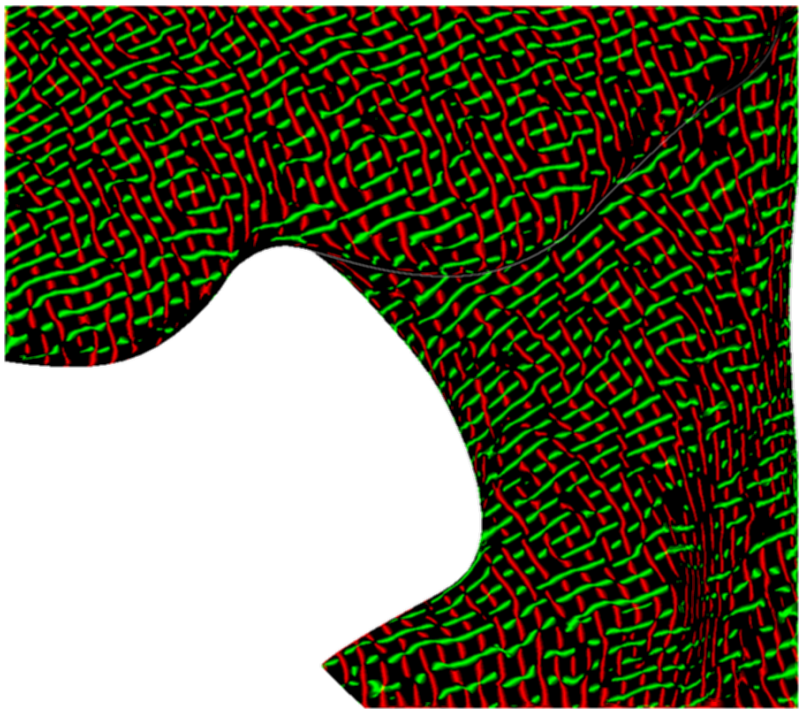


Figure 7.8: *Interpreting the final image from Figure 7.5 as streamtubes along the geometry's surface. When lighting them accordingly, the results are tremendously less blurry and create a crisp and clean impression of a mesh-like structure on it.*

input and output textures of each step and their execution order. The whole pipeline is implemented using OpenGL and framebuffer objects (FBO), which allow the efficient offscreen rendering and screen space based processing we need. Each step is implemented using a vertex/fragment shader pair and simply evaluates the above mentioned formulae for each pixel. The projection step is the beginning of the pipeline and the only step which is not in screen space. For the consecutive steps, we render a quad, filling the whole viewport of the FBO. The outputs and inputs are then bound as textures to the FBO and the quad respectively. The used textures are 8 bit per channel textures and of the same size as the viewport to avoid any interpolation when reading the textures as input. The increased numerical precision in floating point textures is not needed for TensorMesh and the postprocessing step.

Types of Surfaces Our implementation is not limited to a special kind of geometry. It is able to handle every second-order 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 dataset to generate a surface in a three-dimensional data domain. Other methods include hyper-stream surfaces [41], wrapped streamlines [49], or domain-dependent methods like

dissection-like surfaces, as presented by Anwander et al. [6]. Direct rendering approaches, like isosurface ray-tracing by Knoll et al. [94] and other similar approaches are possible too. The only requirement for the surface is that it is non-self-intersecting and that smooth normals are provided as they are required for the projection step and for proper lighting.

Tensor Upload and Processing As 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 available on the GPU, it is possible to map the two main directions to the surface described by the normal n at the current vertex using Equation (7.3). This projection is implemented in a per-vertex manner in the vertex shader. In contrast, eigenvalue decomposition, eigenvector calculation, and screen space projection need to be done in the fragment shader. Since the eigenvectors are without orientation, it is possible to have sign flips between adjacent vertices. If the interpolation on the surface in between the vertices takes place after the eigenvector decomposition, these sign changes can render the interpolation useless.

The projected eigenvectors v'_{λ_1} and v'_{λ_2} need to be scaled since textures are used for transportation, where each value must be in the interval $[0, 1]$. To simplify further data handling and storage on the GPU, we scale the eigenvectors as follows:

$$\|v\|_\infty = max\{|v_x|, |v_y|\} \tag{7.17}$$

$$v''_{\lambda_i} = \frac{v'_{\lambda_i}}{\|v'_{\lambda_i}\|_\infty} \text{ with } i \in \{1, 2\}, \quad \text{and} \quad \|v'_{\lambda_i}\|_\infty \neq 0 \tag{7.18}$$

The maximum norm (L_∞-norm) ensures that one component of the eigenvector is 1 or -1 and, therefore, one avoids numerical instabilities arising when limited storage precision is available, and can use memory-efficient eight-bit textures. The special case $\|v'_{\lambda_i}\|_\infty = 0$ only appears, if the surface normal and the eigenvector both point into the same direction. This case needs to be handled in the shader. The scaling can be avoided when using floating point textures. However, we found that the increased storage precision provides no further advantage visually and qualitatively.

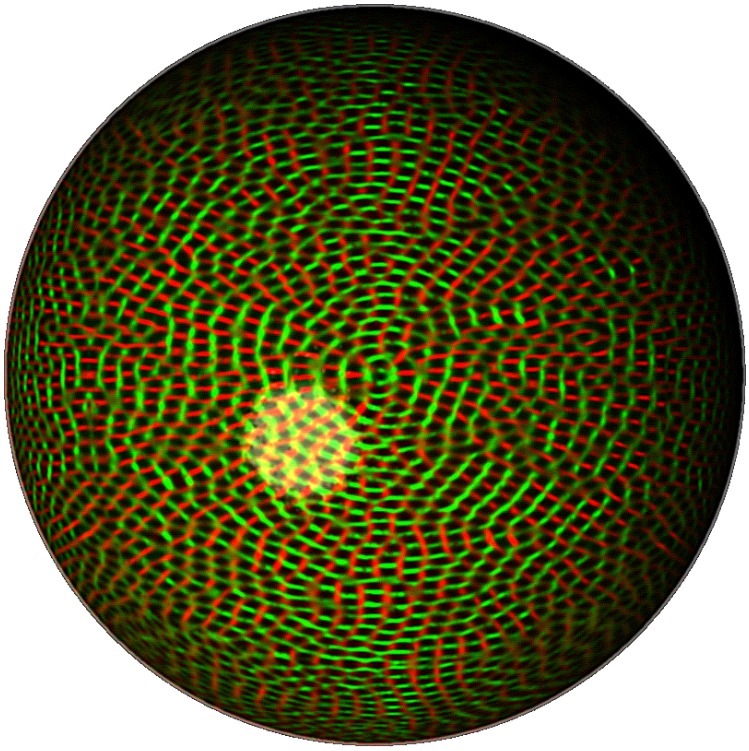
7.4 Results

In the last section, we introduced our extension of TensorMesh to improve visual quality and increase structural perception of the represented tensor data. It is a fast screen space approach, similar to the one introduced by Hotz et al. [83] and used ideas from [104] to transform the algorithm into screen space.

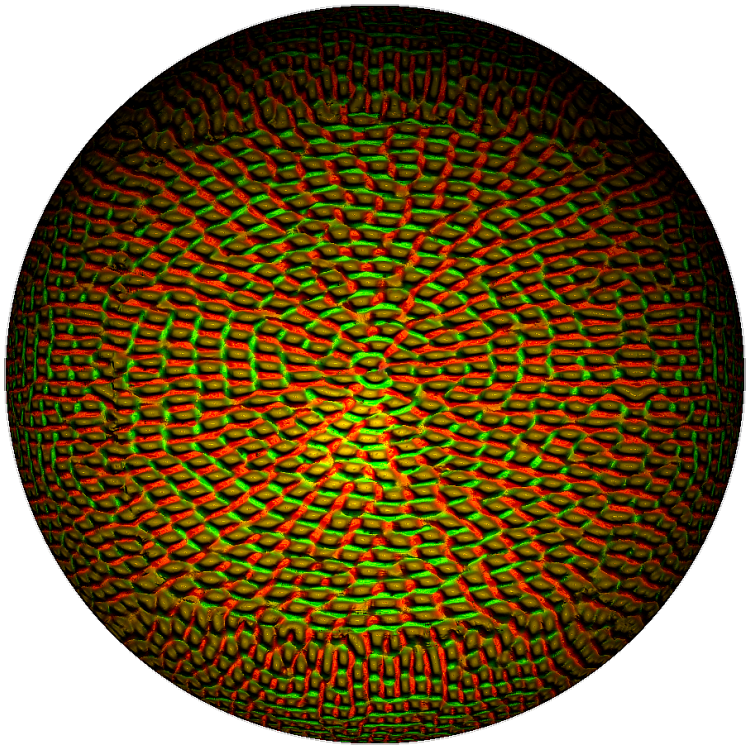
Our implementation, using this method, was able to reach frame rates high enough for real-time user interaction. The only bottleneck is the hardware's ability to render large and triangle-rich geometry. All further steps can be done in constant time. This section shows several datasets, rendered with our improved TensorMesh method. We compare it to the original method and show the computational overhead we introduced.

Artificial Test Data We calculated a spherical test dataset as scalar volume. We used an isosurface generated using the marching cubes algorithm [113]. The Laplacian of the spherical scalar field is used as second-order tensor on the surface. The result is displayed in Figure 7.9 and compares the original TensorMesh with the improved version we introduced here. Figure 7.9(b) provides a hugely improved structural perception of the mesh structure.

Medical Data Even though many higher-order methods have been proposed, due to scanner, time, and cost limitations, second-order tensor data is still dominant in clinical application. Medical second-order diffusion tensor datasets differ from engineering datasets 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 datasets, where x, y, and z alignment are displayed in red, green, and blue, respectively. This coloring is often called *local directional coloring* [145] in medical applications. In areas, where a secondary direction in the plane exists, we display this in gray-scale. Figure 7.10 demonstrates this and, again, compares the original TensorMesh method with our streamtube improvement. When comparing both renderings, the streamtube improvements create a more crisp and less blurry result.

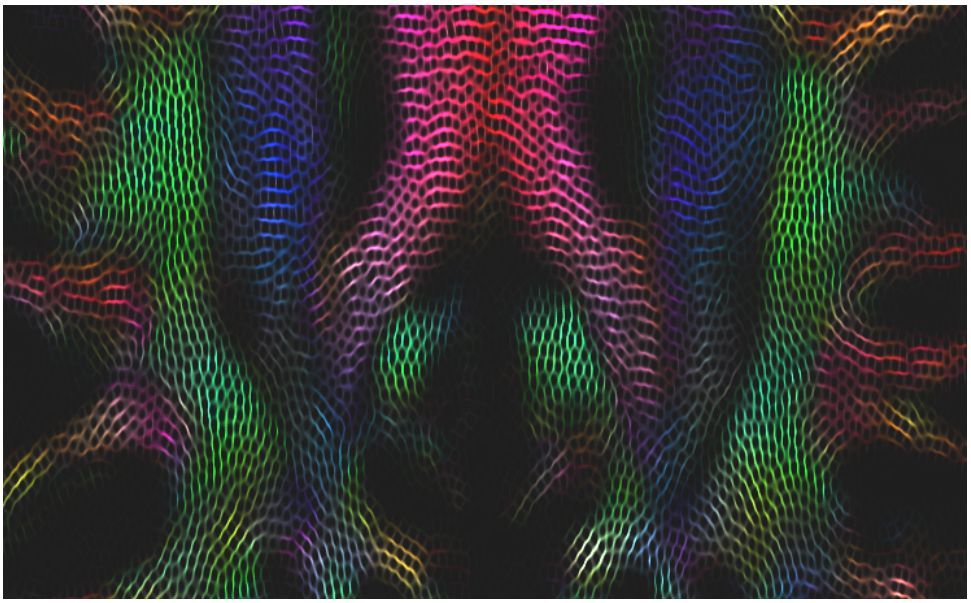


(a) Original TensorMesh

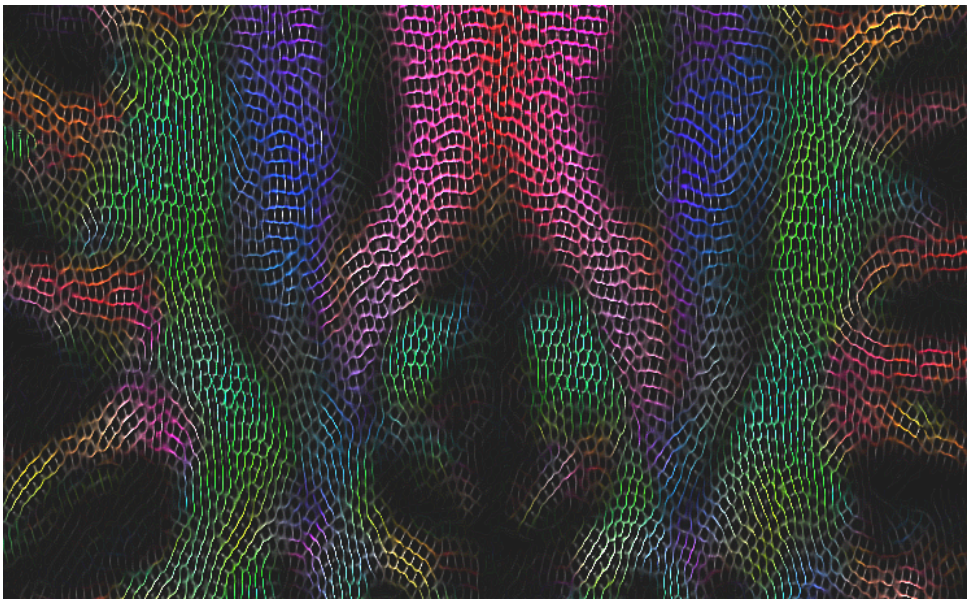


(b) Bump mapping on TensorMesh

Figure 7.9: *TensorMesh applied to a spherical test dataset. We applied our method to an isosurface and the scalar field's Laplacian to demonstrate the visual difference between the original TensorMesh (top) and our improved version introduced here (bottom).*

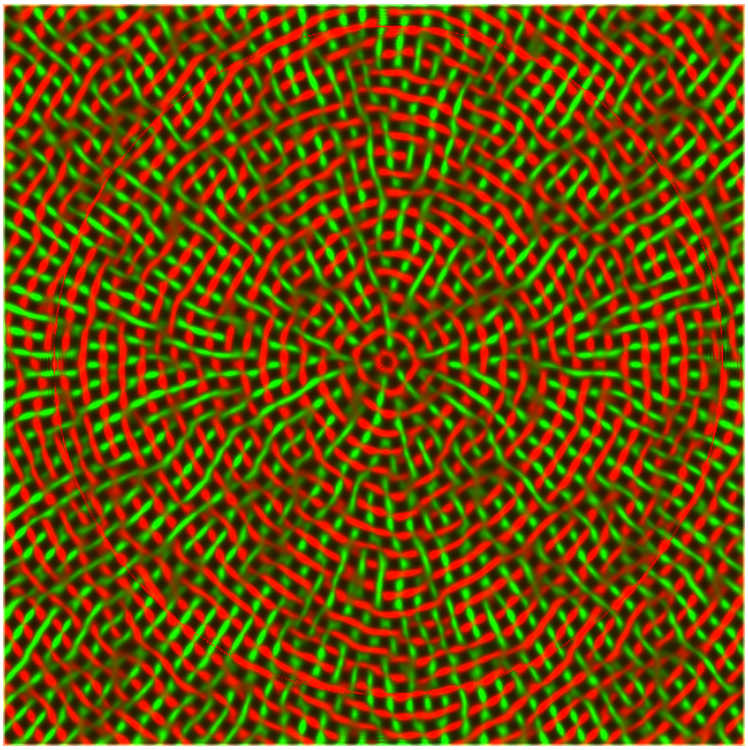


(a) Original TensorMesh

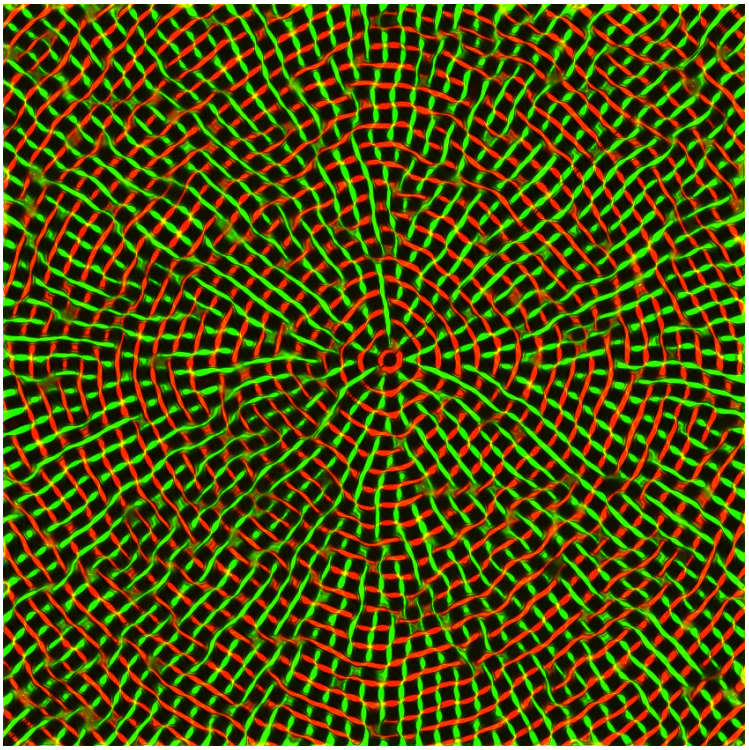


(b) Streamtube improvement

Figure 7.10: *An axial slice through a human brain: Corpus callosum (CC) (red), pyramidal tract (blue), and parts of the cinguli (green in front and behind the CC) are visible. The main direction in three-dimensional space is indicated by the RGB colormap, where red indicates lateral (left–right), green anterior–posterior, and blue superior–inferior direction. The left–right structure of the CC can clearly be seen in its center, whereas color and pattern indicate uncertainty towards the outer parts. The same is true for the cinguli's anterior–posterior structure. As seen from the blue color, the pyramidal tract is almost perpendicular to the chosen plane and, therefore, secondary and ternary eigenvectors dominate the visualization. Alternatively, we could easily fade out those out-of-plane structures in cases, where they distract the user. Please note that we have applied a contrast enhancement filter on both images to improve the print quality.*



(a) Original TensorMesh



(b) Bump mapping on TensorMesh

Figure 7.11: *A slice in the well known single point load data set, showing the symmetric strain tensor at the surface of the slice.*

In Figures 5.15 and 5.16 of Chapter 5, we also applied the bump mapping scheme to a screen space based LIC of an electrical field on the skull. The figure shows two different projection angles causing a noisy and a very LIC-like result. Due to bump mapping, the structure of the LIC noise is very clear and the contrast between ridges and valleys is high. It is possible to defer the shading of the surface itself to the shading of the structure on the surface. This way, the surface's shading does not influence the contrast of the LIC result on it. This already shows that screen space postprocessing is not only useful for TensorMesh, but also for other surface-based visualizations.

Mechanical Datasets Our approach is not only applicable to medical datasets, but it can also be applied to many other kinds of tensor data. Figure 7.11 shows a slice in an analytical strain tensor field. The analytical dataset is the well known single point load dataset, where a single, infinitesimally small point source pushes on an infinite surface. The forces and distortions inside the object are represented by stress and strain tensors, which are symmetric, second-order tensors. Similar to the above examples, we compare the original method with the bump mapping improvement. The visual quality and the perception of the mesh structure is tremendously improved.

7.4.1 Performance

As the major part of the TensorMesh method works in screen space, it is mostly independent of the input data complexity. As indicated before, the only "bottleneck" in the visualization pipeline is the strongly geometry-dependent projection step. But in practice, this is not that critical, since the rendering time of geometry is determined by the GPU itself. More interesting is the overhead the TensorMesh method adds on top.

Theoretical View and Expectations When analyzing Figures 7.2 and 7.6, as well as the working principles of each of the steps, it gets obvious that the advection step might be the most GPU intensive part. From the GPU's point of view, the silhouette detection and compositing steps do nothing else than accessing the bound input texture locally and blending RGBA quadruples/RGB triples — the hardware implemented core functions of a GPU. These steps do not need branching nor do they access textures non-locally. This avoids waiting shaders in a group and cache misses in the shader local memory. In contrast, the advection step uses branching (if-condition) to check whether a

Postprocessing	Frames per second
None	350
Bump Mapping	341
Streamtubes	311

Table 7.1: *Frames per second (FPS) with different postprocessings for the strain tensor data in Figure 7.11. It is obvious that the method works in real-time and that the postprocessings add only a minor overhead.*

silhouette is reached. Additionally, depending on the number of iterations, it might leave the local texture memory. The effect can be tested easily when increasing the number of iterations per frame. Per default, we use ten iterations. The same is true for the postprocessing step; especially the streamtube effect, as it samples the surroundings of the current pixel. However, the number of samples needed to find the tube boundaries is low, as the tubes are usually only several pixels in diameter. Additionally, the postprocessing and advection steps discard pixels early. This leads to performance gain in advection and postprocessing.

Measuring Performance As screen space approaches depend on the amount of pixels covered, we use the rendering in Figure 7.11 as an example. They cover the screen completely. The measurements were done on an Intel Core i7 CPU at 3.33GHz and 24GB RAM with a NVidia GeForce GTX Titan. As usual, the CPU does not play an important role for screen space methods. The images were rendered in a resolution of 1080×1080.

Table 7.1 shows the measured FPS values for different postprocessings. The frame rates for the other result images we have shown are similar, since they cover a similar amount of screen space and their geometric complexity is also very low. Hence, their exact values are not relevant here. In my original diploma thesis, it was shown that the TensorMesh method works in real-time with a NVidia GeForce 8800 GTS. Today, this can be seen as low end GPU.

Table 7.2 validates the above theoretical assumption that an increasing iteration count for the advection step will cause a tremendous drop in performance. Similarly, when zooming into a streamtube rendering, as done in Figure 7.12, the amount of samples to find the tube borders increases. This causes a frame rate drop comparable to the iteration count. As the advection step *and* the streamtube postprocessor use a step size of one pixel, a tube ra-

Iterations	Frames per second
10	350
20	230
30	180
100	70
150	58

Table 7.2: *Frames per second (FPS) with different iteration counts in the advection step for the strain tensor data in Figure 7.11. The frame rates drop fast, when increasing the iteration count.*

dius of 100 pixel will cause a frame rate drop to 20% of the original streamtube rate (cf. Table 7.2).

However, in practice, these high iteration counts play no important role. For the above images, we used an iteration count of ten, yielding interactive results. Additionally, zooming into a TensorMesh rendering should increase the resolution of the noise on the surface, showing more details and ensures thin tubes.

7.5 Discussion

7.5.1 Limitations and Problems

Projection to a Surface

Whether the surface itself is the domain of the data, a surface defined on the tensor information (e.g., hyper streamsurfaces), 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, because only in-plane information is visualized. In Figure 5.16, the effect of different maximum projection angles for LIC is shown. Projecting nearly perpendicular vectors to the surface yields questionable results.

To overcome 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.

Artifacts

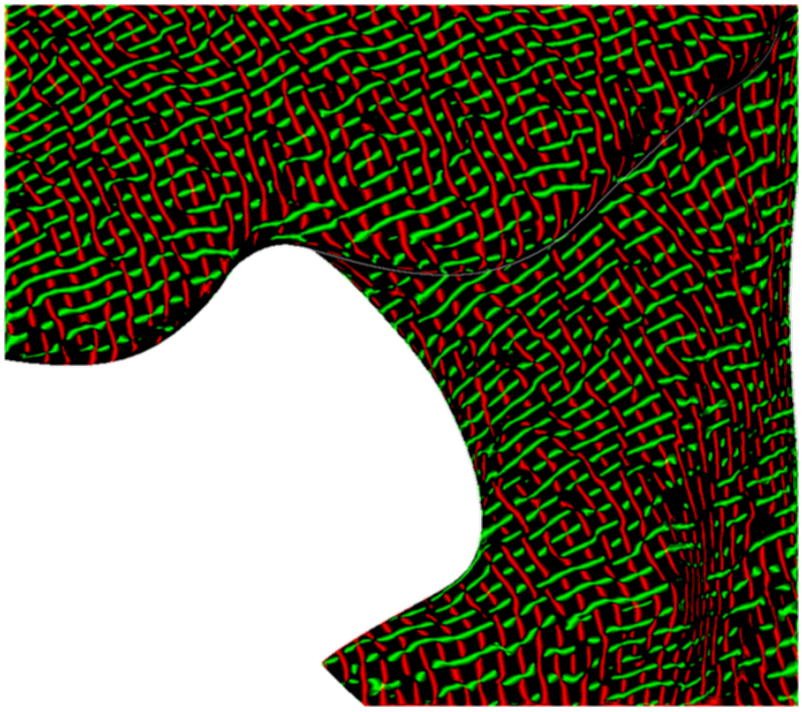
As screen space methods work with a super- or sub-sampled version of the original data, they might suffer artifacts introduced by interpolation or skipped data points. Figure 7.9 already indicates this. The green mesh on the sphere's surface does not look that circular. This is due to the resolution of the mapped noise and sub-sampling near the visible center, spreading out the available data. Respectively, the data at the visible border of the sphere is very compressed due to the projection. This kind of limitation was also present in the original TensorMesh and cannot be solved easily.

Another type of artifact is introduced by the streamtube postprocessor. The artifacts can be seen in Figure 7.8, where we show a zoomed part of the original rendering. As we do not integrate along the whole eigenvector field, there may be discontinuities along a tube in the produced image. There are also artifacts caused by a blurry input field, where borders cannot be found clearly. But, since the frequency of the fabric structure is normally much higher, and the tubes are not that large, these artifacts vanish and play no important role, as shown in Figure 7.12(b).

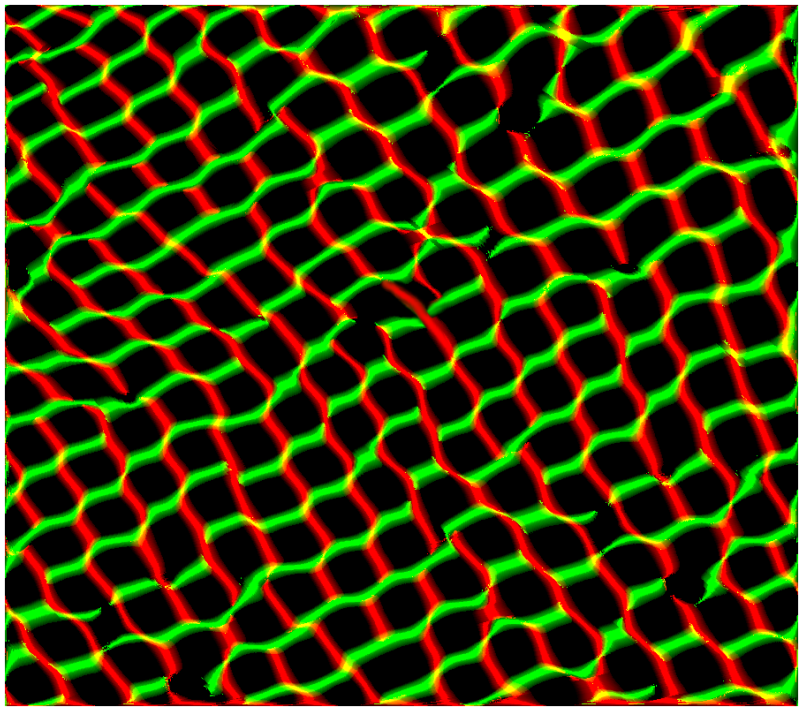
Resolution

A general problem TensorMesh has in common with many other methods is the resolution limitation caused by the *structural density* of the visualization. This means, the maximally visible frequency of the mesh structure limits the resolution of the visualized data. The problem gets especially evident when considering it in the context of projection, as mentioned above. The resolution of the data mapped to a certain area on screen changes when changing the view on the scene. Important details of the data might get projected to an area on screen, whose resolution is smaller than the structural TensorMesh resolution.

Approaches to preliminarily find interesting areas and highlighting them, are no solution to this problem, as the interesting areas still might be too small with respect to their projected area on screen. Instead, possible solutions would include focus and context techniques, where the level of detail is adapted according to zooming or by using virtual lenses.



(a) Tube effect



(b) Artifacts when zoomed in

Figure 7.12: *These renderings show a zoomed part (bottom) of the streamtube postprocessor effect from Section 7.3.1. Although the artifacts in the tubes are not visible in the top image, they get visible when zooming in. These artifacts are caused by discontinuities during advection and the partially blurry input images, calculated by the original TensorMesh method.*

7.5.2 Future Work

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 (cf. Schultz and Seidel [179]). Whereas the number of output textures can easily be adapted in our setup, the major remaining problem is a lack of suitable decomposition algorithms on the GPU. Screen space techniques, by their very nature, resample the data and, therefore, require one to use proper interpolation schemes. In addition, maintaining orientations and assigning same fibers in higher-order data to the same texture globally is not possible today and, therefore, is a potential topic for further investigation.

An important future step is to evaluate the shown methods with scientists of several fields to measure the improvements and to find possible issues critical to a field's scientist.

7.6 Conclusion

The original TensorMesh method is a fast, GPU-based second-order tensor visualization, inspired by the PBM technique of Hotz et al. [83]. It is very well suited to visualize tensorial data in mechanical and medical data by utilizing the structural perception capabilities of the human vision apparatus. Unfortunately, the results of the original TensorMesh approach where rather blurry, hence the perception of structure was often suboptimal.

We have presented a useful and reasonable extension to the original TensorMesh method. We have shown the improvement of ***structural perception*** of this mesh-like tensor visualization and made the results crisp and clear. With this, the tensor field structure can be grasped even better.

We have shown that computer game methods and other postprocessing effects can help to improve visualization techniques. They do not only create visually appealing images, which often is frown upon in science, but also improve perception. Of course they are not able to negate issues inherent to a certain method, like the sampling and projection issue mentioned above.

During my PhD, the experiments done with TensorMesh were the "igniting spark" to engage myself in computer graphics and screen space methods

to improve the spatial and structural perception of existing visualization techniques.

In this chapter, we have shown how to improve structural perception of data represented *on* surfaces, while the next chapters focus on spatiality and structural perception at different scales for line and point data.

8

LineAO – Improved Three-Dimensional Line Rendering

This chapter is based on the following publications:

[P9] – S. EICHELBAUM, M. HLAWITSCHKA, and G. SCHEUERMANN. **LineAO – Improved Three-Dimensional Line Rendering**. *IEEE Transactions on Visualization and Computer Graphics* 19.3 (2013), 433–445
Online: http://sebastian-eichelbaum.de/pub13a

[P10] – S. EICHELBAUM, M. HLAWITSCHKA, and G. SCHEUERMANN. **Vue en tractographie d'un cerveau humain.** *LeMonde Science Online, 2012: la science en images*. 2012
Online: http://sebastian-eichelbaum.de/pub12a

[P11] – S. EICHELBAUM, J. KASTEN, M. HLAWITSCHKA, G. SCHEUERMANN, and B. R. NOACK. **Leading edge vortices of flow over a delta wing**. *Gallery of Fluid Motion, Poster, P55*. 2012
Online: http://sebastian-eichelbaum.de/pub12b

8.1 Overview

The Data: Lines In many areas of visualization, line rendering techniques play an important role. In flow visualization, three-dimensional vector fields are visualized using streamlines, streaklines, or pathlines. They have a direct physical meaning and are used to understand simulated and measured data in many parts of engineering. Lines are used to represent electric and magnetic fields as well as velocity fields in a very intuitive way. Even in tensor fields, where a line tangent is not given explicitly, tensor lines and hyperstreamlines [42, 157] are used to display important structures of the field.

But vector and tensor fields do not only play a crucial role in flow visualization. In medical visualization, a large variety of measuring and imaging methods, such as electroencephalograhy (EEG) or magnetic resonance tomography (MRT), are available. From EEG, it is possible to derive the describing electric field in the head and represent it using field lines. Additionally, field line representations for simulations of different head and tissue models allow a better understanding of the underlying models and parameter influences, as shown in Chapter 5.

Using diffusion tensor imaging (DTI) or high angular-resolution diffusion imaging (HARDI), it is possible to coarsely reconstruct the neuronal connections inside the brain and to obtain a better understanding of its structure. These techniques are called fiber tracking or tractography [9, 15, 78, 134, 179] and are often based on enhanced streamline techniques.

In general, lines are very well suited for visualizing directional information on a global scope. The above mentioned techniques and imaging approaches are only examples of the myriad of use cases, where line data plays an important role. Accordingly, visualization of line data is crucial to understand the intrinsic properties of a huge variety of real-world phenomena and simulated scenarios.

Visualization of Line Data Each kind of line data, each visualization technique, and each scientific use case has its own specific properties and constraints. Mostly, large and dense line data is given and the structural relation of bundles of lines as well as local shape information is crucial for understanding the represented structures. For exploring this kind of data, filtering and rendering of line data in real-time is an important requirement for modern visualization techniques. Besides this, consistency of rendered images under

modification and interaction with the data is important to retain the mental image of its structure.

Current line data rendering approaches usually employ local illumination models for lines to emphasize the shape of lines and line bundles [117, 238]. These techniques apply a simplified Blinn-Phong [18] shading, which provides diffuse reflection and specular highlights and allows depiction of local shape features. A hurdle to overcome with local illumination is the definition of a proper normal for lines. As there are endlessly much perpendicular vectors at each point of a line, Mallo et al. [117] uses the vector whose angle to the camera vector is the smallest. Section 8.3.2 explains this in more detail.

An alternative approach is to create the look of real cylindrical tubes by rendering line data using quad strips [173, 198] or triangle strips [128]. These approaches allow correct lighting (according to Phong's lighting model) and keep the density of the rendered lines while zooming, which often is a requirement.

Besides these shading techniques, there are approaches which utilize depth cueing and haloing to provide further structural information. In [52], depth-dependent halos are rendered around lines to emphasize tight bundles of lines. The additional depth cueing further increases depth perception in this method. Unfortunately, due to the heavy overlapping of halos, this type of depth cueing loses its effect if the line data is very dense.

Another approach is to interpret dense line data as volumetric data. Schussman and Ma propose a method for sampling extremely dense line data and rendering them with direct volume rendering [180]. Unfortunately, this method does not focus on spatial perception in the final volume rendering.

Hair rendering is another shading technique for dense line data, which relies on simplifications implied by their underlying model: All hair rendering techniques are optimized to mimic the effects of light scattering in a multitude of thin, translucent hair without caring about the exact shading of each single strand of hair. Therefore, most of the geometric simplifications that make hair rendering efficient cannot be used in visualization since, in scientific data, each single line's shading is important. For a comprehensive overview and up-to-date hair rendering techniques, we refer to [151, 216, 229, 230, 231].

The Problem Although most current approaches are able to depict local shape of line data, they are not able to properly represent global, spatial relations and the local structure in bundles of dense line data at the same time.

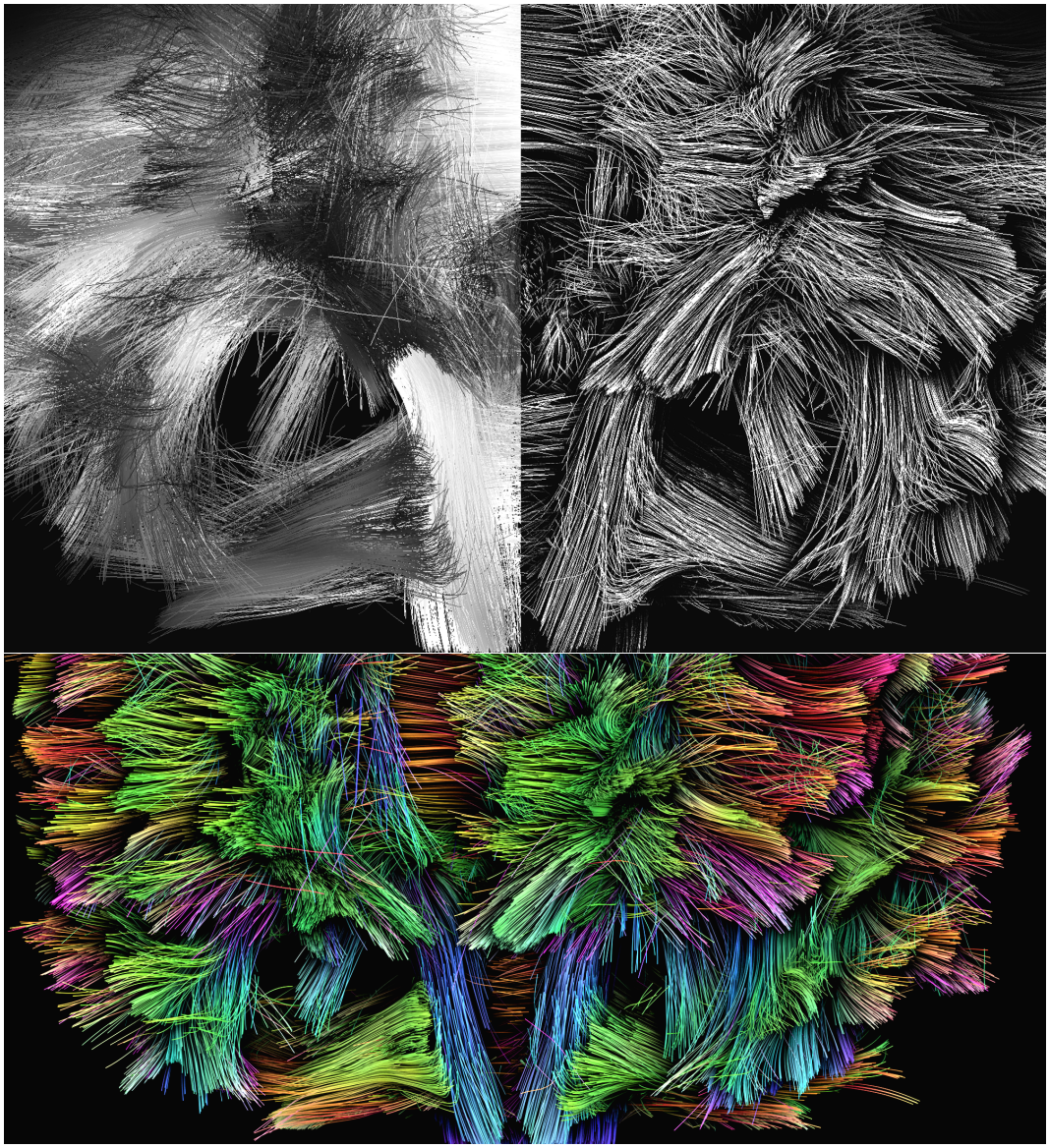


Figure 8.1: *Fiber tractography rendered using illuminated lines [117] (left) versus LineAO (right) and LineAO colored according to the local tangent direction [145] below. The improved perception of spatial relations between and in bundles of lines can be seen especially well in the brain stem (center bottom part of the image), where the* Pons *and* Medulla Oblongata *pass into the* Spinal Cord*. The layering of these bundles as well as the fissure structure in the* Frontal Lobe *can be observed very distinctly. This was not possible before and the fact that LineAO works in real-time, without any precomputations, makes it a perfect addition to nearly every line-based visualization approach.*

Our Solution: LineAO With LineAO, we contribute a novel approach, which overcomes these problems and provides a greatly ***improved structural and spatial perception*** for the rendered line data in a very intuitive and natural way, as demonstrated in Figure 8.1. It uses ideas from ambient occlusion and global illumination to allow a ***simultaneous depiction of local and global line structures***. It is known that global lighting effects are very important for determining an object's position and spatial relations [103, 155, 214, 215]. Since real-time ability and dynamic scenes are a major demand in many fields, our method ***renders in real-time, without precomputation*** and is, therefore, capable of being used in explorative tools, where the researcher interactively modifies the line data. We combine global ambient lighting with the scattered light contributions from surrounding lines and adhere to the intrinsic fixed line width on screen, ensuring ***consistency under modification and interaction***. LineAO is a computer graphics method. As such, it ***can be applied to any kind of line-based visualization***, independent of the underlying type of imaging modality, simulation type, or measurement method.

The next section, will introduce the principles of ambient occlusion (AO), the underlying theoretical model for LineAO. It explains different practical approaches to implement the AO model and why they are not applicable to line data. Section 8.3 then introduces the LineAO scheme and its transition to screen space on a theoretical basis, followed by practically relevant implementation details. The chapter closes with several visualizations a detailed discussion on LineAO.

8.2 Background

8.2.1 Ambient Occlusion

The term *ambient occlusion*, or AO for short, refers to a group of algorithms that can be seen as a crude approximation of global illumination. In general, the term *global illumination* refers to computer graphics algorithms, which include global effects on the illumination of an object in the scene. These effects might be shadows, reflections, refraction, absorption in volumetric objects, ambient light, and others.

Roughly spoken, AO represents the diffuse lighting effect on a day with overcast sky. This ambient light is a very complex and globally defined problem, since the whole scene defines the distribution of ambient light. For this

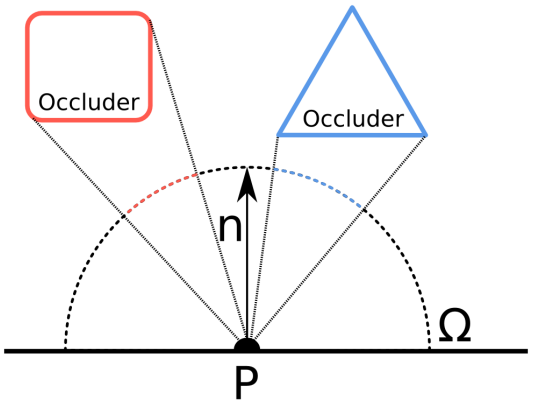


Figure 8.2: *Illustration of Equation (8.1). Each geometry in the scene can occlude a part (red and blue) of the unit hemisphere around P. This fraction weighted by the relative direction to the surface normal describes the ambient occlusion of the small surface element represented by the normal n at point P.*

reason, real-time computer graphics was using only direct illumination for a long time, where the ambient light is assumed to be constant at every point in the scene. AO estimates the ambient light distribution in a complex scene to imitate radiance of light on non-reflective surfaces. This increases the realism of computer generated images and improves the perception of relations between objects [103, 155, 214, 215].

Model of Ambient Occlusion

The AO factor describes the amount of ambient background light not reaching the surface. In other words, it determines, how much of a surface is concealed by other surfaces, prohibiting ambient light to reach the surface. To describe this mathematically, we locally define a surface using its tangential plane at point P with surface normal n. To measure the amount of occlusion for the point P, it is required to measure the surface area of an unit hemisphere occluded by surrounding objects as demonstrated in Figure 8.2. Mathematically, this surface area is defined by the integral over the unit hemisphere. The actual check whether a point on the hemisphere is occluded or not is done by a binary visibility function $V(\omega, P)$, being 1 if the surface point is visible, and 0 if it is occluded. The unit vector ω hereby samples the unit hemisphere by pointing from P to the surface point of the hemisphere Ω.

The amount of light energy reaching a point on the surface is defined by the angle between the light direction and the surface's normal. Using this, AO defines the amount of skylight energy *not* reaching P due to the occlusion.

This yields the standard ambient occlusion definition used in literature [7, 123, 158, 183]

$$AO(P,n) = \frac{1}{\pi}\int_{\Omega}(1 - V(\omega, P))\langle\omega, n\rangle d\omega, \tag{8.1}$$

for point P on a surface and its normal n. Due to the influence of the scene in the visibility function, it is obvious that there is no easy analytical solution to the integral. Therefore, it is usually approximated and, for reasonable setups of the geometry, can be approximated by the Riemann sum

$$AO(P,n) = \frac{1}{\pi}\lim_{s\to\infty}\sum_{i=1}^{s}(1 - V(\omega_i, P))\langle\omega_i, n\rangle\frac{\pi}{s}, \tag{8.2}$$

and truncated to a series of s samples

$$AO(P,n) \approx AO_s(P,n) = \frac{1}{s}\sum_{i=1}^{s}(1 - V(\omega_i, P))\langle\omega_i, n\rangle, \tag{8.3}$$

which approximates Equation (8.1) for a sufficiently high sample count s and a well chosen distribution of $\omega_i \in \Omega$ on the unit hemisphere.

When replacing the binary visibility function $1 - V(\omega, P)$ with a magnitude function $\rho(\omega_i, P)$, which gives the amount of light reflected from the first occluder in direction ω_i from P, Equation (8.1) would be called *obscurance* of P. This can be seen as an extension of AO in a way that it also includes refracted ambient light from other objects in the scene. This is usually used to create color bleeding effects.

Current Ambient Occlusion Techniques

To solve the above equation for complex scenes, many approximative algorithms have evolved. Their main goal is to efficiently evaluate the visibility function V for complex and large scenes. Thereby, these methods can be classified in ray-tracing approaches, which try to approximate the AO effect on a physical basis and real-time approaches which try to achieve the AO effect phenomenologically. In this section, we give an overview on these methods and their drawbacks regarding line rendering.

Ray-tracing approaches often use proxy geometry to reflect the rather complex scene with simpler primitives. This way, occluders can be described and tested faster by the visibility function V. These proxy primitives are often analytic objects such as spheres or discs [7, 10, 27, 81, 159, 237] or more complex primitives that better match the given geometry [21, 213]. These approaches

often include heavy precomputation steps and produce an over-estimation of the AO effect due to the approximative scene description. Due to the precalculation step, these methods are mostly limited to static scenes and only allow limited interaction and interactivity, which is why we do not further consider them.

Unlike physically correct techniques, the class of phenomenological approaches is able to run in real-time. The cornerstone of most of these techniques is the image enhancement using unsharp masking of the depth buffer, which has been presented by Luft et al. [115]. In their method, the depth buffer of the rendered scene is interpreted as height field and compared with a low-pass filtered copy. This is utilized to provide information about spatially important areas, which then allows a modification of several image properties such as local contrast. Although this is no real ambient occlusion effect, it provides a remarkable improvement in spatial perception and spawned a whole set of methods grouped under the term screen space ambient occlusion (SSAO). A widely known SSAO technique is CryTec SSAO [90, 131]. It adapts unsharp masking and uses sparse sampling of the depth buffer to derive visibility information in the neighborhood of a point in the scene, based on information available in screen space. It uses the depth information in the neighborhood of a pixel to estimate the amount of ambient light that reaches the corresponding point on the surface. Due to the limited resolution in screen space, this approach samples the ambient occlusion factor at a low angular resolution, leading to inaccurate, results especially for small or distant objects. It can be extended by using better sampling schemes [12, 55] or by adding better filtering and distant occluders [183]. Two of the main disadvantages of these methods are the low spatial resolution and the noise induced due to the stochastic sampling scheme. Hoang and Low [79] introduced a multi-resolution approach to combine the AO effects of distant objects with detailed local AO.

To circumvent the spatial resolution problem, hybrid approaches have been developed that combine ray-tracing a simplified scene with SSAO [51, 158]. Other methods precompute additional fields or acceleration structures abstracting the occlusion caused by objects [96, 101, 118] or precalculate the transfer of incident light from an environment map into the incident radiance on the surfaces [88, 186, 187]. Unfortunately, these methods suffer in being constrained to static scenes or require precomputation steps.

Each of the above-mentioned techniques has its limitations towards some of the render properties we mentioned in Section 8.1. Physically based approaches

fail to provide real-time rendering or rely on heavy precomputation steps. As these techniques aim at physically correct shading, they cannot emphasize structure in line data using a physically incorrect but detail-emphasizing, illustrative shading.

Phenomenological approaches work well in compact scenes with objects with a certain minimal volume, due to the fixed sampling radii and low spatial resolution. The low spatial resolution of these approaches prohibit the proper shading of small and thin structures in line data due to aliasing. Although the multi-resolution SSAO approach [79] can solve the problem of low spatial resolution by sampling at multiple scales, it reaches its limits when applied to thin structures as stated by Hoang and Low [80]. It does not modify the AO approach itself and thus, does not incorporate any special obscurance weight that allows for emphasizing global structures while retaining the local detail in these line bundles. Additionally, phenomenological approaches often are not able to create coherent renderings when zooming, as line bundles get less dense when zooming. This is due to the intrinsic fixed line width on the screen, which stays constant during zoom.

8.2.2 Ambient Occlusion in Visualization

Global illumination techniques and ambient occlusion have found their way into more and more scientific visualization techniques recently. When considering that the well known direct volume rendering (DVR) technique is about the practical implementation of optical models for light absorption and emission in volumes, it gets obvious that it was naturally predestined for incorporating ideas from global illumination into its underlying light models. For a comprehensive introduction to DVR and its underlying theoretical background, please refer to Engel et al. [50]. In 1995, Max [121] published a survey on different optical models in DVR and also handled a topic called *obscurance*. Obscurance describes the influence of the vicinity of a point in space onto the points lighting properties [237]. This was first used in DVR by Stewart [197], and called *vicinity shading*, where the illumination of a sample point in DVR was also influenced by the attenuation of light caused by surrounding voxels.

Later, a huge amount of techniques focusing on real-time global illumination of direct volume rendering (DVR) were introduced. This was mainly driven by the increasing computational power of modern graphics hardware and the established understanding, that global illumination and occlusion-based shading can help to provide the required cues to keep track of the spatial

relations in the rendered images. Ruiz et al. [165] used the idea of obscurance to provide realistic volume renderings and in 2009, Schott et al. introduced a fast approach, which is able to handle transparent structures in a volume. This was done by tracking the amount of light reaching each slice in the volume, using a directed, cone-shaped phase function. Other methods do not only focus on light attenuation in a volume, but also incorporate dynamic illumination effects [98, 189]. For further details on direct volume rendering and other depth enhancement techniques, please refer to the literature, especially Preim and Botha [153] as well as Engel et al. [50].

Not only DVR profited from the increasing acceptance of several global illumination techniques. In 2006, Melek et al. [127] and Wyman et al. [228] introduced global shading approaches for solid, triangle-based geometry. They show the advantages of proper, global shading for isosurfaces [228] and thread-like microscopy images. Later, methods were introduced to combine surface-based global shading with volumetric shading [176].

Especially in medical applications, these techniques help to understand structure and relations in three-dimensional anatomical data. Besides the medical use case, chemical and bio-chemical visualization also profits from more realistic rendering and, e.g., [201], [99], and [65] added ambient occlusion for a better structural perception of very complex molecule structures. Gribble and Parker [64] applied ambient occlusion to particle rendering and investigated its effect in a formal user study.

8.3 Method

In the previous sections, we have summarized the state of the art in line rendering, introduced the mathematical fundamentals behind AO, and given an overview on available approaches for ambient occlusion rendering.

In this section, we introduce LineAO and show how LineAO uses the intrinsic fixed line width on screen for visibility evaluation while ensuring consistency under zoom and interaction. We introduce a sampling scheme to solve the problem of low spatial resolution in screen space and provide an obscurance weight tailored towards line rendering to furthermore emphasize local detail in bundles while retaining global structural shading. It prohibits heavy darkening of local structures due to large, global structures. We finally provide a pragmatic implementation guideline and show how we combine LineAO with illuminated lines and tube-based rendering.

8.3.1 LineAO Sampling Scheme

As we aim at a solution in screen space, we need to handle the problem of low spatial resolution, common to nearly all SSAO approaches. To achieve this, sampling on a single hemisphere is not sufficient. Occluders inside the hemisphere will be missed, as well as occluders far away from the hemisphere. We need to sample near and distant lines and classify them into levels of distance. We, therefore, begin to extend the sampling scheme, weighting, and view evaluation in Equation (8.3) to increase spatial resolution, while tailoring it towards correct handling of local and distant occluders. First, we replace the orientation-based weighting, with a custom obscurance weight g, which attenuates the occlusion term $1 - V$:

$$AO_s(P) = \frac{1}{s} \sum_{i=1}^{s} \left[(1 - V(\omega_i, P)) g(\omega_i, P) \right]. \tag{8.4}$$

To additionally allow evaluation of AO on multiple hemispheres, we include a parameter r, defining the radius of the hemisphere used to sample the surrounding objects. We call

$$AO_s(P, r) = \frac{1}{s} \sum_{i=1}^{s} \left[(1 - V(r\omega_i, P)) g(r\omega_i, P) \right] \tag{8.5}$$

the local AO for a hemisphere with the radius r. Although the weighting function g and the hemisphere radius r now allow the combination of AO for multiple hemispheres, the radius is not sufficient for classifying distance levels and, therefore, the different effects of local and global occluders to the current point P. To accommodate this classification issue, we modify the visibility and weighting function to depend on a parameter l, defining the level of distance. The smaller l, the more local detail is emphasized. The larger, the more global structures are important. It allows us to handle local and distant occluders properly using V and g. This defines an ambient occlusion term for a given distance level l, which accommodates for the different properties of distant and near occluders:

$$AO_{s,l}(P, r) = \frac{1}{s} \sum_{i=1}^{s} \left[(1 - V_l(r\omega_i, P)) g_l(r\omega_i, P) \right]. \tag{8.6}$$

This yields the final evaluation of the AO fraction of one hemisphere with the radius r for a point P with s_h samples and distance level l. In contrast to other approaches, we are able to handle distant occluders and local structure

directly with our specialized obscurance weighting function g, without the use of proxy geometry as, e.g., in [183].

With Equation (8.6), we are now able to classify each occluder into distance levels and to combine their AO effect using

$$\text{LineAO}_{s_r,s_h,r_0}(P) = \sum_{j=0}^{s_r-1} AO_{\frac{s_h}{j+1},j}(P, r_0 z \cdot (j^2 + j)). \tag{8.7}$$

LineAO evaluates Equation (8.6) s_r times for increasing sampling hemisphere radii and distance level l. The first iteration $j = 0$ represents the smallest hemisphere, which is responsible for the local details, and uses s_h samples on the hemisphere. For the following iterations, the term $\frac{s_h}{j+1}$, ensures that the number of samples per hemisphere is reduced. As LineAO sums up the occlusion effects of near and distant occluders and ignores the possibly overlapping occlusions on different hemispheres, it generates the intended effect of heavier occlusion in very dense areas. The term $r_0 z \cdot (j^2 + j)$ hereby defines the increasing hemisphere radius for increasing distance levels. The zoom level z, represents the relation between the supposed line volume in eye space and the *intrinsic fixed line width* on screen and, therefore, creates a coherent AO effect when zooming. The next section provides a definition for z in screen space. Since j is zero for the first level, $r_0 z$ defines the smallest hemisphere for sampling in the direct vicinity of a line. Finally, LineAO is clamped to the interval $[0, 1]$.

In this section, we have presented our sampling scheme, which handles the problem of low spatial resolution. With this, we are able to combine the advantages of local detail with perception of global structure in line data, due to an adaptive spatial resolution, depending on the distance level. To achieve the goals mentioned in Section 8.1, namely emphasizing local detail and global structure of line data simultaneously, we need to define a weighting function g tailored towards this. In the upcoming sections, we show how LineAO is efficiently evaluated by providing the definition of the zoom and visibility function in screen space as well as a proper weighting function g to ensure correct handling of thin structures locally and globally.

8.3.2 LineAO Evaluation in Screen Space

Until now, the LineAO sampling scheme was described in a general form. Admittedly, an evaluation in eye space is not feasible, if real-time rendering is

required. To solve this problem, we transfer the LineAO algorithm to screen space. We therefore evaluate Equation (8.7) as a function of each pixel P in the rendered line dataset. We, therefore, replace the notation of *point* with *pixel* and provide a visibility function V and a special weighting function g for emphasizing high-frequency detail in narrow line structures and low-frequency features in a global context.

As these functions work in screen space, they allow simplifications in visibility testing and weighting, which are not feasible in eye space. For an introduction to the modern graphics pipeline and screen space rendering, please refer to Section 6.2.

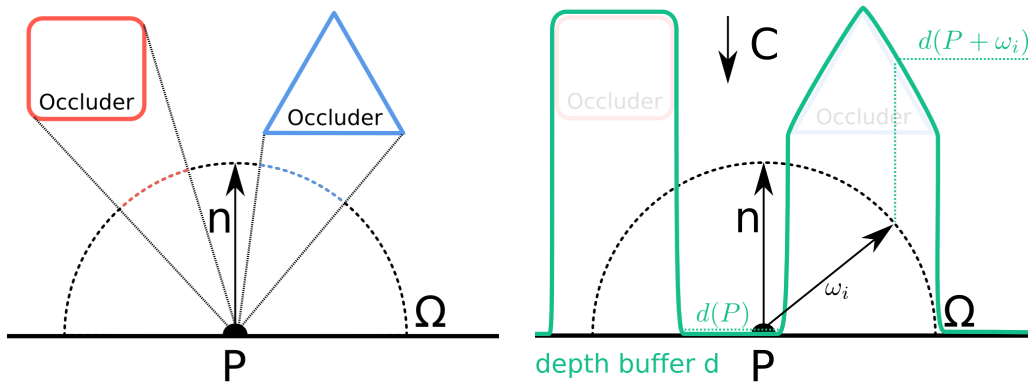
For illustrating the different effects of certain parameters and functions, we use an artificial spiral line dataset. It combines dense line bundles as well as global overlapping of bundles at different distances.

Conceptual Overview

In LineAO, we follow the widely used concept of interpreting the depth buffer as a height map around a pixel P. This map contains hills and valleys, which represent the objects in the original scene. The sampling around the point P on the hemisphere Ω is then defined by sampling the depth buffer around the pixel P, in the area defined by the projected hemisphere. In screen space, this is a circle. The pixel P is then assumed to be occluded to a certain degree by hills above P, due to the fact that large hills cast shadows into a valley. Figure 8.3 demonstrates this for a single hemisphere.

The LineAO sampling scheme now copes with the fact that a single sampling hemisphere only captures nearby hills. To capture more distant hills while retaining high local detail, one would need to increase the radius of the sampling area, which requires an impractical amount of samples. Instead, LineAO uses multiple sampling radii with decreasing amounts of samples. LineAO densely samples the depth buffer around P to capture the local structural details and uses less samples on larger sampling areas to capture huge hills, caused by distant line bundles.

How this works in detail and how LineAO weights the different samples to achieve the desired effect, is described in the next section.



(a) Concept of AO (b) Interpretation in screen space

Figure 8.3: *Illustration of Equation (8.1) as shown in Figure 8.2 and its interpretation in screen space. In (b), the scene was rendered and the depth buffer of the original scene around P is shown in green. To measure the amount of occluded surface area of Ω, one can sample the depth buffer around P and check whether the sample $P + \omega_i$ is higher. This follows the interpretation of the depth buffer as a rocky landscape, where hills cast shadows into the valley of P and occlude a certain amount of ambient sky light.*

The Per-Pixel Data

This is a list of additional information needed by LineAO in screen space and where it can be gathered. Again, remember that each piece of information is available on a per-pixel basis; thus, we describe them as functions of a pixel P.

Projected Normal $n_l(P)$ LineAO requires a normal at each point on the line. This was introduced in [117], where a line is interpreted as infinitesimally thin cylinder for normal calculation. We calculate the normal of a line during the rendering pass using its tangent T. With the vector C (pointing from the line point towards the camera), the normal in eye space is defined as $\frac{T \times C}{|T \times C|} \times T$, which represents a vector pointing towards the camera that is perpendicular to the tangent. This normal is now projected using the projection matrix of the rendering pipeline. As we normalize the resulting vector, we omit the scaling by $\frac{1}{w}$ for de-homogenization as it is not needed. This yields a normal map $n_0(P)$ for the rendered lines. In addition, LineAO needs several l-times low-pass filtered versions of this normal map. Therefore, we create a Gaussian pyramid $n_l(P)$ and call l the *Gauss level*.

Depth $d_l(P)$: During rendering, we store the depth value of each pixel in a depth map $d_0(P)$. Similar to the normal map, the depth map needs to be

available in several low-pass filtered versions. Thus, $d_l(P)$ denotes the l-times low-pass filtered depth map for each pixel P. Again, l is called the *Gauss level*.

Zoom Level z: If we assume a sampling sphere with a radius of one in eye space, the sampling sphere will have the radius r' after projection, without perspective scaling. It represents the zooming factor applied by the projection. Mathematically spoken, this factor can be defined as the length of an unit vector after projection from eye space to screen space. If we define a placeholder matrix M_{CP}, which represents the specific rendering systems camera (view) and projection setup, we can calculate z as

$$z = |M_{CP}(1, 0, 0, 0)^T|. \tag{8.8}$$

Keep in mind that it is common in computer graphics to use an additional homogeneous coordinate w. In our case, it is 0 to avoid any perspective scaling that might be done by the projection matrix. The length of the projected vector can then be interpreted as the scaling ratio of the current camera and projection setup. Using this zoom level z in Equation (8.7) causes the hemisphere radii to adapt to the zoom level, thus ensuring a coherent LineAO effect when zooming.

With these fundamentals, we are now able to solve the LineAO equation in screen space.

Evaluating the Visibility Function

The visibility function V detects, whether a certain part on the hemisphere around a pixel P is occluded or not. Compared to other screen space approaches such as [90, 131], we do not do a back projection to clip space for visibility checks. The idea is to interpret the depth map as a height field. A point in a valley is darkened due to the shadow a hill in its direct vicinity casts. With this metaphor in mind, we evaluate the occlusion term $1-V_l(r\omega_i, P)$ from Equation (8.6) by checking whether the pixel $P + r\omega_i$ on the sampling hemisphere is higher than P and, therefore, occluding P. Visibility is then defined by the discontinuous step function

$$V_l(\omega, P) = \begin{cases} 1 & \text{if } d_l(P) - d_l(P + \omega) < 0 \\ 0 & \text{else,} \end{cases} \tag{8.9}$$

where a smaller depth value d_l indicates that the object is closer to the viewer.

By using the distance level as the Gauss level, dense line structures like bundles merge to solid geometry at higher distances and coarse or single lines disappear. In the next section, we introduce a very specialized weighting function which attenuates the visibility due to certain criteria, like distance, distance level l, and its surface properties to provide differentiated occlusion weighting for local detail and global structures.

Evaluating the Obscurance Weight

So far, we determined occlusion on a binary basis only. Either a pixel was occluded from one direction or not. The AO description from literature in Equation (8.3) weights the binary occlusion by the diffuse reflection surface property. This models the real-world phenomenon of ambient sky light very well for solid geometry. For dense lines and other thin structures, we need a more sophisticated weighting scheme, which handles local structures, global structures, and their interaction at once. This is not yet available in other SSAO approaches. Additionally, we include a weight to diminish the AO effect by the surface's lighting properties using the material's bidirectional reflectance distribution function (BRDF). This allows light sources to properly illuminate areas even if they are nearly invisible in terms of ambient occlusion, which cannot be compensated by applying lighting afterwards. This increases visibility of occluded structures if they are lit directly, which is very important in visualization.

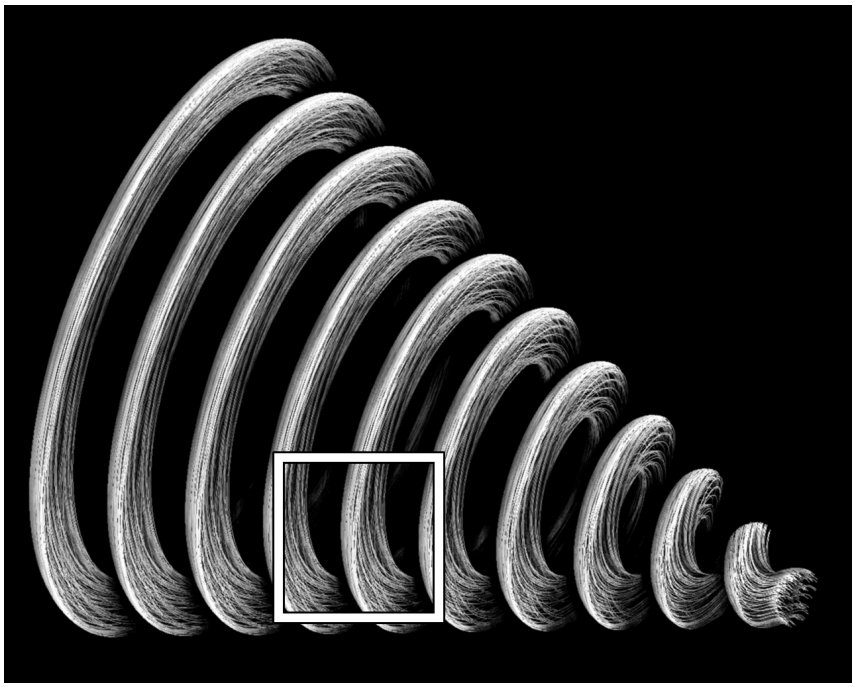
To achieve this, we split the weighting into two parts: a depth-based attenuation of visibility and an attenuation of occlusion by the surface's lighting properties:

$$g_l(\omega, P) = g_l^{depth}(\omega, P) \cdot g_l^{light}(\omega, P). \tag{8.10}$$

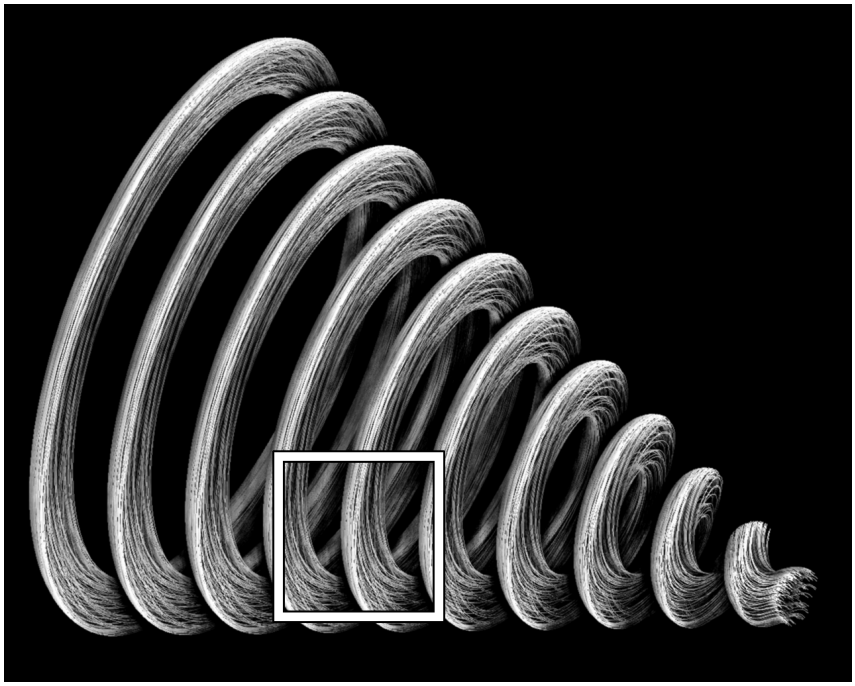
Depth-based Attenuation For the depth-based attenuation, we use the same depth difference that was used for the visibility function:

$$\Delta d_l(\omega, P) = d_l(P) - d_l(P + \omega) \in [-1, 1]\,. \tag{8.11}$$

For near occluders, $\Delta d_l(\omega, P)$ is the intensity of occlusion inside dense line bundles. For distant occluders, this describes the intensity of drop shadows and the inverse influence of empty space between several bundles. We need to define a falloff function $\delta(l)$ that attenuates the depth difference according to what kind of structure is currently sampled. Without a falloff, far occluders



(a) Constant $\delta(l) = 1$



(b) Our $\delta(l)$

Figure 8.4: *The influence of a proper falloff function for near and far occluders. In (a), a constant falloff is used. This creates high occlusion in line bundles but overestimates the occlusion of far away lines. In (b), our quadratic falloff is used. It creates high ambient occlusion in the spiral bundle, while handling the more distant bundles properly. In contrast to (a), this ensures a see-through effect to the back of the spiral, as marked by the white square.*

with a large depth difference would occlude each other too much, as shown in Figure 8.4. Remembering that the parameter $l \in [0, s_r - 1]$ in Equations (8.6) and (8.7) specifies whether local or global structures are sampled, the falloff is defined as

$$\delta(l) = \left(1 - \frac{l}{s_r}\right)^2 \in (0, 1]. \tag{8.12}$$

For near occluders, it is 1, generating a high occlusion for lines in proximity to others and converges towards 0 for far occluders. To additionally define the minimal depth difference needed to apply depth-based attenuation for occluder on P, we introduce the threshold $\delta_0 = 0.0001$. This now yields the depth-based attenuation as

$$g_l^{depth}(\omega, P) = \begin{cases} 0, & \text{if } \Delta d_l(\omega, P) > \delta(l) \\ 1, & \text{if } \Delta d_l(\omega, P) < \delta_0 \\ 1 - h(\frac{d_l(\omega,P) - \delta_0}{\delta(l) - \delta_0}), & \text{else.} \end{cases} \tag{8.13}$$

The value of g_l^{depth} is 1 for near occluders, whose depth difference to the current pixel is below δ_0, thus maximally emphasizing local structure in direct vicinity of the line. It is 0 and suppresses occlusion, if the depth difference exceeds the maximum defined by $\delta(l)$. For $l > 0$ (more distant occluders), this helps to avoid overly occluded distant line bundles as seen in Figure 8.4. In between δ_0 and $\delta(l)$, we use the Hermite polynomial

$$h(x) = 3x^2 - 2x^3, \forall x \in [0, 1] : h(x) \in [0, 1] \tag{8.14}$$

on the depth difference scaled by the falloff function. For near occluders, g_l^{depth} emphasizes lines in direct vicinity of the line at P, thus emphasizing local structure in dense line bundles. For occluders near in the image plane but with a high depth difference, the occlusion effect is weakened to avoid heavy influence on the shading of the local structures. In Figure 8.4(b), this can be observed in the line bundles at the back side of the spiral. They still show the proper local structure without being darkened too much by the front side bundles as in Figure 8.4(a).

Illumination-based Attenuation By separating the calculation of LineAO and local illumination [117], dense and heavily occluded areas will be very dark even if these areas are directly lit by a source in their direct vicinity. To avoid this

unwanted effect, we incorporate all the light sources of a scene into the LineAO calculation to attenuate the AO effect in these directly lit areas.

We define $BRDF(L_s, I_s, n, v)$ to be the illumination intensity of a light source s for a given light vector L_s, light intensity I_s, view vector v, and normal vector n. We calculate the reflected light L_l at the current point P in direction of ω using

$$L_l(\omega, P) = \sum_{s \in \text{Lights}} BRDF(L_s, I_s, n_l(P), \omega). \tag{8.15}$$

The lighting-based occlusion weight can then be defined as

$$g_l^{\text{light}}(\omega, P) = 1 - min(L_l(\omega, P), 1). \tag{8.16}$$

With this weight, we can attenuate the occlusion in brightly lit areas, with respect to the current sampling direction ω. As we did not normalize $L_l(\omega, P)$, we combine the influence of multiple lights on the local occlusion.

With the combination of depth- and light-based attenuation, the weighting function is able to emphasize local detail and their spatial relation as well as global structures without overemphasizing their shadowing effect (cf. Figure 8.4). The depth-based attenuation seamlessly scales between local and global structures and, therefore, allows LineAO to create an AO-like effect for dense line renderings on multiple scales. The inclusion of light intensities ensures high visibility of local detail in otherwise occluded areas if they are directly lit.

LineAO Parameter Summary

In Equation (8.7), we introduced three parameters. In this section we have a closer look on these parameters, specific meaning, and their recommended values to achieve optimal results. Please note, that we have used these values for all images in this paper.

- $s_r = 3$ – the number of radii to evaluate. This defines how many hemispheres are sampled around each pixel. The higher this value, the more detail influence the global AO effect. Evaluation at three radii ensures that detail in line bundles are rendered properly as well as smooth shadows of distant occluders. Higher values increase the total influence of smaller structures on the global AO effect. Lower values reduce visual

quality, since many mid-range structures cannot be detected. Due to Equation (8.12), s_r needs to be larger than one.

- $s_h = 32$ – the maximum number of samples on the hemisphere for each of the s_r iterations in LineAO. As we decrease the number of samples for increasing hemisphere radii as a harmonic series, the value of s_h defines the overall quality of the rendering with a trade-off regarding render speed. In Section 8.4.1, we show several values of s_h in comparison. We found that 32 is the best trade-off between quality and speed, since higher values effect quality only marginally. According to Equation (8.7), $s = 32$ and $s_r = 3$ yields in 59 samples being taken at each pixel.

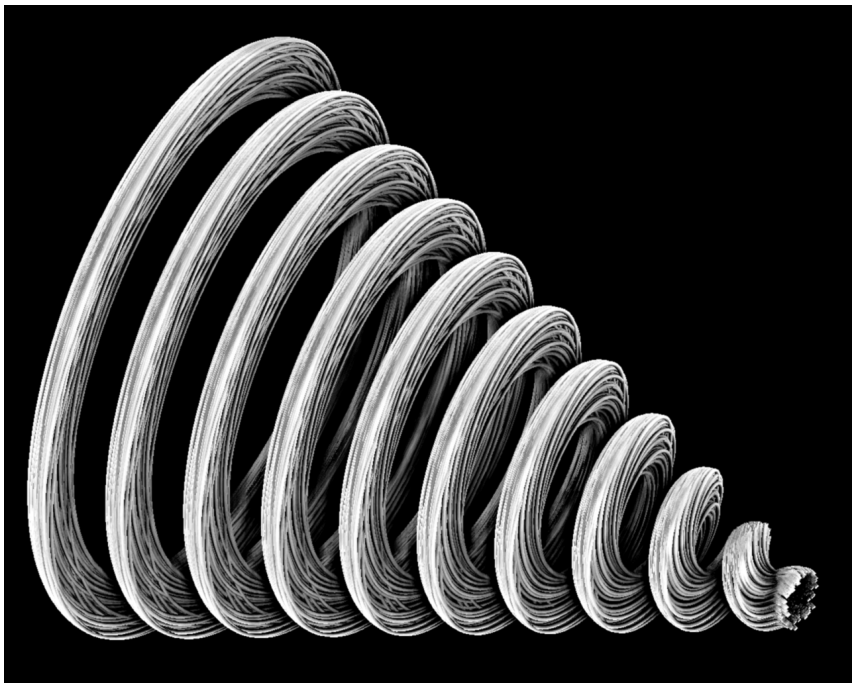
- $r_0 = 1.5$ times the line width – the minimum radius. This radius defines the smallest hemisphere for sampling near occluders. This value defines how much local detail is incorporated into the global AO effect. To handle all local detail properly, this value needs to be close to the line width, defined by the rendering system. We chose 1.5 times the line width here, which ensures all local detail are preserved.

8.3.3 Combining LineAO with other Methods

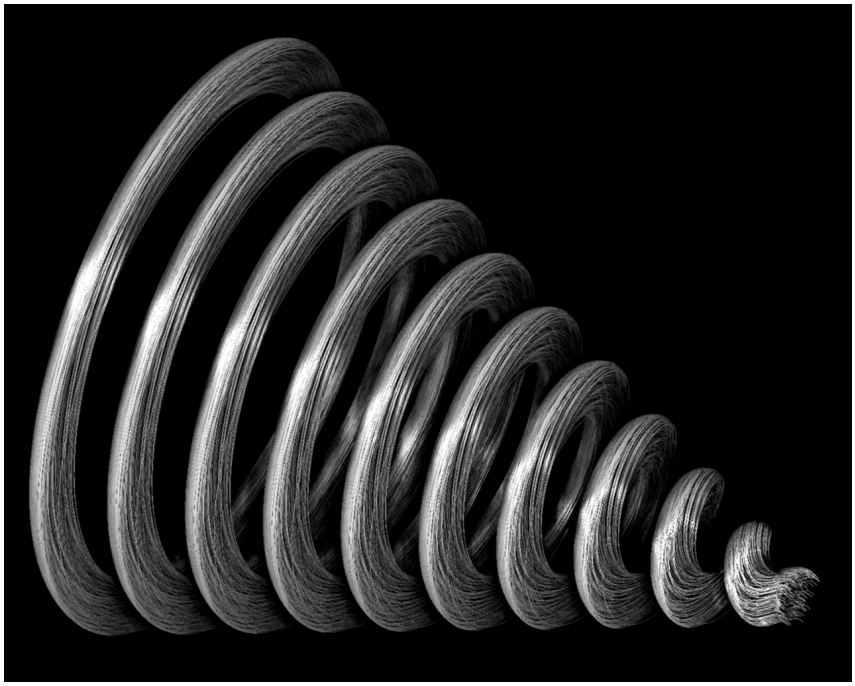
The LineAO approach can be easily combined with other approaches. When used with illuminated lines (introduced by Mallo et al. [117]), an additional shape cue is added: the specular highlight. This can be seen in Figure 8.5(b). As LineAO already represents the diffuse and ambient reflection, it is enough to additively combine the specular term of the illuminated lines approach with the LineAO intensity. When combining local illumination multiplicatively instead, an overly strong suppression of diffuse light will occur and local details might get invisible.

Besides illuminated lines, tube rendering is another interesting and commonly used alternative. In contrast to line renderings, tubes have a thickness in camera space and in screen space. Thus, we need to incorporate this in the parameter r_0 and the radius scaling term in Equation (8.7). We can now define the local neighbourhood using a radius larger than the tube width to ensure that the current tube is not occluding itself. We, therefore, define a new initial radius to replace r_0 in Equation (8.7) of Section 8.3.1 as

$$r_0^{\text{tubes}}(P) = 1.5 \cdot t_s. \tag{8.17}$$



(a) LineAO on Tubes



(b) LineAO using Illuminated Lines

Figure 8.5: *LineAO in combination with tube rendering (a) and illuminated lines (b). This yields additional cues for the shape of the line data and provides additional local structure detail. A downside of combining LineAO with local illumination can be the suppression of local details due to the partially small diffuse reflection on curved surfaces. A possible solution is to use the specular portion of the local illumination model only, especially since LineAO already includes diffuse reflection in its light-based obscurance weight (cf. Equation (8.16)). The result can be seen in Figure 8.7.*

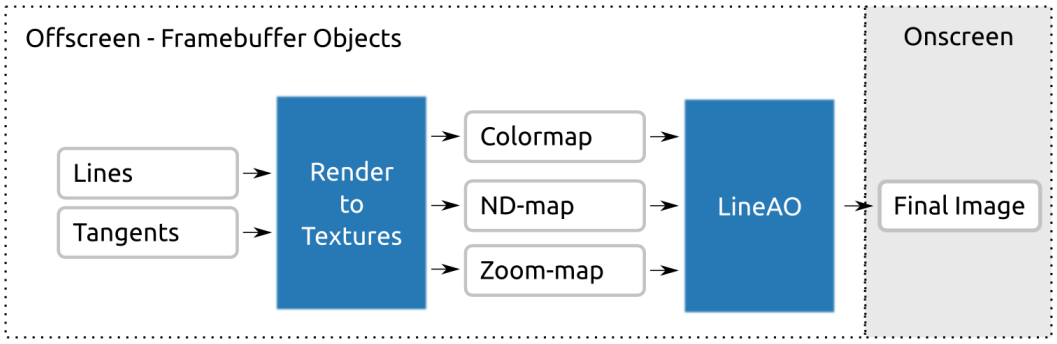


Figure 8.6: *The rendering pipeline. This figure shows the flow of information through the different rendering passes (blue boxes). The Line and Tangent data is provided by the application and uploaded to the GPU, where it is rendered to the ND-map, colormap and zoom-map. The ND-map additionally gets processed by the GPU to create the needed s_r Gauss levels using mipmapping. The final rendering pass then applies the LineAO algorithm on a per-pixel basis and renders it to the screen.*

Additionally, we keep the radius scaling scheme for tubes. This might sound a bit counter-intuitive, but yields smoother shadows for global bundles. This is caused by the fact that tubes are not overly thick and the rendered scene has the same density properties as a scene rendered using line primitives. Figure 8.5(a) shows the combination of LineAO with tube rendering [128].

LineAO is very flexible and allows combination with other lighting and shading schemes. Therefore, LineAO can be combined with any other line rendering methods easily.

8.3.4 Implementation

In the previous sections, we introduced our LineAO approach theoretically. In Sections 6.2 and 8.3.2, we gave an overview on how screen space approaches work in general and which specific information LineAO needs beforehand.

In this section, we provide an implementation guideline using OpenGL and GLSL. The implementation in OpenGL is straight forward. We need frame buffer objects to render the scene to several output textures instead of the screen. These output textures contain the scene itself and the required LineAO inputs, like a normal at each pixel for the represented line. The LineAO pass then renders a screen-filling quad and uses the LineAO fragment shader to evaluate Equation (8.7) for each pixel visible on screen. Figure 8.6 shows our specific LineAO rendering pipeline, the two needed render passes, and the flow of data between the them.

Pass 1: Render To Texture

We start by transferring the line data, including the tangent vector at each vertex, to the GPU using the standard OpenGL pipeline. In fact, this means rendering the scene. Although the tangent vector can also be described by the start and end vertex of each line segment, we have to transfer it separately as the vertex shader of the first pass has no access to the other vertices of a primitive. An alternative to uploading the tangents is to use a geometry shader, which has access to the vertices of each line segment.

The Colormap The first pass now renders and colors the lines as usual. As we use frame buffer objects (FBO), the first pass is able to render these lines to a texture instead of the visible framebuffer. This creates a texture containing the plain rendered lines, including their coloring. For further reference, we call it *colormap*.

ND-map As LineAO requires normals for all visible lines, we utilize the fragment shader of the first pass to calculate these normals. The fragment shader can utilize the tangent that has been interpolated for the current fragment automatically by the GPU. The tangent T and the camera vector C then define a normal for the current fragment P as $n_0(P) = \frac{T \times C}{|T \times C|} \times T$. This calculation has been explained in Section 8.3.2. The normal is stored in the RGB-triple of the second output texture called *ND-map*.

Besides the normal, LineAO needs a linear depth value for each pixel it processes. The GPU does this automatically, but scales it by the homogenization factor $\frac{1}{w}$. As GLSL provides the projected coordinates of each fragment P in `gl_FragCoord`, it is trivial to get a linear depth $d_0(P) = P.z \cdot P.w$ and storing it in the alpha channel of the ND-map.

Zoom-map Finally, the zoom level z can be calculated. We already introduced this calculation in Equation (8.8), but used some kind of a place-holder matrix M_{CP} to represent the view and projection setup of the used rendering setup. In OpenWalnut, we zoom the scene by scaling the OpenGL *modelview matrix* $M_{ModelView}$. The orthographic projection matrix M_P is left untouched and contains only the scaling to the clip coordinate system and no perspec-

tive scaling. As we use an unit vector whose w coordinate is 0, perspective projection matrices work equally well. The zoom level is then defined as

$$z = |M_P M_{ModelView}(1, 0, 0, 0)^T|. \tag{8.18}$$

The zoom level is written to a floating point texture, called *zoom-map*.

Pass 2: LineAO

After the first rendering pass, only the unfiltered ND-map is available. From Equation (8.7), one can see that we need $s_r - 1$ filtered versions of this map. To build this Gaussian pyramid automatically, we enable mipmapping for this texture. In OpenGL, this can be achieved by setting the ND-map's texture min-filter to enable mimap generation via `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR )`.

With this, the second render pass has given everything needed for LineAO. Typically, the second render pass is done using a screen-filling quad with all the above output textures bound to it. This way, OpenGL calls the LineAO fragment shader for each pixel of the originally rendered scene of the first pass. As we also disabled the FBO now, the results get rendered directly onto the screen.

The LineAO fragment program is now applied to each pixel and evaluates Equation (8.7). The implementation is straight forward and can be done by a nested for-loop in GLSL. As the LineAO parameters s_r, s_h, and r_0 are compile-time constants, the GLSL compiler unrolls the loops and creates optimized GPU code.

Sampling To avoid artifacts while sampling the hemisphere, we use a Monte Carlo sampling method. To achieve this, we create a low resolution, random vector map $R(P)$ on the CPU and tile it on the quad to avoid up-sampling in the LineAO pass. We query two random vectors $R_1 = R(P)$ and $R_2 = R(\frac{i}{s}, \frac{j}{s_r - 1})$ for the current pixel P. The vector R_2 is queried at $(\frac{i}{s}, \frac{j}{s_r - 1})$, which is a point in the texture space, depending on current level and sample, according to Equations (8.6) and (8.7). The sampling vector ω_i is then defined to be the random vector R_1 reflected along R_2. This avoids that the sampling vector ω_i is the same on different levels l of the LineAO equation and thus, avoids sampling in the same direction multiple times.

Boundary Cases An important point to mention here is the proper handling of boundary cases, e.g., if $P + r\omega$ (Equation (8.4)) is outside the texture space. A simple approximation to handle this is by reflecting the vector $\vec{\omega}$ on the normal $n_l(P)$. Unfortunately, this causes over-estimation if the area around the border is salient compared to the invisible area and under-estimation in the opposite case. An alternative solution is to render the scene slightly larger than the viewport. But to provide the invisible but needed information for global occluders, the invisible area needs to be impractically large. We decided to ignore samples outside the texture space. This creates a coherent LineAO effect at the borders without the risk of overestimation.

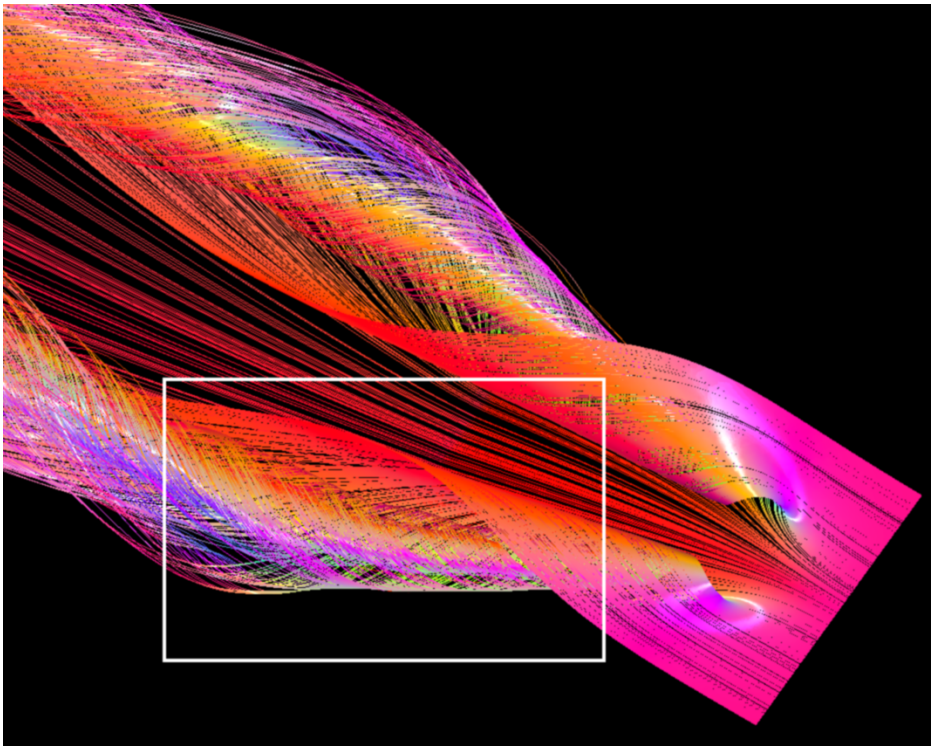
With this guideline, one is able to implement LineAO completely on the GPU. LineAO perfectly fits into the standard graphics pipeline and runs on consumer-level hardware. The source codes are available in OpenWalnut (see Chapter 3 for details or online at `www.openwalnut.org`).

8.4 Results

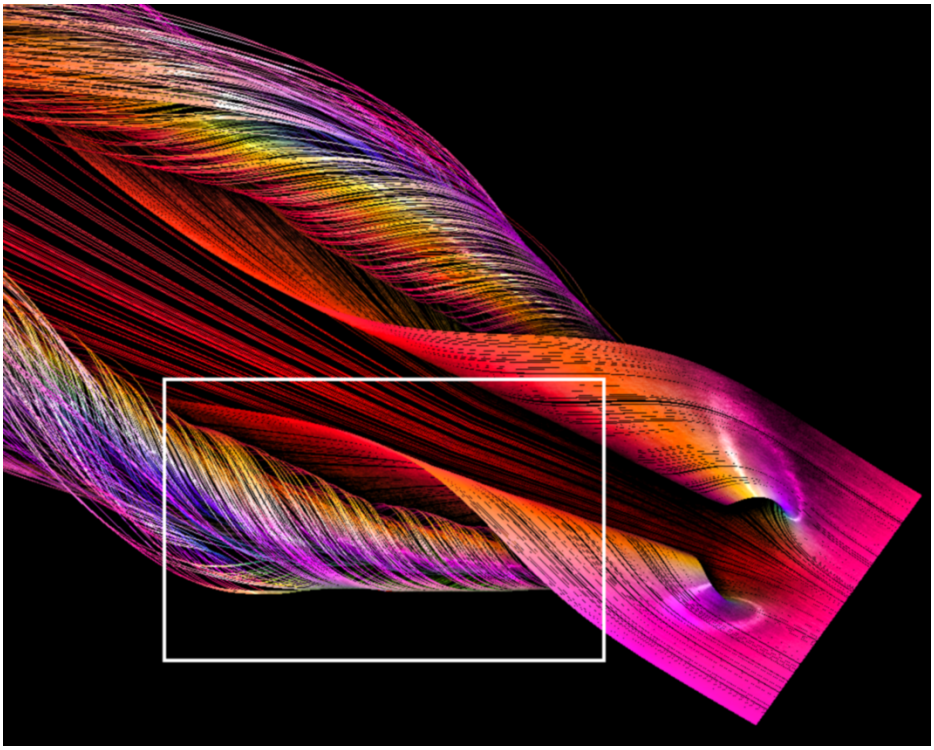
In the previous section, we have introduced our approach for improved shading and illumination of dense line data. Our LineAO approach modifies and extends standard AO and SSAO, making it comply to the requirements and properties of line data. It phenomenologically creates the ambient occlusion effect, generating a more realistic image in real-time with greatly increased spatial perceptibility.

Figures 8.7 and 8.8 show the streamlines around the two main vortices of a delta wing dataset obtained from a fluid dynamics simulation. Especially Figure 8.7(a) does not provide any cues that allow derivation of streamline structure and depth. Our method adds these missing cues and depicts the folding around the main vortices much better.

Figure 8.9 shows a fiber tractography dataset of realistic size (74 313 lines containing a total of 11 000 000 vertices). This particular image was published in LeMonde “20012: la science en images” [P10] in 2012. Figure 8.10 compares this LineAO rendered image with illuminated lines rendering, standard SSAO from Crytec [131], and ray-tracing using the ray-tracer package POV-Ray [208]. Although the plain illuminated line rendering does not provide any spatial cues and only little shape hint due to local specular highlights, it is still the standard way of exploring large medical data like this. Only interaction can reveal further structural information. Figure 8.10(b) shows



(a) Phong illuminated lines



(b) LineAO with Phong illumination

Figure 8.7: *Streamlines around the main vortices of a delta wing dataset obtained from a fluid dynamics simulation. (a) The specular highlights provide local shape information but are not able to properly represent the folding around the vortices. (b) LineAO enormously improves the perception of these folding structures.*

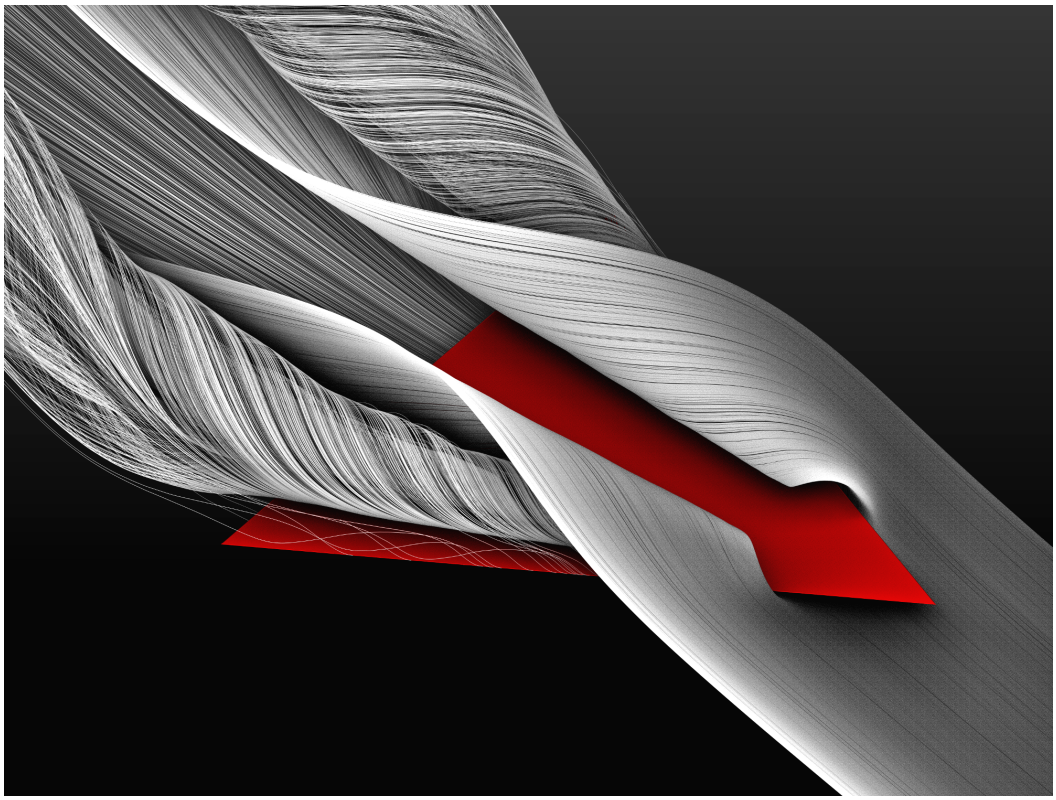


Figure 8.8: *The streamlines around the main vortices of a delta wing as in Figure 8.7. LineAO works properly with solid geometry and can be combined easily with other shading methods for surrounding and embedded objects. This image was published in the gallery of fluid motion [P11] in 2012.*

a standard SSAO algorithm applied to the same line data. Although it reveals some global structure with a halo effect, the technique generally is not suited for line data. This can be seen in the under-estimated thin structures on top of the image as well as on the over-estimated fissures of the *Frontal Lobe*. Increasing the number of samples in SSAO does not solve this problem, as it is still not able to distinguish local and distant occluders properly. For comparison, we also rendered a ray-traced image, with POV-Ray's radiosity approach enabled. As it is not possible to ray-trace lines per se, we have used thin, arithmetically described cylinders with spheres as joints to describe the line strips. Besides the enormous calculation time of 14 hours, the POV-Ray renderer suffers the problem of intense self-shadowing in dense areas, causing overlapping and dense bundles to be undistinguishable. With LineAO, it is possible to distinguish individual lines and bundles as well as their layering even without interaction. It is possible to see the structure of the fissures in the *Frontal Lobe* and the layering of bundles in the brain stem.

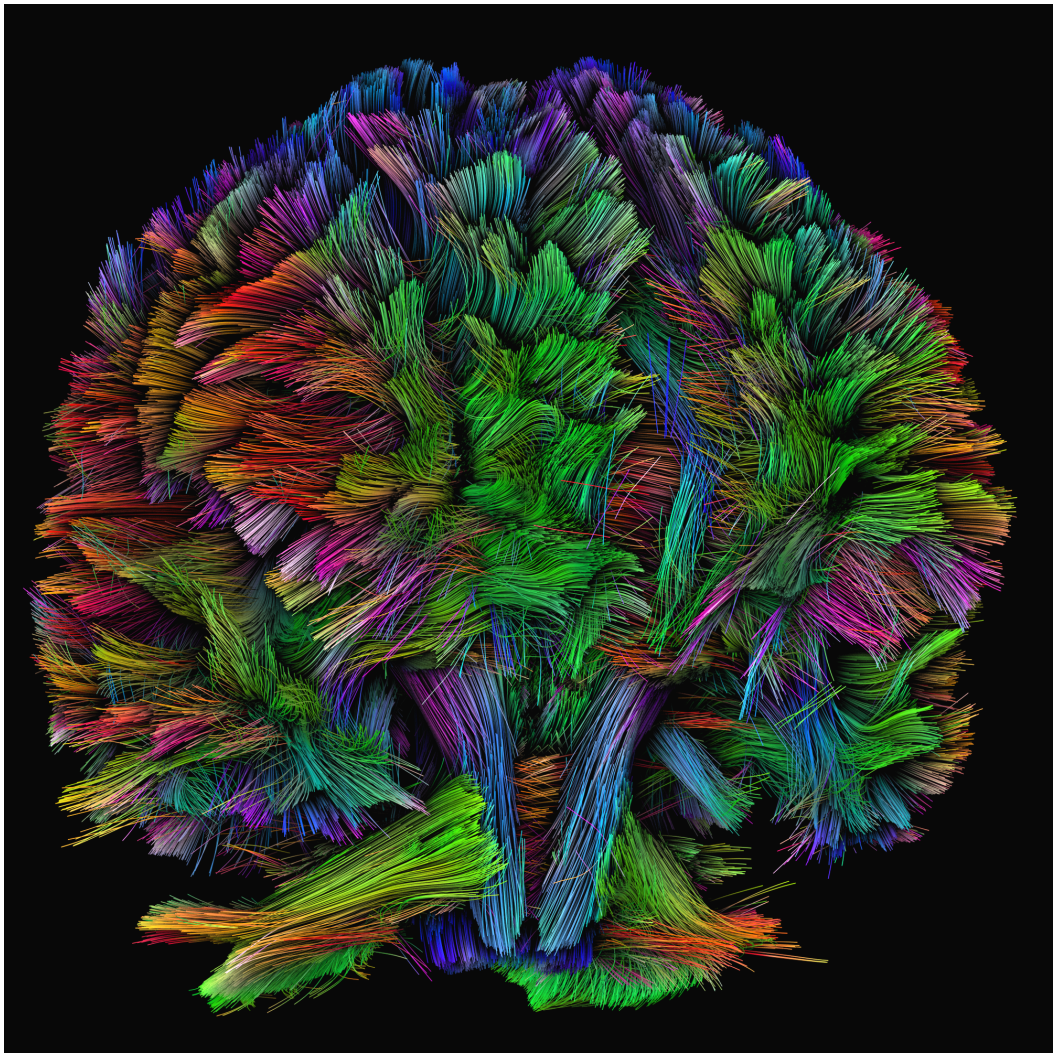
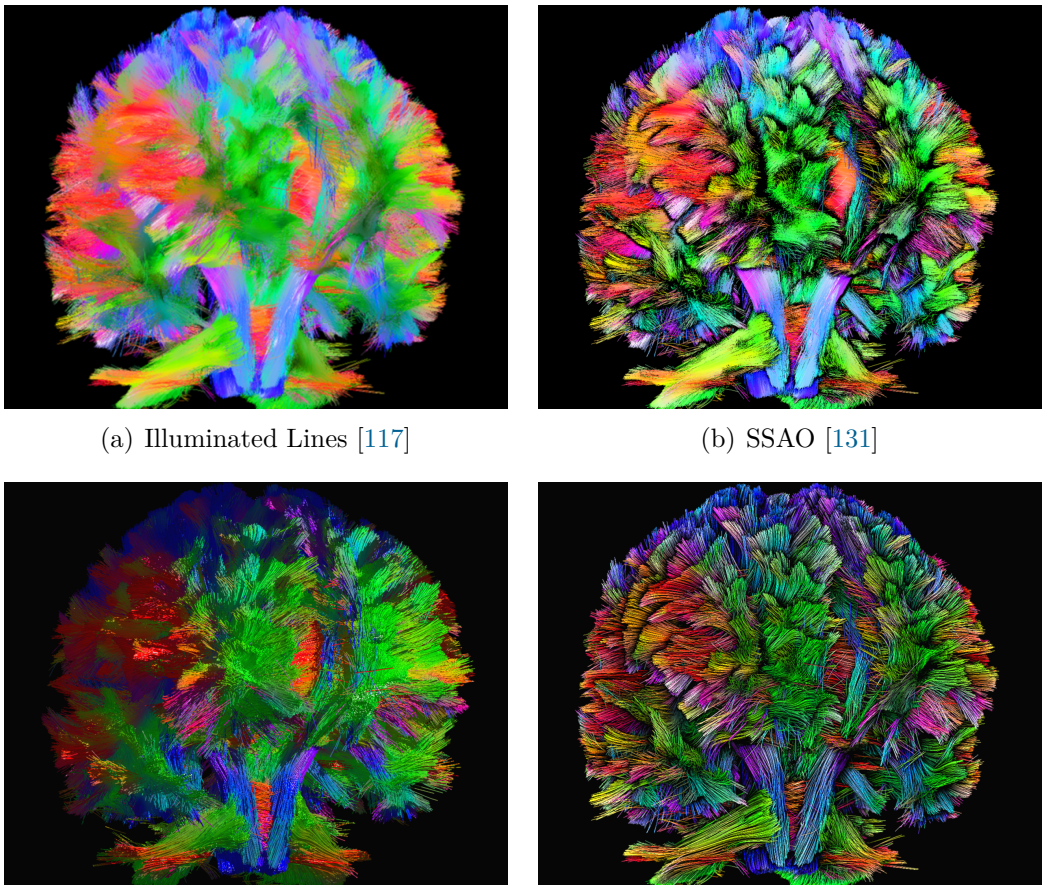


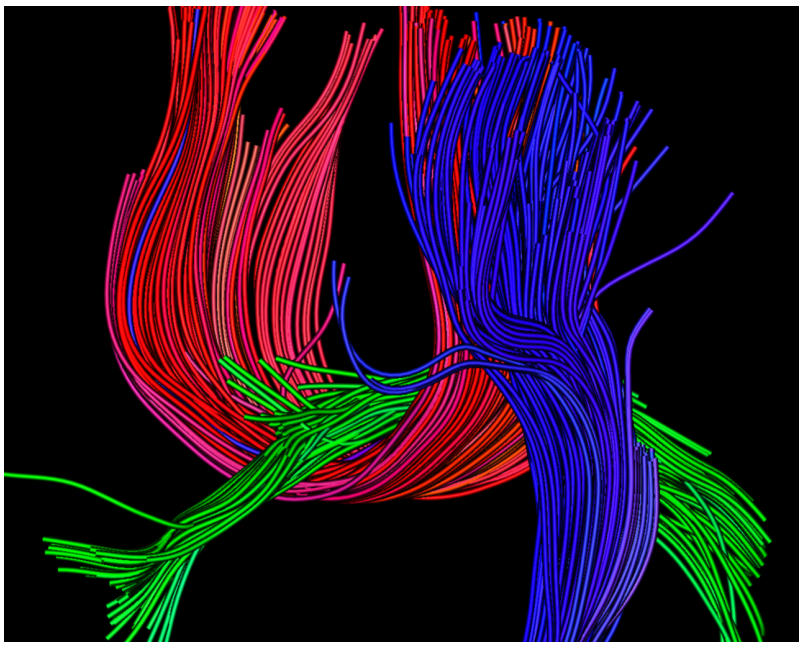
Figure 8.9: *Deterministic fiber tracking of the human brain, similar to Figure 8.10. This image was published 2012 by LeMonde in the gallery "20012: la science en images" [P10]. A similar image is on the front cover of the UC San Diego's Discoveries Magazine 2014 [P12].*



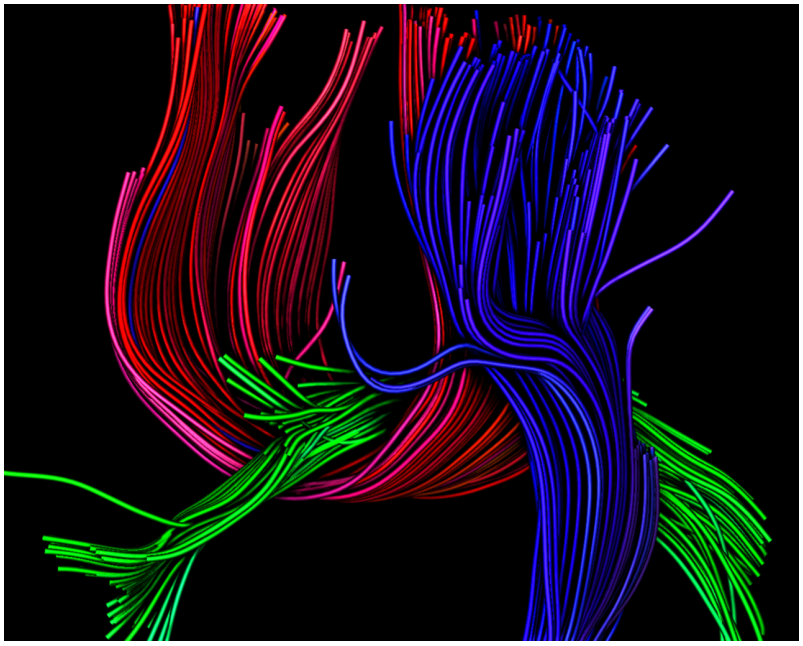
(a) Illuminated Lines [117] (b) SSAO [131]

(c) Ray-traced using POV-Ray (d) LineAO with Phong illumination

Figure 8.10: *Comparison of different line rendering approaches with a deterministic fiber tracking of the human brain. The fibers are colored using the tangent-based directional colormap [145], which is very common in the neurosciences. A prominent example for comparing each technique is the layering of bundles at the brain stem, where the* Pons *and* Medulla Oblongata *pass into the* Spinal Cord*. In comparison to (a),(b), and (c), LineAO (d) is able to show the local structure of fibers in a bundle, as well as their spatial relation in a global scope, without intense self-shadowing in dense and overlapping areas. Besides ray-tracing, all methods perform in real-time. (c) took 14h to compute.*



(a) Illuminated tubes



(b) LineAO with Phong illuminated tubes

Figure 8.11: *This picture shows the combination of local illumination, tubes, and LineAO in a part of the* Corpus Callosum *(red),* Cingulum *(green), and* Corticospinal tract *(blue). The illuminated tubes already provide some structural information for each line but make it hard to distinguish individual lines. Their spatial relation is not completely visible. For example, the shape of the* Corticospinal Tract *(blue) is not fully determined with illuminated tubes and seems to be a round bundle. With LineAO, it is immediately clear that this tract has a flat shape. Besides this, its distance to the* Corpus Callosum *(red) cannot be estimated with illuminated tubes, whereas LineAO reveals their closeness.*

(a) $s = 8$ — 63FPS (b) $s = 32$ — 37FPS (c) $s = 64$ — 24FPS

Figure 8.12: *Top view of the delta wing vortices, zoomed into the part framed in Figure 8.7 to show quality difference between the different sample counts. (a) Contains a lot of noise artifacts compared to (b), whereas the visual difference between (b) and (c) is insignificant.*

In Figure 8.11 a selected part of a brain fiber tractography dataset is shown. The LineAO approach in combination with tube rendering shows spatial relations of these bundles in a very natural way. Even distinguishing a single fiber in a bundle is possible. Without LineAO, estimation of vicinity between these three selected bundles is difficult and bundle shape can only be recognized with interaction. For example, the closeness of the *Corpus Callosum* (red) and the *Corticospinal Tract* (blue) can be seen very clearly with LineAO, whereas Figure 8.11(a) provides no visual cue regarding this. Only anatomic knowledge and interaction with the scene helps to grasp the relevant spatial information.

8.4.1 Performance and Accuracy

In Section 8.3.2, we listed the parameters and our default settings. We have used these values for rendering all images throughout the paper. Thereby, the sampling count s influences the overall rendering performance and a trade-off between accuracy and performance has to be done. Figure 8.12 compares a drastically zoomed part of the delta wing vortices dataset, rendered with different sample counts. It clearly shows that sample counts lower than 32 introduce significant noise artifacts, whereas values above 32 only slightly improve rendering quality while diminishing rendering performance drastically.

Figure	Line strips	Vertices	FPS Phong	FPS SSAO	FPS LineAO
8.5(b)	1 000	1 000 000	460	59	37
8.5(a)	1 000	1 000 000	140	42	17
8.7	1 000	3 600 000	260	55	20
8.10(d)	74 313	11 000 000	30	22	16
8.11	564	102 700	500	80	30

Table 8.1: *Performance of LineAO for several datasets in comparison to plain Phong illuminated lines and SSAO [131].*

To measure LineAO performance, we used a machine with two Quad-Core AMD Opteron 2352 processors, 32GB RAM, and a GeForce GTX480 graphics card. However, the CPU and RAM do not play an important role here since the computation is done on the GPU only. Table 8.1 compares the frames per second (FPS) of LineAO with plain Phong illumination at a screen resolution of 1280×1024 pixel and clearly shows that LineAO works in real-time on todays lower to medium-level consumer graphics hardware (2014).

With this, it is evident that Phong illumination is data-bound, whereas LineAO and SSAO mainly depend on the number of pixels covered by the rendered lines. Phong is evaluated for each fragment instead of each pixel, as LineAO does. This explains why larger dataset sizes do not have a drastic impact on FPS for the additional LineAO pass, compared to Phong illuminated lines.

8.5 Discussion

8.5.1 Limitations and Problems

This section is about four major issues associated with LineAO. Although the first three issues are not imposed by LineAO, they affect it in a general way. Hence, we explain these issues and show possible solutions to avoid or alleviate them.

Density

The main limitation of LineAO is that it does not provide improved shading for very coarse data. In our case, dense line structures provide a high occlusion

inside the bundles and smooth shadows for salient structures. Very coarse lines and thin structures *naturally* do not cause much occlusion. However, this is what one would expect naturally. But it might be desired in some case to have improved shading, even if the line data density is not high enough. In those cases, line structures have to be thickened. A good option to achieve this is to use a fast tube rendering techniques, like the one introduced by Merhof et al. [128]. We have used their approach to overcome the density problem in Figure 8.11(b).

Defining a Minimal Density

The above limitation rises the question whether a minimal density can be defined. Lines have a certain width on screen in computer graphics. This width is fixed and independent from any scaling of the scene or the line primitive. This was the reason for introducing the zoom level z in the term $r_0 z \cdot (j^2 + j)$ of Equation (8.7), Section 8.3.1. It copes with the problem of decreasing line density on screen when zooming in, but also introduces the dependency on the current camera/view/projection setup. This would allow to define a minimal screen space line density in dependence on the current camera/view/projection.

Unfortunately, this is of no value for defining a density in eye space for two reasons: (1) the projection from eye space to screen space is not uniquely invertible and (2) the intrinsic fixed line width on screen is not related to any size-measure in eye space. Figuratively speaking, dense line bundles in eye space might be projected to a single pixel or spread on the whole screen as thin lines with huge gaps in between. In other words, defining a minimal density of lines per volume is not feasible.

Self-Occlusion

Another, but general problem is the self-occlusion problem. Although this is no issue specifically related to LineAO, it affects it. LineAO might even intensify the problem to a certain degree, as it shades areas that would be visible otherwise.

Specifically for line data, there are several possible solutions. Akers et al. [2] introduced a method to interactively filter large line datasets using regions of interest. This way, occluding parts of the data can be removed manually. An alternative approach is to automatically filter line data using known structures as regions of interest. In Chapter 4, we have demonstrated this using known

anatomical regions in the human brain. Other methods automatically remove or blend out lines that do not contribute to the understanding of the data [66, 119].

When combining LineAO with one of these methods, it is possible to reduce visual clutter and self-occlusion, while retaining spatial and structural perceptibility of the line data. This was also stated by Günther et al. [66].

Global GPU Memory Access

This is a rather technical issue of LineAO in the context of the current GPU memory architecture. When reviewing the sampling radius term $r_0 z \cdot (j^2 + j)$ of Equation (8.7), Section 8.3.1 again, it gets obvious that it can grow drastically, depending on the amount of hemispheres (increasing j) and zoom level. This is intended and is for capturing global structures in the scene.

From a purely theoretical point of view, there is nothing to argue against it. When implementing LineAO on modern GPU architecture, huge sampling radii, which might cover large parts of the texture space, are indeed an issue to keep in mind. The GPU consists of shader units, which then run the different shaders. In our case, the LineAO fragment shader, which processes a specific pixel of the input textures. Each unit has a small, local memory. The GPU populates this with data of the currently bound textures for the specific area the shader is processing. This way, each shader can access local data in the textures very fast. On a higher level, these shaders are organized in groups. These groups have their own group-shared memory, usually working as global memory cache. In other words, the memory model of the GPU is designed for very local operation.

In the context of LineAO, this means that the global sampling is somewhat contradictory to the local data model of the GPU. The Monte-Carlo sampling of large texture areas creates a lot cache misses in the shader-local and group memory. This causes long wait cycles for the whole group to re-populate the memory, as the global, shared memory access time is tremendously higher. This is especially true, when considering the fact that a lot of groups will have cache misses at the same time, causing a lot of synchronization waits. In turn, this leads to the assumption that a major share of the LineAO calculation time is caused by global memory access. This assumption is backed up in two ways:

1. We used a very small r_0 to ensure that the sampling radius is small and always stays inside the locally cached part of the input textures. While increasing r_0, we found that there are certain values for r_0 where the

frame rate suddenly changes. To be precise, we found three values for r_0 as we use three hemispheres. Every time a hemisphere gets larger than the cached texture area, the frame rate suddenly drops, while it was nearly constant for values in between. The exact values are of no interest here, since they depend on

- the zoom level z;
- the GPU architecture and the caching strategies;
- and the GPU's local memory sizes.

2. We used a modern graphics card: the NVIDIA GeForce GTX Titan. In contrast to the originally used GTX 480, this card has 2688 processing cores instead of 448. This is six times the computational power apart from increased clock speeds, memory speeds, and memory bandwidths. Interestingly, the frame rate of LineAO only doubled for the above examples. The doubling of the frame rate probably relates to the doubled memory clock and the increased memory bandwidth of the GeForce Titan, which compensates the increased number of cores waiting for data.

In general, proving or measuring this is hard. The hardware details, internal structures, the core-assignment, and the caching strategies are not known to the public. Although OpenGL profiling systems exists, they are not able to provide useful information on excessive global memory access and related wait times on a sufficient level of detail.

When considering Table 8.1, it gets clear that this is no critical performance issue. But it should be considered an issue, when using a very high resolution for rendering. This way, even the hemisphere used for local occluders tends to get larger than the local shader memory.

8.5.2 Future Work

Other Methods: A major direction for further research is to combine LineAO with other line rendering techniques. An example is the halo-based techniques of Everts et al. [52], which is able to provide depth cues for coarse line structures. To tackle the self-occlusion problem, another option is to combine LineAO with opacity optimization algorithms [66].

Other Types: Another very interesting direction for further research is about the application of LineAO with other types of data, i.e. points or complex

triangle meshes. In 2013, we already explored the possibilities of LineAO for point-based data. The next chapter will introduce this extension in detail. Additionally, using LineAO with triangle meshes yielded promising results, but required some further tweaking of the weighting functions. For example, we used LineAO in Chapter 5 for Figure 5.3. The advantage of LineAO over several standard SSAO approaches is the same as for lines: it emphasizes local details and spatial relations at once.

Sampling Scheme: Another major goal is to adaptively sample the occlusion integral, depending on density information. This could replace the Monte-Carlo sampling scheme and might provide more accurate shading, especially for global occluders. A possible way to achieve this is to estimate the density of lines in screen space, by using an additional precomputation step. This step could calculate a map of dense and less dense areas for a given view – interestingly, this is what LineAO basically does. When combining this with a map of already sampled pixels, LineAO would be able to achieve a very high sampling resolution over multiple frames by re-using the previous samples and focusing on high-density areas. This would furthermore increase smoothness and accuracy of the shading.

8.6 Conclusion

With current line rendering methods, it is not sufficiently possible to distinguish associated bundles of lines and their spatial relation to others locally and globally. Thus, we proposed a novel approach, tailored towards line rendering. It allows to grasp the ***structure of bundles and the spatial relation*** between structures at ***local and global scope***. The combination with directed local illumination adds further perceptual cues for the local characteristics of single lines and line bundles. Its ***simplicity and performance*** distinguishes it from other approaches, which use complex, surface-based techniques or expensive precalculations to provide insight into structures. Our presented method does ***not require any precomputation*** and does not introduce additional geometry for each line segment. The ***real-time*** ability, its ***consistency under modification and interaction***, and the possibility of combining LineAO with other interactive methods allows it to be used as ***add-on to existing approaches*** in interactive and explorative visualization environments.

Our approach is a considerable step towards visual quality, spatial perception, and usefulness of line-based visualization techniques and perfectly fits into the constraints of modern, interactive data exploration and visualization environments.

9

PointAO – Improved Ambient Occlusion for Point-based Visualization

This chapter is based on the following publication:

[P13] – S. EICHELBAUM, G. SCHEUERMANN, and M. HLAWITSCHKA. **PointAO – Improved Ambient Occlusion for Point-based Visualization**. *EuroVis - Short Papers*. Ed. by M. Hlawitschka and T. Weinkauf. 2013, 13–17

Online: http://sebastian-eichelbaum.de/pub13b

9.1 Overview

The Data: Points In many fields of science, the visualization of large amounts of particles, glyphs and point-based data plays an important role. Typically, these points and particles have a direct physical meaning and there is a huge variety of applications, where point data is used.

Generally, the term "point-based data" can be understood in two ways.

1. as an unordered cloud of points, where the *positional information* is of relevance. Typical examples include molecular simulations, protein structures, and spatial laser scans. In these examples, the spatial arrangement of single atoms, molecules, and laser samples is crucial to understanding the data.

2. as *data associated with a point*, where the measured or simulated information is associated with points. These points are usually, but not necessarily arranged in a lattice. Typical examples include magnet resonance imaging (MRI), diffusion tensor imaging (DTI), high angular-resolution diffusion (HARDI), physical and mechanical simulations, and much more. In general, one could say that this interpretation includes nearly all measurement and sampling techniques.

Of cause, this list is not exhausting. It gives an overview on how important point-based data is in many fields of science and application. The proper perception of the global spatial relations and local structure are crucial to understand this particular kind of data and the underlying models.

Visualization of Point-based Data In the context of PointAO, we focus on discrete visualizations of point-based data. Instead of representing a continuum, the points of the dataset can be visualized directly with glyphs. Glyphs allow for direct visualization of positional information and associated information and do not need any reconstruction step to re-create a continuous domain, like [4] or [97]. They can capture both cases mentioned in the previous chapter. Basic representations are spheres, but more-complex objects can describe more properties of the associated data they represent. With the vanishing memory restrictions and the increasing data throughput, unstructured point data becomes more and more popular and the possibility for direct visualization of point data often leads to a good first impression of it.

Especially in the area of medical visualisation, a lot of different glyphs are available, depending on the imaging modality used. Some commonly known

examples for second-order tensors are super-quadric glyphs by Kindlmann [92] or tensor splats by Benger and Hege [13]. For visualization of higher order tensors and HARDI data, other types of glyphs [149, 178] were introduced. This is only a small list of examples. As PointAO does not rely on a specific type of glyph or a specific kind of imaging modality, simulation model or use case, we refer the reader to a survey on glyph-based visualization by Ropinski et al. [161] for further details.

Besides glyph-based rendering, other researchers use splats and screen-space processing to create the visual effect of solid surfaces in the point data, namely Rosenthal and Linsen [162] and Rusinkiewicz and Levoy [168]. Dobrev et al. [45] have added shadows to their point cloud rendering to improve the realism of the resulting images.

In the area of molecular visualization, proper shading for improved spatial perception is known to be important. Several methods for visualizing molecular structures were published recently that deal with ambient occlusion [65, 201] and illustrative rendering [239].

The Problem It is known that global lighting effects are very important for determining an object's position and spatial relations [103, 155, 214, 215]. Hence, global illumination became popular in recent years, especially in molecular data visualization. Unfortunately, these methods often suffer several limitations like immense precalculation costs, reduced accuracy due to needed simplifications, or limitations towards the simultaneous shading of local *and* global detail.

For point-based visualization, ambient occlusion is not yet sufficiently researched. Although the application of standard SSAO approaches like Crytek SSAO [131] improves spatial perception in dense glyph renderings to a certain degree, they fail to emphasize the shape of the glyphs/splats/points. But especially with higher-order glyphs, the shape of the glyph is of crucial interest.

Our Solution: PointAO In this chapter, we improve the LineAO method of the last chapter. It was tailored to the specific problems in line rendering and its global illumination and shading. We contribute an enhanced screen space ambient occlusion approach for point and glyph visualization, which overcomes the before-mentioned problems and provides a greatly ***improved structural and spatial perception*** with ***simultaneous depiction of local shape and global structures*** in ***real-time, without precomputation***. PointAO can be applied to any kind of glyph-based and point-based visualization, independent of the specific source, modality, and physical mean-

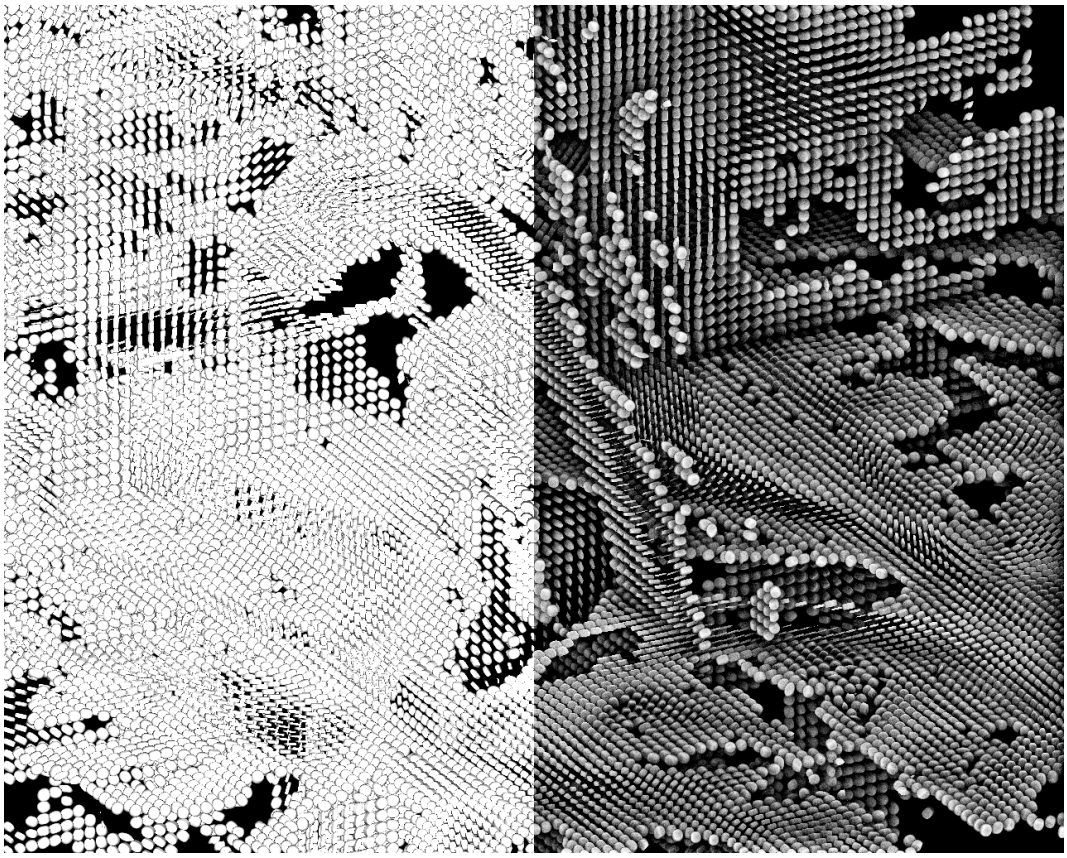


Figure 9.1: *DTI super-quadric glyphs [92] rendered on three orthogonal slices through the brain. The Phong shaded glyphs on the left do not provide any spatial relation between them. With PointAO on the right, the spatial relations of the glyphs on the three slices gets obvious. This was not possible before and the fact that PointAO works in real-time, without any precomputations, makes it a perfect addition to nearly every point- and glyph-based visualization approach.*

ing of the data. In the remainder of this chapter, we refer to all the different kinds of three-dimensional point data simply as *point data*.

In the next sections, we shortly recap the important ideas of LineAO and introduce the necessary changes to improve the visualization of point data. The chapter closes with several resulting images, showing PointAO in the context of several application cases and types of point data.

9.2 Background

The very foundation of each ambient occlusion technique is the discrete description of ambient light that does *not* reach the point P on a surface with normal n due to occlusion. When all objects are opaque, we only need to know

the amount of light being occluded on a hemisphere around P in direction of n. This yields the standard, discrete AO equation (cf. Equation (8.3)),

$$AO_s(P,n) = \frac{1}{s}\sum_{i=1}^{s}(1 - V(\omega_i, P))\langle\omega_i, n\rangle, \tag{9.1}$$

which approximates the ambient occlusion factor at a point of a surface by sampling the surroundings of P hemispherically. ω_i is one of the well chosen s hemispherical direction samples taken into account. The computationally most expensive part is the function $V(\omega_i, P)$, which describes whether the ambient light is reaching P from direction ω_i. Most SSAO approaches, including LineAO, render the scene, the normal information and depth information for each pixel into a texture. Evaluation of V is then done by sampling around a pixel P and checking whether the sampled pixel is to the front or to the back of P, by comparing the previously saved depth buffer of the scene. If it is in front, it occludes light. Depending on the number of samples s and the distribution of the sampling directions ω_i, this can lead to severe artifacts, missed occluders, and makes the result dependent on the scaling of the scene and the sampling distance.

The key element of LineAO is that it computes an average of the AO factors for multiple hemisphere radii. It increases the radius in each distance-level j, while reducing the number of samples on outer shells. Additionally, it modifies the sampling radius depending on the current zoom of the scene to stay consistent and weights the visibility function V by a factor g, which depends on the depth-distance, distance-level, and light properties of the occluder. This led to the LineAO Equations (8.6) and (8.7):

$$AO_{s,l}(P,r) = \frac{1}{s}\sum_{i=1}^{s}[(1 - V_l(r\omega_i, P))g_l(r\omega_i, P)] \tag{9.2}$$

$$\text{LineAO}_{s_r,s_h,r_0}(P) = \sum_{j=0}^{s_r-1} AO_{\frac{s_h}{j+1},j}(P, r_0 z \cdot (j^2 + j)). \tag{9.3}$$

The only parameters s_r, s_h, and r_0 were described in the LineAO parameter summary at Section 8.3.2. They represent the number of hemispheres (s_r), the number of samples per hemisphere (s_h), and the smallest sampling radius (r_0).

9.3 Method

To achieve the effects of LineAO for glyph rendering, we need to change three parts of the original algorithm.

Sampling Scheme. As shown above, LineAO reduces the amount of samples taken into account for each, increasing hemisphere. This can be done since the algorithm uses a Gaussian pyramid for depth- and normal-maps. Thick, more distant bundles merge to single objects in higher levels of the pyramid, thus, reducing the probability to miss them during sampling. This is not the case for glyphs and points. Even dense areas might contain a lot of holes, which do not quickly vanish in higher Gauss-pyramid levels. Hence, there is a need for more samples, even for distant occluders. To achieve this, we change the term $\frac{s_h}{j+1}$ in Equation (9.3), which defines the decreasing number of samples used in Equation (9.2) to s_h. In other words, PointAO does not decrease the number of samples per hemisphere, but always uses s_h samples. This ensures a better sampling of distant occluders in point data.

Radius Scaling. The original algorithm as published in 2013 increased the radius linearly using $r_0 + jz(P)$, where r_0 is a predefined minimal radius, $j \geq 0$ the current hemisphere, and $z(P)$ a function denoting the zoom of the scene. In Chapter 8 we already changed this to the improved version $r_0 z \cdot (j^2 + j)$, as shown in Equation (8.7) and Equation (9.3) respectively.

In PointAO, we use $\frac{1}{1-d_j(P)} \cdot z(P) \cdot (j^2 + j \cdot r_0)$ as the radius parameter for each hemisphere, where $d_j(P)$ is the depth of the pixel P with d being 1 at the far clipping plane and 0 at the front. The term $\frac{1}{1-d_j(P)}$ causes pixels to the front to use a smaller radius, thus, containing more local detail, whereas pixels further back are influenced more by distant occluders. This creates a crisp shading at prominent glyphs in front and a smooth shadow on glyphs in the back.

Weighting function. Finally, we modified the weighting function $g_l(\omega, P)$ from Equation (8.10), which weights the occluder influence on P for a given sampling direction ω at a certain distance-level l. It was a combination of depth-based weighting and light-based weighting. The light-based weighting benefits from the fact that bundles of lines merge to surface-like objects with useful normals on them for increasing distance-level. Due to their shape, spherical glyphs scatter the light too much, causing it to be over-estimated

in LineAO. Instead, we combine the depth-based weight with $\langle \omega, n_l(P) \rangle$, as in Equation (9.1) to better retain the glyphs shape in the shading. This yields

$$g_l(\omega, P) = g_l^{depth}(\omega, P) \cdot \langle \omega, n_l(P) \rangle. \tag{9.4}$$

as the weighting function for occluders in PointAO, where $g_l^{depth}(\omega, P)$ is the same depth-based weight as in LineAO (cf. Equation (8.13)).

Implementation We have implemented PointAO the same way as LineAO. This was described in Section 8.3.4. The above mentioned changes were incorporated in the LineAO fragment shader. Everything else was left untouched. The source codes are available in OpenWalnut (see Chapter 3 for details or online at `www.openwalnut.org`).

9.4 Results

To demonstrate our technique, we apply the PointAO rendering to four types of data. We compare our method to the well known and widely-used Crytek SSAO [131] approach and Blinn-Phong shaded [18] spherical glyphs. The performance measures were taken on a GeForce GTX 480, a lower to medium-level consumer graphics card (2014). We are using FullHD resolution and naive rendering, without any optimization towards occlusion-culling or similar geometry reduction techniques. Table 9.1 summarizes the shown examples and their frame rates for the different rendering techniques.

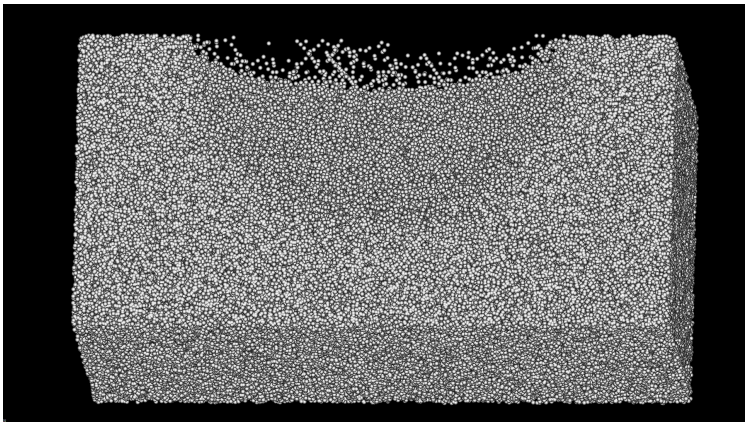
DTI Super-Quadric Glyphs. Figure 9.1 shows a comparison of Phong shaded super-quadric glyphs [92], rendered purely on the GPU [76]. The rendering shows three orthogonal slices through a DTI second-order tensor dataset. Although the glyphs are aligned in a regular grid and the glyphs do not contain explicit spatial information, it is important to grasp the anatomical context and structures contained in the data. PointAO's improved rendering allows to see the underlying anatomical details as well as the slices itself. The plain Phong-rendered image did not unveil this information.

Argon Bubble. Figure 9.2 shows a cut through a cube of simulated argon fluid, which contains an argon gas bubble. The simulation uses a truncated Lennard-Jones pair [89] potential for the intermolecular repulsion and short-range dispersive attraction. The standard Phong-shaded spherical glyphs allow

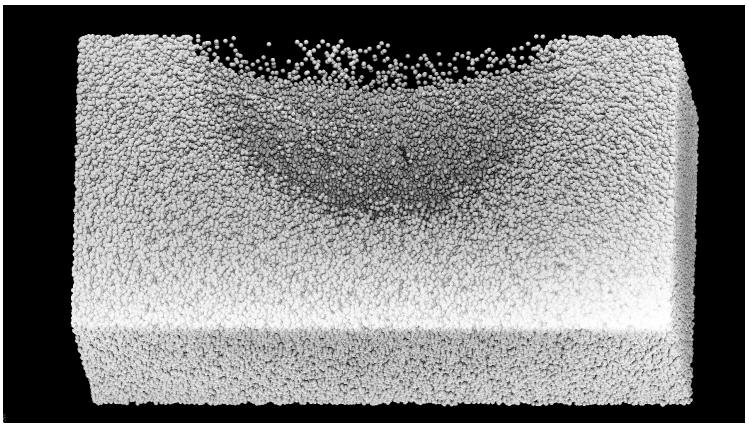
only to guess that there is a cavity inside the block. Although SSAO clearly shows the cavity, it provides no further local detail due to the very large radius needed. Our PointAO approach keeps the global spatial information and shows the cavity but also unveils the flowing argon gas atoms, which are not clearly visible in the other approaches.

Flow Particle Data. Figure 9.3 shows traced particles in the leading edge vortices of an inclined delta wing at four different time steps. Tracers are seeded on an initial plane (gray) and shown after different time intervals. The simple glyph rendering does not provide any cue on the spatial structure of the particles inside the vortex, nor does it provide any cue for estimating depth distance between the different particle planes and the vortices. The difference between the SSAO and PointAO renderings are rather subtle at a first glance. The SSAO rendering provides a structural cue inside the vortices and a subtle global shading between the planes. This is caused by the very solid-geometry alike structure, the well chosen SSAO parameters, and an overemphasized AO factor. Generally, SSAO is much more dependent on well chosen algorithm parameters, which vary from dataset to dataset. Choosing a smaller radius for SSAO would emphasize the particles inside the vortex, whereas a larger radius would emphasize the spatial relation between the planes and the vortices as a whole. With PointAO, we are able to emphasize local and global structure in equal measure.

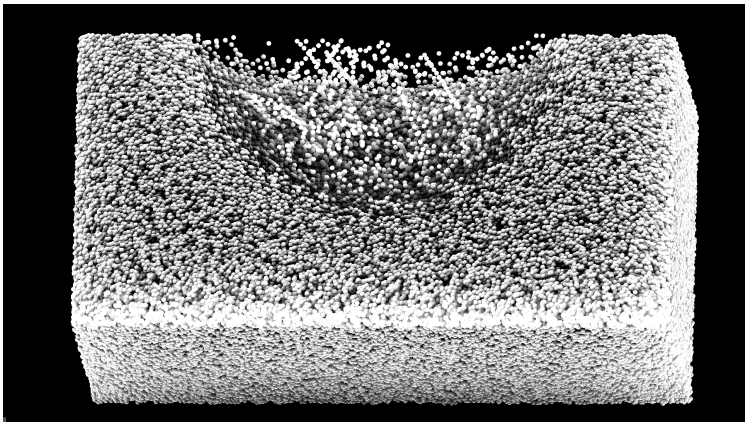
LIDAR Scan. Figure 9.4 shows light detection and ranging (LIDAR) data of the Golden Gate Bride area. It is an imaging technique that acquires millions of points on visible surfaces leading to a point-based reconstruction of objects scanned. Instead of preprocessing the data to obtain surface meshes [4] or rendering the data after assigning surface normals [97], we directly display the point data. Especially in the chosen dataset, surface-based techniques fail to provide sufficient representations of the bridge or the trees in front of the bridge. Due to the sharp edges of the glyphs, the simple glyph rendering already shows the basic structure of the Golden Gate bride area. The SSAO rendering adds additional, very local shading details due to the chosen, small sampling radius. This can be seen especially in the trees area. Our PointAO approach provides these local shading details as well, but adds more global shadows to the scene, e.g., the shadows between the groups of trees and below the bridge, which are not directly visible in the SSAO image.



(a) Phong Shading – 30 FPS

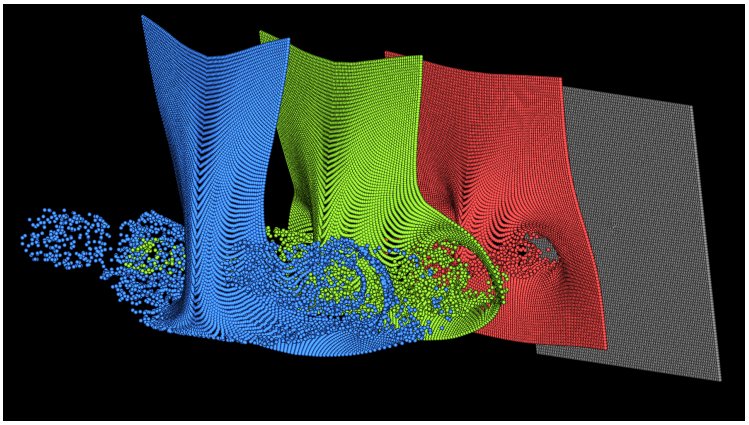


(b) SSAO – 19 FPS

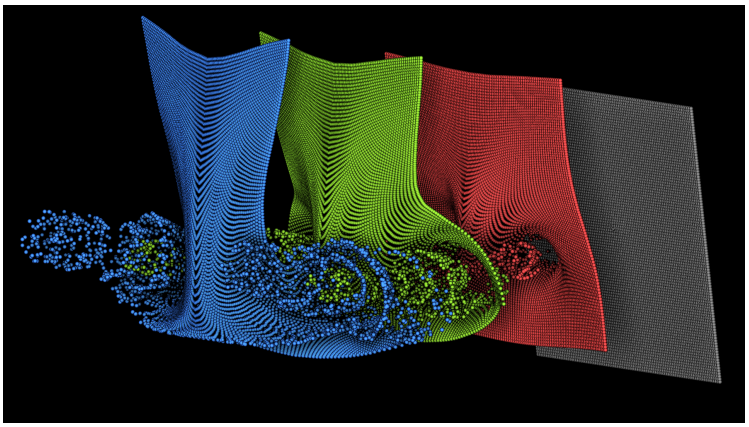


(c) PointAO – 12 FPS

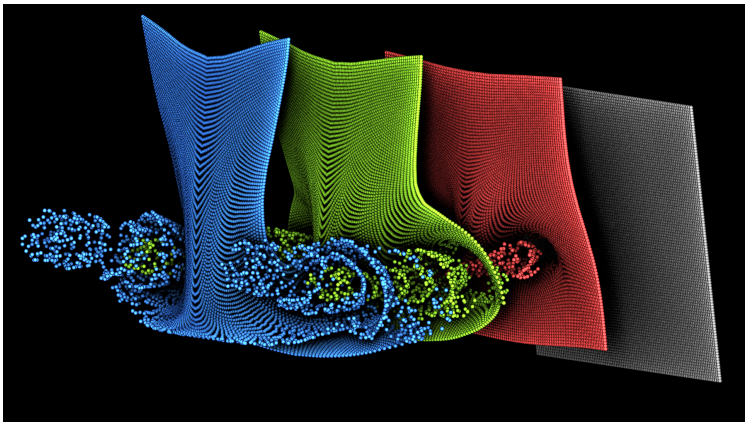
Figure 9.2: *Cut through an Argon fluid with enclosed Argon gas bubble (3 529 344 particles). The standard Phong shaded rendering (a) indicates the gas bubble inside but provides no further spatial information. With SSAO (b), one can clearly see the cavity inside the fluid, but local structures are nearly invisible and the gas particles inside the cavity can only be depicted clearly in the PointAO (c) rendering.*



(a) Phong Shading – 73 FPS

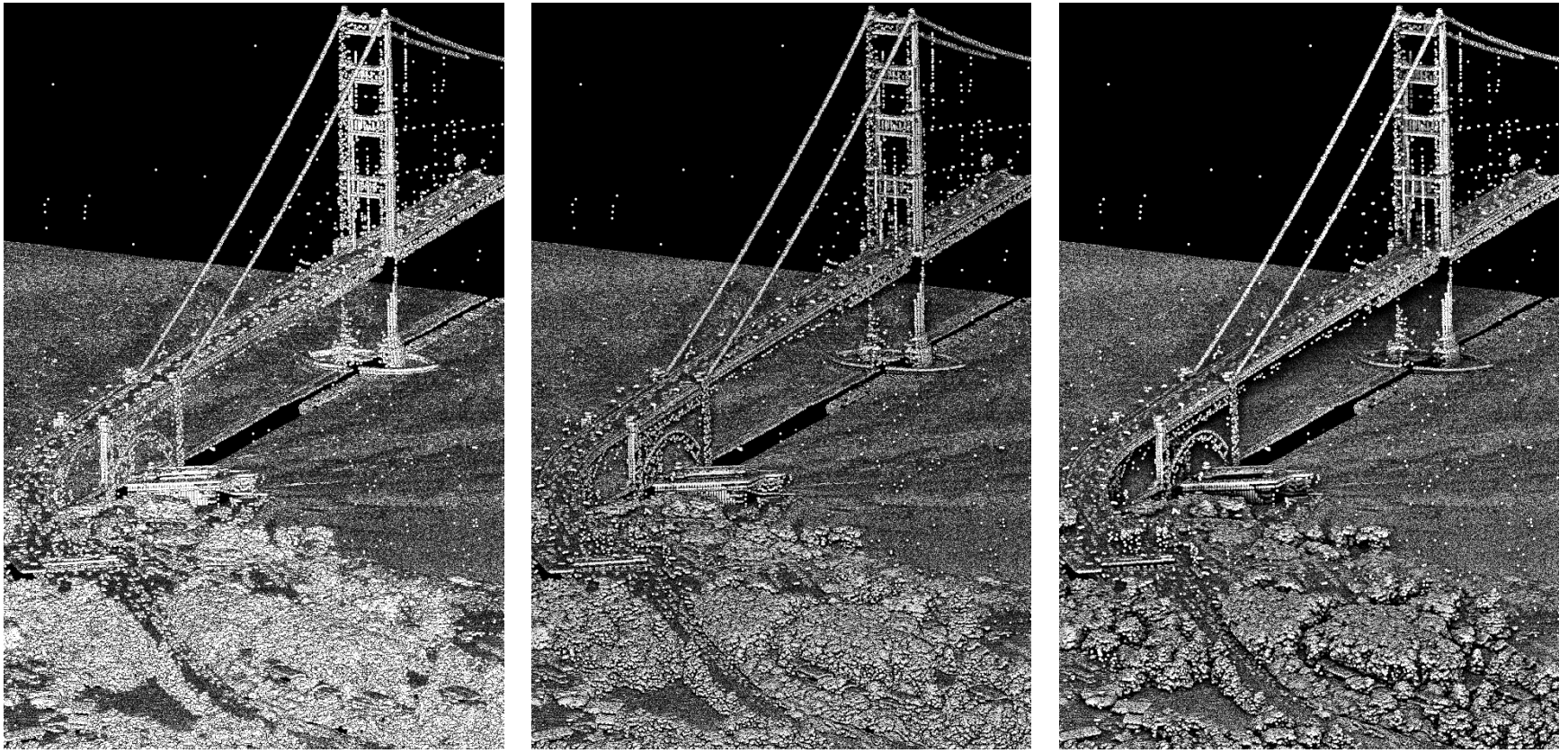


(b) SSAO – 29 FPS



(c) PointAO – 20 FPS

Figure 9.3: *81 554 Particles flowing into the leading edge vortices of an inclined delta wing. The differently colored slices describe different time steps in the data. The SSAO rendering (b) already shows useful spatial information between the different time slices, but to achieve this image, we had to put some effort into fine-tuning the SSAO parameters. Still, the PointAO approach (c) yields much better distinction between the different particles at a local* and *a global scope without parameter-fiddling.*



(a) Phong Shading – 9 FPS (b) SSAO – 7 FPS (c) PointAO – 5 FPS

Figure 9.4: *Light detection and ranging (LIDAR) scan of the Golden Gate bridge area with 14 614 901 sample points. Similar to the above figures, only PointAO is able to reproduce a proper global shading and local shading. This can be seen at the smooth shadow below the bridge and the crisp details in the trees in front of the bridge.*

Figure	Points/Glyphs	FPS Phong	FPS SSAO	FPS LineAO
9.1	Super-Quadric [92]: 82 203	64	22	18
9.2	Spherical: 3 529 344	30	19	12
9.3	Spherical: 81 554	73	29	20
9.4	Spherical: 14 614 901	9	7	5

Table 9.1: *Performance of PointAO for several datasets in comparison to plain Phong illuminated glyphs and SSAO [131].*

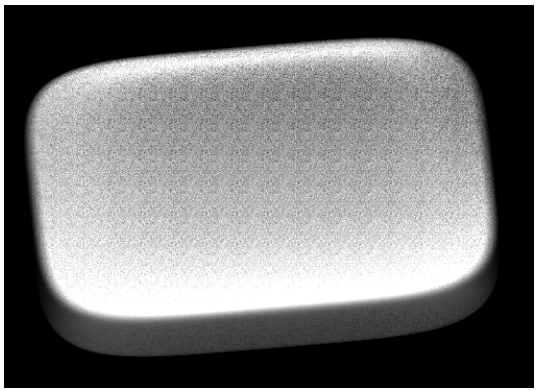


Figure 9.5: *Sampling artifacts on a glyph zoomed to screen size. Although the artifacts are quite subtle, they are visible in the pattern of the shading. The effect can be alleviated by increasing the amount of samples per hemisphere s_h.*

9.5 Discussion

9.5.1 Limitations and Problems

As PointAO is an extension to LineAO, it has the same limitations. A major difference though, is that glyphs have a spatial extend and are affected by zooming. They get larger when zooming in and their surfaces begin to show the sampling pattern used. Figure 9.5 shows this effect. On small and thin structures, this is not an issue. However, viewing single glyphs at these sizes is a rather rare case and the visible sampling artifacts are not that strong.

9.5.2 Future Work

Similar to the previous section, PointAO shares its possible future research directions with LineAO. Especially interesting for PointAO is, whether the improved sampling scheme, indicated in Section 8.5.2, can make the specific PointAO sampling scheme superfluous.

9.6 Conclusion

We have presented an improved version of the LineAO algorithm, which is optimized towards ***real-time rendering of point-based data***. It is able to handle ***arbitrary types of glyphs*** and does ***not need any precomputation***, thus, making it ideal for interactive exploration and filtering of data. It allows simultaneous depiction of ***local and global spatial relations and structure*** in the data, while being ***consistent under modification and interaction***. The possibility of combining PointAO with other interactive methods allows it to be used as ***add-on to existing approaches*** in interactive and explorative visualization environments.

LineAO as well as PointAO are a considerable step towards the improvement of visualization with regard to visual quality, structural, and spatial perception.

10 Thesis Conclusions

The importance of visualization in many areas of science cannot be disclaimed. It is a very powerful tool to grasp and understand the structure of data from a myriad of different sources. It allows for an effective conveyance of complex data and enables quick qualitative and quantitative assessments. Visualization can unveil structures and properties inside the data that statistical measures cannot. One can see visualization as the interface between the data and the human mind.

During my research, I intensively collaborated with neuroscientists and learned that visualization is a very application-dependent science. In fact, other scientists see visualization as a tool to support them in analyzing their data, which is true, but falls short on the *science* behind this "tool". Especially in neuroscience, the number of actively used visualization techniques is rather small. In my experience, this is not necessarily caused by the techniques being unknown – instead, these techniques are simply

- not proven to be useful in the specific application area,
- too parameter-dependent and complicated,
- and often not available/accessible.

Tackling this was the major goal of the first part of my thesis. We developed OpenWalnut to implement different common and novel visualization techniques to make them *available and accessible* by neuroscientists. We presented a tool that was designed to be the common platform for us and our neuroscientific collaborators. Today, it has reached a stable state and is used by several groups for research in different areas, not only neuroscience. We developed a novel visualization approach for functional neuroscience and, although it certainly is not a new de-facto standard visualization technique, it once more shows the importance of application-specific visualizations. It helped the scientists to visually understand the anatomical meaning of the formerly abstract models. Finally, we have evaluated some of the most well known visualization techniques in the context of three neuroscientifically relevant scenarios. We published this in one of the most important journals of the neuroscience community to reach the real target community – neuroscientists. We have shown the possibilities of visualization and also made clear that every method, every visualization has its limitations and advantages for a certain task. This was not done before in the neuroscience community and triggered the interest of several external groups on our work, visualization and OpenWalnut.

Visualization is the combination of smart data processing ***and*** computer graphics. The processing methods behind modern visualization get smarter, faster, and better day by day. However, in my opinion and expressed with some exaggeration, the graphical representation is often still at the level of the mid '80s, when Silicon Graphics introduced their IRIS graphics workstation series. The crux of this statement gets obvious, once looking at the possibilities of modern computer graphics in comparison to the graphical outcome of today's visualization techniques. I am not saying that the graphics in visualization are bad, but as a matter of fact, they do not exploit the whole spectrum of today's possibilities.

This was the reason for working on improvements of existing visualization paradigms and techniques. The focus lied on the improvement of structural perception, spatiality, and better visual detection of relations in the data. This was furthermore fortified by our neuroscientist collaborators, stating that three-dimensional visualization techniques are often of limited use, since they do not provide the structural coherency with anatomy and context; not to mention the missing spatial relationships.

We have shown that computer graphics techniques help to improve the structural perceptibility on surfaces at the example of TensorMesh, representing a whole class of surface-based techniques. With LineAO and PointAO, we continued to use the screen space paradigm to improve the spatial perception in line and point data. We achieved to create a technique, able to represent local and global structures in spatial relation to each other; in a very intuitive and natural way. Especially for line data, this was not possible before.

Beside its scientific achievement, LineAO also attracted attention in the press, simply because the rendered images are visually appealing. LineAO was shown in LeMonde Science Online, 2012: la science en images [P10] and is on the cover of the 2014 volume of the Discoveries Magazine, the UC San Diego Health Sciences' publication about their innovations in research, health care and education [P12].

We showed that screen space postprocessing is a valuable tool to improve existing rendering techniques. They do not only create "nice looking pictures", but also improve the value of visualization. Expressiveness of visualization is not only defined by its smart data processing in the background, but also by its graphical representation and how effective the graphical representation transports information.

List of Publications

List of my original publications as of December 8, 2014. The list is sorted by appearance in the chapters of this thesis. The publications are available online at http://www.sebastian-eichelbaum.de/publications.

[P1] S. EICHELBAUM, M. GOLDAU, S. PHILIPS, A. REICHENBACH, R. SCHURADE, and A. WIEBEL. **OpenWalnut: A New Tool for Multimodal Visualization of the Human Brain**. *EG VCBM 2010 Posters*. 2010.

[P2] S. EICHELBAUM, M. HLAWITSCHKA, A. WIEBEL, and G. SCHEUERMANN. **OpenWalnut - An Open-Source Visualization System**. *Proceedings of the 6th High-End Visualization Workshop*. Ed. by W. Benger, A. Gerndt, S. Su, W. Schoor, M. Koppitz, W. Kapferer, H.-P. Bischof, and M. D. Pierro. 2010, 67–78.

[P3] S. EICHELBAUM, M. HLAWITSCHKA, and G. SCHEUERMANN. **OpenWalnut: An Open-Source Tool for Visualization of Medical and Bio-Signal Data**. *Biomedical Engineering / Biomedizinische Technik*. Ed. by O. Dössel. 2013.

[P4] S. EICHELBAUM, A. WIEBEL, M. HLAWITSCHKA, A. ANWANDER, T. KNÖSCHE, and G. SCHEUERMANN. **Visualization of Effective Connectivity of the Brain**. *Proceedings of the 15th International Workshop on Vision, Modeling and Visualization (VMV) Workshop 2010*. Ed. by R. Koch, A. Kolb, and C. Rezk-Salama. 2010, 155–162.

[P5] S. EICHELBAUM, M. DANNHAUER, M. HLAWITSCHKA, D. BROOKS, T. R. KNÖSCHE, and G. SCHEUERMANN. **Visualizing Simulated Electrical Fields from Electroencephalography and Transcranial Electric Brain Stimulation: A Comparative Evaluation**. *NeuroImage* 101 (2014), 513–530. ISSN: 1053-8119.

[P6] S. EICHELBAUM, M. DANNHAUER, G. SCHEUERMANN, D. BROOKS, T. R. KNÖSCHE, and M. HLAWITSCHKA. **A Comparative Evaluation of Electrical Field Visualization from EEG/tDCS**. *The 20th Annual Meeting of the Organization for Human Brain Mapping (HBM), Poster 3029*. 2014.

[P7] S. EICHELBAUM, M. HLAWITSCHKA, B. HAMANN, and G. SCHEUERMANN. **Fabric-like Visualization of Tensor Field Data on Arbitrary Surfaces in Image Space**. *New Developments in the Visualization and Processing of Tensor Fields*. Ed. by D. H. Laidlaw and A. Vilanova. *Mathematics and Visualization*. 2012, 71–92.

[P8] S. EICHELBAUM, M. HLAWITSCHKA, B. HAMANN, and G. SCHEUERMANN. **Image-space Tensor Field Visualization Using a LIC-like Method**. *Visualization in Medicine and Life Sciences 2*. Ed. by L. Linsen, B. Hamann, H. Hagen, and H.-C. Hege. *Mathematics and Visualization*. 2012, 193–210.

[P9] S. EICHELBAUM, M. HLAWITSCHKA, and G. SCHEUERMANN. **LineAO – Improved Three-Dimensional Line Rendering**. *IEEE Transactions on Visualization and Computer Graphics* 19.3 (2013), 433–445.

[P10] S. EICHELBAUM, M. HLAWITSCHKA, and G. SCHEUERMANN. **Vue en tractographie d'un cerveau humain.** *LeMonde Science Online, 2012: la science en images*. 2012.

[P11] S. EICHELBAUM, J. KASTEN, M. HLAWITSCHKA, G. SCHEUERMANN, and B. R. NOACK. **Leading edge vortices of flow over a delta wing**. *Gallery of Fluid Motion, Poster, P55*. 2012.

[P12] S. EICHELBAUM, M. HLAWITSCHKA, and G. SCHEUERMANN. **Cover picture on Discoveries Magazine.** *Discoveries Magazine, Volume 5*. 2014.

[P13] S. EICHELBAUM, G. SCHEUERMANN, and M. HLAWITSCHKA. **PointAO – Improved Ambient Occlusion for Point-based Visualization**. *EuroVis - Short Papers*. Ed. by M. Hlawitschka and T. Weinkauf. 2013, 13–17.

List of Talks

List of my given talks as of December 8, 2014. The list is sorted chronologically. The talks are available online, alongside the respective publication at http://www.sebastian-eichelbaum.de/publications.

[T1] **Image Space Tensor Field Visualization using a LIC-like Method**. July 2009, Visualization in Medicine and Life Sciences 2009 in Bremerhaven, Germany.

[T2] **Visualization of Effective Connectivity of the Brain**. November 2010, 15th International Workshop on Vision, Modeling and Visualization in Siegen, Germany.

[T3] **Image Space Tensor Field Visualization using a LIC-like Method**. December 2010, 6th High End Visualization Workshop in Obergurgl, Austria.

[T4] **LineAO – Improved Three-Dimensional Line Rendering**. October 2012, IEEE VisWeek 2012 in Seattle, Washington, USA.

[T5] **LineAO – Improved Three-Dimensional Line Rendering**. October 2012, Lawrence Berkeley National Laboratory, Berkeley, California, USA.

[T6] **PointAO – Improved Ambient Occlusion for Point-based Visualization**. June 2013, EuroVis – The Eurographics Conference on Visualization 2013 in Leipzig, Germany.

[T7] **OpenWalnut – An Open-Source Tool for Visualization of Medical and Bio-Signal Data.** September 2013, Dreiländertagung der Deutschen, Schweizerischen und Österreichischen Gesellschaft für Biomedizinische Technik in Graz, Austria.

List of Figures

List of Tables

Bibliography

[1] J. AHRENS, B. GEVECI, and C. LAW. **ParaView: An End-User Tool for Large Data Visualization**. *Visualization Handbook*. Ed. by C. Hansen and C. Johnson. Elsevier, 2005.

[2] D. AKERS, A. SHERBONDY, R. MACKENZIE, R. DOUGHERTY, and B. WANDELL. **Exploration of the Brain's White Matter Pathways with Dynamic Queries**. *VIS '04: Proceedings of the conference on Visualization '04*. Washington, DC, USA: IEEE Computer Society, 2004, 377–384. ISBN: 0-7803-8788-0.

[3] M. AKHTARI, H. BRYANT, A. MAMELAK, E. FLYNN, L. HELLER, J. SHIH, M. MANDELKEM, A. MATLACHOV, D. RANKEN, E. BEST, M. DIMAURO, R. LEE, and W. SUTHERLING. **Conductivities of Three-Layer Live Human Skull**. *Brain Topography* 14 (3 2002), 151–167. ISSN: 0896-0267.

[4] M. ALEXA, J. BEHR, D. COHEN-OR, S. FLEISHMAN, D. LEVIN, and C. T. SILVA. **Point set surfaces**. *Proceedings of the conference on Visualization '01*. *VIS '01*. San Diego, California: IEEE Computer Society, 2001, 21–28. ISBN: 0-7803-7200-X.

[5] **Amira - Visualize Analyze Present**.
URL: http://www.amira.com/.

[6] A. ANWANDER, R. SCHURADE, M. HLAWITSCHKA, G. SCHEUERMANN, and T. KNÖSCHE. **White Matter Imaging with Virtual Klingler Dissection**. *NeuroImage* 47.Supplement 1 (2009). Organization for Human Brain Mapping 2009 Annual Meeting, S105–S105. ISSN: 1053-8119.

[7] O. ARIKAN, D. A. FORSYTH, and J. F. O'BRIEN. **Fast and detailed approximate global illumination by irradiance decomposition**. *ACM*

SIGGRAPH 2005 Papers. *SIGGRAPH '05*. Los Angeles, California: ACM, 2005, 1108–1114.

[8] N. BANGERA, D. SCHOMER, N. DEHGHANI, I. ULBERT, S. CASH, S. PAPAVASILIOU, S. EISENBERG, A. DALE, and E. HALGREN. **Experimental validation of the influence of white matter anisotropy on the intracranial EEG forward solution**. *Journal of Computational Neuroscience* (2010). ISSN: 0929-5313.

[9] P. J. BASSER, S. PAJEVIC, C. PIERPAOLI, J. DUDA, and A. ALDROUBI. **In vivo fiber tractography using DT-MRI data.** *Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine* 44.4 (2000), 625–632. ISSN: 0740-3194.

[10] D. R. BAUM, H. E. RUSHMEIER, and J. M. WINGET. **Improving radiosity solutions through the use of analytically determined form-factors**. *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*. *SIGGRAPH '89*. New York, NY, USA: ACM, 1989, 325–334. ISBN: 0-89791-312-4.

[11] S. BAUMANN, D. WOZNY, S. KELLY, and F. MENO. **The Electrical Conductivity of Human Cerebrospinal Fluid at Body Temperature**. *IEEE Transactions on Biomedical Engineering* 44.3 (1997), 220–3.

[12] L. BAVOIL and M. SAINZ. **Multi-layer dual-resolution screen-space ambient occlusion**. *SIGGRAPH 2009: Talks*. *SIGGRAPH '09*. New Orleans, Louisiana: ACM, 2009, 45:1–45:1. ISBN: 978-1-60558-834-6.

[13] W. BENGER and H.-C. HEGE. **Tensor Splats**. *Conference on Visualization and Data Analysis*. Ed. by Erbacher, Chen, Roberts, Grohn, and Borner. 5295. *Proceedings of SPIE*. 2004, 151–162.

[14] H. BERGER. **Über das Elektroenkephalogram des Menschen**. *Archiv für Psychiatrie und Nervenkrankheiten* 99.1 (1933), 555–74.

[15] J. BERMAN, M. BERGER, P. MUKHERJEE, and R. HENRY. **Diffusion-tensor imaging-guided tracking of fibers of the pyramidal tract combined with intraoperative cortical stimulation mapping in patients with gliomas**. *Neurosurgery* 101.1 (2004), 66–72.

[16] O. BERTRAND. **3D Finite element method in brain electrical activity studies.** *Biomagnetic localization and 3D Modeling* 1 (1991), 154–171.

[17] J. BLAAS, C. P. BOTHA, B. PETERS, F. M. VOS, and F. H. POST. **Fast and Reproducible Fiber Bundle Selection in DTI Visualization**. *Visualization Conference, IEEE* (2005), 59–64.

[18] J. F. BLINN. **Models of light reflection for computer synthesized pictures**. *SIGGRAPH '77: Proceedings of the 4th annual conference on Computer graphics and interactive techniques*. San Jose, California: ACM, 1977, 192–198.

[19] J. F. BLINN. **Simulation of wrinkled surfaces**. *SIGGRAPH Comput. Graph.* 12.3 (1978), 286–292. ISSN: 0097-8930.

[20] P. BOGGIO, R. FERRUCCI, S. RIGONATTI, P. COVRE, M. NITSCHE, A. PASCUAL-LEONE, and F. FREGNI. **Effects of transcranial direct current stimulation on working memory in patients with Parkinson's disease**. *Journal of the Neurological Sciences* 249 (2006), 31–38.

[21] G. BRADSHAW and C. O'SULLIVAN. **Sphere-tree construction using dynamic medial axis approximation**. *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation. SCA '02*. San Antonio, Texas: ACM, 2002, 33–40. ISBN: 1-58113-573-4.

[22] J. E. BRESENHAM. **Algorithm for Computer Control of a Digital Plotter**. *IBM System Journal* 4.1 (1965), 25–30.

[23] S. P. BROEK, F. REINDERS, M. DONDERWINKEL, and M. J. PETERS. **Volume conduction effects in EEG and MEG.** *Electroencephalography and Clinical Neurophysiology* 106.6 (1998), 522–534.

[24] J. BRONSON, J. LEVINE, and R. WHITAKER. **Lattice Cleaving: Conforming Tetrahedral Meshes of Multimaterial Domains with Bounded Quality**. *International Meshing Roundtable*. 2012, 191–209.

[25] A. BRUNONI, P. BOGGIO, R. FERRUCCI, A. PRIORI, and F. FREGNI. **Transcranial Direct Current Stimulation: Challenges, Opportunities, and Impact on Psychiatry and Neurorehabilitation**. *Frontiers in Psychiatry* 4.19 (2011).

[26] A. BRUNONI, M. NITSCHE, N. BOLOGNINI, M. BIKSON, T. WAGNER, L. MERABET, D. EDWARDS, A. VALERO-CABRE, A. ROTENBURG, A. PASCUAL-LEONE, R. FERRUCCI, A. PRIORI, P. BOGGIO, and F. FREGNI. **Clinical research with transcranial direct current stimulation (tDCS): Challenges and future directions**. *Brain Stimulation* 5.3 (2012), 175–95.

[27] M. BUNNELL. **GPU Gems 2**. Addison-Wesley Professional, 2005. *14. Dynamic Ambient Occlusion and Indirect Lighting*.

[28] B. CABRAL and L. C. LEEDOM. **Imaging vector fields using line integral convolution**. *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. Anaheim, CA: ACM, 1993, 263–270. ISBN: 0-89791-601-8.

[29] E. CAPARELLI-DAQUER, T. ZIMMERMANN, E. MOOSHAGIAN, L. PARRA, J. PARRA, J. RICE, A. DATTA, M. BIKSON, and E. WASSERMANN. **A pilot study on effects of 4x1 High-Definition tDCS on motor cortex excitability.** *Conference Proceedings - IEEE Engineering in Medicine and Biology Society*. IEEE. 2012, 735–8.

[30] A. CHU, W.-Y. CHAN, J. GUO, W.-M. PANG, and P.-A. HENG. **Perception-aware Depth Cueing for Illustrative Vascular Visualization**. *BMEI '08: Proceedings of the 2008 International Conference on BioMedical Engineering and Informatics*. Washington, DC, USA: IEEE Computer Society, 2008, 341–346. ISBN: 978-0-7695-3118-2.

[31] P. CIGNONI, P. MARINO, C. MONTANI, E. PUPPO, and R. SCOPIGNO. **Speeding Up Isosurface Extraction Using Interval Trees**. *IEEE Transactions on Visualization and Computer Graphics* 3 (2 1997), 158–170. ISSN: 1077-2626.

[32] R. L. COOK and K. E. TORRANCE. **A Reflectance Model for Computer Graphics**. *ACM Trans. Graph.* 1.1 (1982), 7–24. ISSN: 0730-0301.

[33] M. DANNHAUER, D. BROOKS, and R. MACLEOD. **A pipeline for the Simulation of Transcranial Direct Current Stimulation for Realistic Human Head Models using SCIRun/BioMesh3D**. *Conference Proceedings - IEEE Engineering in Medicine and Biology Society*. 2012, 5486–5489.

[34] M. DANNHAUER, E. LÄMMEL, C. WOLTERS, and T. R. KNÖSCHE. **Spatio-temporal Regularization in Linear Distributed Source Reconstruction from EEG/MEG: A Critical Review**. *Brain Topograghy* 26 (2013), 229–46.

[35] M. DANNHAUER, B. LANFER, C. WOLTERS, and T. KNÖSCHE. **Modeling of the human skull in EEG source analysis**. *Human Brain Mapping* 32.9 (2011), 1383–1399.

[36] A. DATTA, J. BAKER, M. BIKSON, and J. FRIDRIKSSON. **Individualized model predicts brain current flow during transcranial direct-current stimulation treatment in responsive stroke patient**. *Brain Stimulation* 4 (2011), 169–74.

[37] A. DATTA, M. BIKSON, and F. FREGNI. **Transcranial direct current stimulation in patients with skull defects and skull plates: high-resolution computational FEM study of factors altering cortical current flow**. *NeuroImage* 52 (2010), 1268–78.

[38] A. DATTA, D. TRUONG, P. MINHAS, L. PARRA, and M. BIKSON. **Inter-Individual Variation during Transcranial Direct Current Stimulation and Normalization of Dose Using MRI-Derived Computational Models**. *Frontiers in Psychiatry* 3.91 (2012), 176–83.

[39] A. DATTA, X. ZHOU, Y. SU, L. PARRA, and M. BIKSON. **Validation of finite element model of transcranial electrical stimulation using scalp potentials: implications for clinical dose**. *Journal of Neuronal Engineering* 10.3 (2013).

[40] DELEVOPER-GROUP-SIMBIO. **Simbio: A generic environment for bio-numerical simulations**.
URL: https://www.mrt.uni-jena.de.

[41] T. DELMARCELLE and L. HESSELINK. **Visualization of second order tensor fields and matrix data**. *VIS '92: Proceedings of the 3rd conference on Visualization '92*. Boston, Massachusetts: IEEE Computer Society Press, 1992, 316–323. ISBN: 0-8186-2896-0.

[42] T. DELMARCELLE and L. HESSELINK. **Visualizing Second-Order Tensor Fields with Hyperstreamlines**. *IEEE Comput. Graph. Appl.* 13 (4 1993), 25–33. ISSN: 0272-1716.

[43] C. DICK, J. GEORGII, R. BURGKART, and R. WESTERMANN. **Stress Tensor Field Visualization for Implant Planning in Orthopedics**. *IEEE Transactions on Visualization and Computer Graphics* 15.6 (2009), 1399–1406. ISSN: 1077-2626.

[44] Y. I. DIMITRIENKO. **Tensor Analysis and Nonlinear Tensor Functions**. Kluwer Academic Publishers (Springer), 2002.

[45] P. DOBREV, P. ROSENTHAL, and L. LINSEN. **An Image-space Approach to Interactive Point Cloud Rendering Including Shadows and Transparency**. *Computer Graphics and Geometry* 12.3 (2010), 2–25.

[46] J. DORMAND and P. PRINCE. **A family of embedded Runge-Kutta formulae**. *Journal of Computational and Applied Mathematics* 6.1 (1980), 19–26. ISSN: 0377-0427.

[47] D. EDWARDS, M. CORTES, A. DATTA, P. MINHAS, E. WASSERMANN, and M. BIKSON. **Physiological and modeling evidence for focal transcranial electrical brain stimulation in humans: A basis for high-definition tDCS**. *NeuroImage* 74 (2013), 266–75.

[48] S. EICHELBAUM. **Image Space Tensor Field Visualization Using a LIC-like Method**. Diploma Thesis. Fakultät für Mathematik und Informatik Universität Leipzig, 2009.
URL: http://www.sebastian-eichelbaum.de/pub09.

[49] F. ENDERS, N. SAUBER, D. MERHOF, P. HASTREITER, C. NIMSKY, and M. STAMMINGER. **Visualization of White Matter Tracts with Wrapped Streamlines**. *Proceedings of IEEE Visualization 2005*. Ed. by C. T. Silva, E. Gröller, and H. Rushmeier. IEEE Computer Society. Los Alamitos, CA, USA: IEEE Computer Society Press, 2005, 51–58.

[50] K. ENGEL, M. HADWIGER, J. KNISS, C. REZK-SALAMA, and D. WEISKOPF. **Real-time Volume Graphics**. A K Peters, 2006.

[51] A. EVANS. **Fast approximations for global illumination on dynamic scenes**. *ACM SIGGRAPH 2006 Courses*. *SIGGRAPH '06*. Boston, Massachusetts: ACM, 2006, 153–171. ISBN: 1-59593-364-6.

[52] M. H. EVERTS, H. BEKKER, J. B. ROERDINK, and T. ISENBERG. **Depth-Dependent Halos: Illustrative Rendering of Dense Line Data**. *IEEE Transactions on Visualization and Computer Graphics* 15.06 (2009), 1299–1306. ISSN: 1077-2626.

[53] M. FALK and D. WEISKOPF. **Output-Sensitive 3D Line Integral Convolution**. *IEEE Transactions on Visualization and Computer Graphics* 14 (2008), 820–834. ISSN: 1077-2626.

[54] **Fiber Navigator**.
URL: http://code.google.com/p/fibernavigator.

[55] D. FILION and R. MCNAUGHTON. **Effects & techniques**. *ACM SIGGRAPH 2008 classes*. *SIGGRAPH '08*. Los Angeles, California: ACM, 2008, 133–164.

[56] A. FLÖEL. **tDCS-enhanced motor and cognitive function in neurological diseases**. *NeuroImage* 85.3 (2014), 934–47.

[57] K. J. FRISTON, A. MECHELLI, R. TURNER, and C. J. PRICE. **Nonlinear responses in fMRI: the Balloon model, Volterra kernels, and other hemodynamics.** *Neuroimage* 12.4 (2000), 466–477. ISSN: 1053-8119.

[58] K. FRISTON, L. HARRISON, and W. PENNY. **Dynamic causal modelling**. *NeuroImage* 19.4 (2003), 1273–1302. ISSN: 1053-8119.

[59] M. FUCHS, M. WAGNER, and J. KASTNER. **Development of volume conductor and source models to localize epileptic foci.** eng. *Journal of Clinical Neurophysiology* 24.2 (2007), 101–119.

[60] E. GAMMA, R. HELM, and R. E. JOHNSON. **Design Patterns. Elements of Reusable Object-Oriented Software**. Addison-Wesley Longman, 1994.

[61] S. GERHARD, L. CAMMOUN, J.-P. THIRAN, and P. HAGMANN. **ConnectomeViewer.org**. Ecole Polytechnique Fédérale de Lausanne and University Hospital Center and University of Lausanne. 2010.
URL: http://www.connectomics.org/.

[62] M. GRABNER and R. S. LARAMEE. **Image Space Advection on graphics hardware**. *SCCG '05: Proceedings of the 21st spring conference on Computer graphics*. Budmerice, Slovakia: ACM, 2005, 77–84.

[63] R. A. GRANGER. **Fluid Mechanics**. Dover Publications, 1995.

[64] C. P. GRIBBLE and S. G. PARKER. **Enhancing interactive particle visualization with advanced shading models**. *Proceedings of the 3rd symposium on Applied perception in graphics and visualization. APGV '06*. Boston, Massachusetts: ACM, 2006, 111–118. ISBN: 1-59593-429-4.

[65] S. GROTTEL, M. KRONE, K. SCHARNOWSKI, and T. ERTL. **Object-Space Ambient Occlusion for Molecular Dynamics**. *Proceedings of IEEE Pacific Visualization Symposium 2012*. 2012, 209–216.

[66] T. GÜNTHER, C. RÖSSL, and H. THEISEL. **Opacity Optimization for 3D Line Fields**. *ACM Transactions on Graphics (Proc. ACM SIGGRAPH)* 32.4 (2013), 120:1–120:8.

[67] H. HAGEN and C. GARTH. **An Introduction to Tensors**. English. *Visualization and Processing of Tensor Fields*. Ed. by J. Weickert and H. Hagen. *Mathematics and Visualization*. Springer Berlin Heidelberg, 2006, 3–13. ISBN: 978-3-540-25032-6.

[68] P. HAGMANN, L. CAMMOUN, X. GIGANDET, R. MEULI, C. J. HONEY, V. J. WEDEEN, and O. SPORNS. **Mapping the Structural Core of Human Cerebral Cortex**. *PLoS Biol* 6.7 (2008), e159.

[69] Y. O. HALCHENKO and M. HANKE. **Open is not enough. Let's take the next step: An integrated, community-driven computing platform for neuroscience**. *Frontiers in Neuroinformatics* 6.22 (2012). ISSN: 1662-5196.

[70] H. HALLEZ, B. VANRUMSTE, P. V. HESE, S. DELPUTTE, and I. LEMAHIEU. **Dipole estimation errors due to differences in modeling anisotropic conductivities in realistic head models for EEG source analysis.** eng. *Physics in Medicine and Biology* 53.7 (2008), 1877–1894.

[71] K. M. HASAN, P. J. BASSER, D. L. PARKER, and A. L. ALEXANDER. **Analytical Computation of the Eigenvalues and Eigenvectors in DT-MRI**. *Journal of Magnetic Resonance* 152.1 (2001), 41–47. ISSN: 1090-7807.

[72] J. HAUEISEN, D. TUCH, C. RAMON, P. SCHIMPF, W. WEDEEN, J. GEORGE, and J. BELLIVEAU. **The Influence of Brain Tissue Anisotropy on Human EEG and MEG**. *NeuroImage* 15 (2002), 159–66.

[73] K. HAYES. **The Current Path in Electric Convulsion Shock**. *Arch. Neurol. Psychiat.* 63 (1950), 103–9.

[74] L. HESSELINK, Y. LEVY, and Y. LAVIN. **The Topology of Symmetric, Second-Order 3D Tensor Fields**. *IEEE Transactions on Visualization and Computer Graphics* 3.1 (1997), 1–11. ISSN: 1077-2626.

[75] M. HLAWATSCH, J. E. VOLLRATH, F. SADLO, and D. WEISKOPF. **Coherent Structures of Characteristic Curves in Symmetric Second Order Tensor Fields**. *IEEE Transactions on Visualization and Computer Graphics* 17.6 (2011), 781–794.

[76] M. HLAWITSCHKA, S. EICHELBAUM, and G. SCHEUERMANN. **Fast and Memory Efficient GPU-based Rendering of Tensor Data**. *Proceedings of the IADIS International Conference on Computer Graphics and Visualization 2008*. 2008, 36–42.

[77] M. HLAWITSCHKA, C. GARTH, X. TRICOCHE, G. KINDLMANN, G. SCHEUERMANN, K. I. JOY, and B. HAMANN. **Direct Visualization of Fiber Information by Coherence**. *International Journal of Computer Assisted Radiology and Surgery, CARS, CUARC.08 Special Issue* (2009).

[78] M. HLAWITSCHKA and G. SCHEUERMANN. **HOT-Lines — Tracking Lines in Higher Order Tensor Fields**. *Proceedings of IEEE Visualization 2005*. Ed. by C. T. Silva, E. Gröller, and H. Rushmeier. 2005, 27–34.

[79] T.-D. HOANG and K.-L. LOW. **Multi-resolution screen-space ambient occlusion**. *Proceedings of the 2011 Computer Graphics International Conference*. *CGI '11*. PREPRINT. 2011.

[80] T.-D. HOANG and K.-L. LOW. **Multi-resolution screen-space ambient occlusion**. *Talk at Computer Graphics International Conference 2011*. 2011.

[81] J. HOBEROCK and Y. JIA. **GPU Gems 3**. Addison-Wesley Professional, 2005. *12. High-Quality Ambient Occlusion*.

[82] D. HOLTEN and J. J. WIJK. **A user study on visualizing directed edges in graphs**. *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems*. Boston, MA, USA: ACM, 2009, 2299–2308. ISBN: 978-1-60558-246-7.

[83] I. HOTZ, L. FENG, H. HAGEN, B. HAMANN, K. JOY, and B. JEREMIC. **Physically Based Methods for Tensor Field Visualization**. *VIS '04: Proceedings of the conference on Visualization '04*. Washington, DC, USA: IEEE Computer Society, 2004, 123–130. ISBN: 0-7803-8788-0.

[84] I. HOTZ, Z. X. FENG, B. HAMANN, and K. I. JOY. **Tensor Field Visualization using a fabric-like texture on arbitrary two-dimensional surfaces**. *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*. Ed. by T. Möller, B. Hamann, and R. D. Russel. Springer-Verlag Heidelberg, Germany, 2009.

[85] C. IM, H. JUNG, J. CHOI, S. LEE, and K. JUNG. **Determination of optimal electrode positions for transcranial direct current stimulation (tDCS)**. *Physics in Medicine and Biology* 53 (2008), 219–25.

[86] H. INGO. **Open Life: The Philosophy of Open Source**. Lulu.com, 2006. ISBN: 978-1-84728-611-6.

[87] Y. IWAKIRI, Y. OMORI, and T. KANKO. **Practical Texture Mapping on Free-Form Surfaces**. *PG '00: Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*. Washington, DC, USA: IEEE Computer Society, 2000, 97. ISBN: 0-7695-0868-5.

[88] D. L. JAMES and K. FATAHALIAN. **Precomputing interactive dynamic deformable scenes**. *ACM Trans. Graph.* 22 (3 2003), 879–887. ISSN: 0730-0301.

[89] J. E. JONES. **On the Determination of Molecular Fields. II. From the Equation of State of a Gas**. *Proceedings of the Royal Society of London. Series A* 106.738 (1924), 463–477.

[90] V. KAJALIN. **ShaderX 7**. Ed. by W. Engel. Charles River Media, 2009. *Screen Space Ambient Occlusion*, 413–424.

[91] U. KALU, C. SEXTON, C. LOO, and K. EBMEIER. **Transcranial direct current stimulation in the treatment of major depression: a meta-analysis.** *Psychological Medicine* 42.9 (2012), 1791–800.

[92] G. KINDLMANN. **Superquadric Tensor Glyphs**. *Proceedings of IEEE TCVG/EG Symposium on Visualization 2004*. 2004, 147–154.

[93] D. B. KIRK and W.-m. W. HWU. **Programming Massively Parallel Processors: A Hands-on Approach**. Morgan Kaufmann, 2010. ISBN: 978-0-123-81472-2.

[94] A. M. KNOLL, I. WALD, and C. D. HANSEN. **Coherent multiresolution isosurface ray tracing**. *Vis. Comput.* 25.3 (2009), 209–225. ISSN: 0178-2789.

[95] A. KNOLL, Y. HIJAZI, R. WESTERTEIGER, M. SCHOTT, C. HANSEN, and H. HAGEN. **Volume Ray Casting with Peak Finding and Differential Sampling**. *IEEE Transactions on Visualization and Computer Graphics* 15.6 (2009), 1571–1578. ISSN: 1077-2626.

[96] J. KONTKANEN and S. LAINE. **Ambient occlusion fields**. *Proceedings of the 2005 symposium on Interactive 3D graphics and games*. *I3D '05*. Washington, District of Columbia: ACM, 2005, 41–48. ISBN: 1-59593-013-2.

[97] O. KREYLOS, G. BAWDEN, and L. KELLOGG. **Immersive Visualization and Analysis of LiDAR Data**. *Advances in Visual Computing*. Ed. by G. Bebis, R. Boyle, B. Parvin, D. Koracin, P. Remagnino, F. Porikli, J. Peters, J. Klosowski, L. Arns, Y. Chun, T.-M. Rhyne, and L. Monroe. 5358. *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2008, 846–855. ISBN: 978-3-540-89638-8.

[98] J. KRONANDER, D. JÖNSSON, J. LÖW, P. LJUNG, A. YNNERMAN, and J. UNGER. **Efficient Visibility Encoding for Dynamic Illumination in Direct Volume Rendering : -**. *IEEE Transactions on Visualization and Computer Graphics* (2011). PREPRINT.

[99] M. KRONE, K. BIDMON, and T. ERTL. **Interactive Visualization of Molecular Surface Dynamics**. *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization / Information Visualization 2009)* 15.6 (2009), 1391–1398.

[100] M. KUO, W. PAULUS, and M. NITSCHE. **Therapeutic effects of non-invasive brain stimulation with direct currents (tDCS) in neuropsychiatric diseases**. *NeuroImage* 85.3 (2014), 948–60.

[101] D. LACEWELL, B. BURLEY, S. BOULOS, and P. SHIRLEY. **Raytracing Prefiltered Occlusion for Aggregate Geometry**. *IEEE Symposium on Interactive Raytracing 2008*. 2008, 19–26.

[102] B. LANFER, M. SCHERG, M. DANNHAUER, T. R. KNÖSCHE, and C. H. WOLTERS. **Influences of Skull Segmentation Deficiencies on EEG Source Analysis**. *NeuroImage* 62.1 (2012), 418–431.

[103] M. LANGER and H. BÜLTHOFF. **Depth Discrimination from Shading under Diffuse Lighting**. *Perception* 29 (2000), 649–660.

[104] R. S. LARAMEE, B. JOBARD, and H. HAUSER. **Image Space Based Visualization of Unsteady Flow on Surfaces**. *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*. Washington, DC, USA: IEEE Computer Society, 2003, 18. ISBN: 0-7695-2030-8.

[105] R. S. LARAMEE, J. J. WIJK, B. JOBARD, and H. HAUSER. **ISA and IBFVS: Image Space-Based Visualization of Flow on Surfaces**. *IEEE Transactions on Visualization and Computer Graphics* 10 (2004), 637–648. ISSN: 1077-2626.

[106] E. LENGYEL. **Mathematics for 3D Game Programming and Computer Graphics**. Cengage Learning PTR, 2011. ISBN: 978-1-4354-5886-4.

[107] A. LEW, D. SLIVA, M.-S. CHOE, P. GRANT, Y. OKADA, C. WOLTERS, and M. HÄMÄLÄINEN. **Effects of sutures and fontanels on MEG and EEG source analysis in a realistic infant head model**. *NeuroImage* 76 (2013), 282–293.

[108] K. LI. **Neuroanatomical Segmentation in MRI Exploiting A Priori Knowledge**. PhD thesis. Department of Computer, Information Science, and the Graduate School of the University of Oregon, 2007.

[109] Y. LI and P. WEN. **Tissue Conductivity Anisotropy Inhomogeneity Study in EEG Head Modelling**. *Bioinformatics & Computational Biology*. 2008, 862–867.

[110] F.-H. LIN, J. W. BELLIVEAU, A. M. DALE, and M. S. HÄMÄLÄINEN. **Distributed Current Estimates Using Cortical Orientation Constraints**. *Human Brain Mapping* 27.1 (2006), 1–13.

[111] Y. LIVNAT, H.-W. SHEN, and C. R. JOHNSON. **A Near Optimal Isosurface Extraction Algorithm Using the Span Space**. *IEEE Transactions on Visualization and Computer Graphics* 2.1 (1996), 73–84. ISSN: 1077-2626.

[112] G. LOHMANN, K. MÜLLER, V. BOSCH, H. MENTZEL, S. HESSLER, L. CHEN, S. ZYSSET, and D. Y. CRAMON. **LIPSIA–a new software system for the evaluation of functional magnetic resonance images of the human brain.** eng. *Computerized Medical Imaging and Graphics* 25.6 (2001), 449–457.

[113] W. E. LORENSEN and H. E. CLINE. **Marching cubes: A high resolution 3D surface construction algorithm**. *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1987, 163–169. ISBN: 0-89791-227-6.

[114] A. LOZANO and M. HALLETT. **Physics of effects of transcranial brain stimulation**. *Brain Stimulation E-Book: Handbook of Clinical Neurology (Series editors: Aminoff, Boller, Swaab)* 116 (2013), 353.

[115] T. LUFT, C. COLDITZ, and O. DEUSSEN. **Image Enhancement By Unsharp Masking The Depth Buffer**. *ACM Transactions on Graphics* 25.3 (2006), 1206–1213.

[116] R. MACIEJEWSKI, I. WOO, W. CHEN, and D. EBERT. **Structuring Feature Space: A Non-Parametric Method for Volumetric Transfer Function Generation**. *IEEE Transactions on Visualization and Computer Graphics* 15.6 (2009), 1473–1480. ISSN: 1077-2626.

[117] O. MALLO, R. PEIKERT, C. SIGG, and F. SADLO. **Illuminated Lines Revisited**. *IEEE Visualization*. 2005, 19–26.

[118] M. MALMER, F. MALMER, U. ASSARSSON, and N. HOLZSCHUCH. **Fast Precomputed Ambient Occlusion for Proximity Shadows**. *Journal of Graphics Tools* 12.2 (2007), 59–71.

[119] S. MARCHESIN, C.-K. CHEN, C. HO, and K.-L. MA. **View-Dependent Streamlines for 3D Vector Fields**. *IEEE Transactions on Visualization and Computer Graphics* 16.6 (2010), 1578–1586. ISSN: 1077-2626.

[120] G. MARIN, C. GUERIN, S. BAILLET, L. GARNERO, and G. MEUNIER. **Influence of skull anisotropy for the forward and inverse problem in EEG: simulation studies using FEM on realistic head models.** eng. *Human Brain Mapping* 6.4 (1998), 250–269.

[121] N. MAX. **Optical Models for Direct Volume Rendering**. *IEEE Transactions on Visualization and Computer Graphics* 1.2 (1995), 99–108. ISSN: 1077-2626.

[122] **MayaVi Data Visualizer**.
URL: http://mayavi.sourceforge.net.

[123] M. MCGUIRE. **Ambient Occlusion Volumes**. *Proceedings of High Performance Graphics 2010*. Saarbrucken, Germany: Eurographics Association, 2010, 47–56.

[124] A. R. MCINTOSH and F. GONZALEZ-LIMA. **Structural equation modeling and its application to network analysis in functional brain imaging**. *Human Brain Mapping* 2.1-2 (1994), 2–22.

[125] **MedINRIA: Medical Image Navigation and Research Tool by INRIA**.
URL: http://www-sop.inria.fr/asclepios/software/MedINRIA/.

[126] L. MEIDEIROS, I. DESOUZA, L. VIDOR, A. DESOUZA, A. DEITOS, M. VOLZ, F. FREGNI, W. CAUMO, and I. TORRES. **Neurobiological Effects of Transcranial Direct Current Stimulation: A Review**. *Frontiers in Psychiatry* 3.110 (2012).

[127] Z. MELEK, D. MAYERICH, C. YUKSEL, and J. KEYSER. **Visualization of Fibrous and Thread-like Data**. *IEEE Transactions on Visualization and Computer Graphics* 12 (5 2006), 1165–1172. ISSN: 1077-2626.

[128] D. MERHOF, M. SONNTAG, F. ENDERS, C. NIMSKY, P. HASTREITER, and G. GREINER. **Hybrid Visualization for White Matter Tracts using Triangle Strips and Point Sprites**. *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006), 1181–1188. ISSN: 1077-2626.

[129] **MeVisLab: development environment for medical image processing and visualization**.
URL: http://www.mevislab.de/.

[130] P. MINHAS, M. BIKSON, A. WOODS, A. ROSEN, and S. KESSLER. **Transcranial direct current stimulation in pediatric brain: a computational modeling study**. *Conference Proceedings - IEEE Engineering in Medicine and Biology Society*. 2012.

[131] M. MITTRING. **Finding next gen: CryEngine 2**. *ACM SIGGRAPH 2007 courses*. *SIGGRAPH '07*. San Diego, California: ACM, 2007, 97–121.

[132] V. MONTES-RESTREPO, P. MIERLO, G. STROBBE, S. STAELENS, S. VANDENBERGHE, and H. HALLEZ. **Influence of Skull Modeling Approaches on EEG Source Localization**. *Brain Topography* 27.1 (2014), 95–111.

[133] K. MORELAND. **A Survey of Visualization Pipelines**. *Visualization and Computer Graphics, IEEE Transactions on* 19.3 (2013), 367–378. ISSN: 1077-2626.

[134] S. MORI and P. C. M. ZIJL. **Fiber tracking: principles and strategies - a technical review**. *NMR in Biomedicine* 15.7-8 (2002), 468–480.

[135] K. MUELLER, T. WELSH, W. ZHU, J. MEADE, and N. VOLKOW. **Brainminer: A visualization tool for ROI-based discovery of functional relationships in the human brain.** *New Paradigms in Information Visualization and Manipulation (NPIVM) 2000*. 2000, 481–485.

[136] G. M. NIELSON and B. HAMANN. **The asymptotic decider: resolving the ambiguity in marching cubes**. *VIS '91: Proceedings of the 2nd conference on Visualization '91*. San Diego, California: IEEE Computer Society Press, 1991, 83–91. ISBN: 0-8186-2245-8.

[137] M. NITSCHE, L. COHEN, E. WASSERMANN, A. PRIORI, N. LANG, A. ANTAL, W. PAULUS, F. HUMMEL, P. BOGGIO, and F. FREGNI. **Transcranial direct current stimulation: State of the art 2008**. *Brain Stimulation* 1.3 (2008), 206–223.

[138] M. NITSCHE and W. PAULUS. **Noninvasive brain stimulation protocols in the treatment of epilepsy: current state and perspectives.** *Neurotherapeutics* 6 (2009), 244–50.

[139] M. NITSCHE and W. PAULUS. **Transcranial direct current stimulation–update 2011**. *Restorative Neurology and Neuroscience* 29.6 (2011), 463–92.

[140] M. A. NITSCHE, L. G. COHEN, E. M. WASSERMANN, A. PRIORI, N. LANG, A. ANTAL, W. PAULUS, F. HUMMEL, P. S. BOGGIO, and F. FREGNI. **Transcranial direct current stimulation: State of the art 2008**. *Brain Stimulation* 1.3 (2008), 206–223.

[141] P. NUNEZ. **Electric Fields of the Brain: The Neurophysics of EEG**. New York: Oxford University Press, 1981.

[142] S. OELTZE-JAFRA and B. PREIM. **Survey of Labeling Techniques in Medical Visualizations**. *Eurographics Workshop on Visual Computing for Biology and Medicine, VCBM 2014, Vienna, Austria, 2014. Proceedings*. Ed. by I. Viola, K. Bühler, and T. Ropinski. 2014, 199–208.

[143] T. F. OOSTENDORP, J. DELBEKE, and D. F. STEGEMAN. **The conductivity of the human skull: results of in vivo and in vitro measurements.** *IEEE Transactions on Biomedical Engineering* 47.11 (2000), 1487–1492.

[144] **OpenSceneGraph: an open source high performance 3D graphics toolkit**.
URL: http://www.openscenegraph.org/.

[145] S. PAJEVIC and C. PIERPAOLI. **Color schemes to represent the orientation of anisotropic tissues from diffusion tensor data: application to white matter fiber tract mapping in the human brain**. *Magnetic Resonance in Medicine* 43.6 (2000), 921–921. ISSN: 1522-2594.

[146] J. PARK, S. HONG, D. KIM, M. SUH, and C. IM. **A Novel Array-Type Direct Current Stimulation (tDCS) System for Accurate Focusing on Target Brain Regions**. *IEEE Transactions on Magnetics* 47.5 (2011), 882–5.

[147] W. PAULUS. **Transcranial electrical stimulation (tES - tDCS; tRNS, tACS) methods.** *Neuropsychol Rehabil.* 21.5 (2011), 602–17.

[148] W. PAULUS, A. ANTAL, and M. NITSCHE. **A reference book for Transcranial Brain Stimulation**. Ed. by C. Miniussi, W. Paulus, and M. Rossini. Taylor and Francis Group, 2012. *Physiological basis and methodological aspects of transcranial electric stimulation (tDCS, tACS, and tRNS)*.

[149] T. PEETERS, V. PRCKOVSKA, M. ALMSICK, A. VILANOVA, and B. HAAR ROMENY. **Fast and sleek glyph rendering for interactive HARDI data exploration**. *Visualization Symposium, IEEE Pacific* (2009), 153–160.

[150] W. PENNY, K. STEPHAN, A. MECHELLI, and K. FRISTON. **Modelling functional integration: a comparison of structural equation and dynamic causal models**. *NeuroImage* 23, Supplement 1 (2004). Mathematics in Brain Imaging, S264–S274. ISSN: 1053-8119.

[151] L. PETROVIC, M. HENNE, and J. ANDERSON. **Volumetric Methods for Simulation and Rendering of Hair**. *Pixar Technical Memo 06-08*. 2006.

[152] N. POLYDORIDES and R. LIONHEART. **A Matlab toolkit for three-dimensional electrical impedance tomography: a contribution to the Electrical Impedance and Diffuse Optical Reconstruction Software project**. *Measurement Science and Technology* 13.12 (2002), 1871–83.

[153] B. PREIM and C. P. BOTHA. **Visual Computing for Medicine, Second Edition: Theory, Algorithms, and Applications**. 2nd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013. ISBN: 0124158730, 9780124158733.

[154] B. PURNOMO, J. D. COHEN, and S. KUMAR. **Seamless texture atlases**. *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. Nice, France: ACM, 2004, 65–74. ISBN: 3-905673-13-4.

[155] V. S. RAMACHANDRAN. **Perception of shape from shading**. *Nature* 331 (1988), 163–166.

[156] S. RAMPERSAD, D. STEGEMAN, and T. OOSTENDORP. **Single-Layer Skull Approximations Perform Well in Transcranial Direct Current Stimulation Modeling**. *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 21.3 (2013), 346–353.

[157] G. REINA, K. BIDMON, F. ENDERS, P. HASTREITER, and T. ERTL. **GPU-Based Hyperstreamlines for Diffusion Tensor Imaging**. *Proceedings of EUROGRAPHICS - IEEE VGTC Symposium on Visualization 2006*. 2006, 35–42.

[158] C. REINBOTHE, T. BOUBEKEUR, and M. ALEXA. **Hybrid Ambient Occlusion**. *EUROGRAPHICS 2009 Areas Papers* (2009), 51–57.

[159] Z. REN, R. WANG, J. SNYDER, K. ZHOU, X. LIU, B. SUN, P.-P. SLOAN, H. BAO, Q. PENG, and B. GUO. **Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation**. *ACM SIGGRAPH 2006 Papers*. *SIGGRAPH '06*. Boston, Massachusetts: ACM, 2006, 977–986. ISBN: 1-59593-364-6.

[160] T. ROHLFING, R. BRANDT, R. MENZEL, D. B. RUSSAKOFF, and C. R. MAURER JR. **Handbook of Biomedical Image Analysis: Registration Models**. Ed. by J. S. Suri, D. Wilson, and S. Laxminarayan. Springer–Verlag Berlin Heidelberg, 2005. *Quo Vadis, Atlas-Based Segmentation?*, 435–486. ISBN: 978-0-306-48607-4.

[161] T. ROPINSKI, S. OELTZE, and B. PREIM. **Survey of Glyph-based Visualization Techniques for Spatial Multivariate Medical Data**. *Computers & Graphics* 35.2 (2011), 392–401. ISSN: 0097-8493.

[162] P. ROSENTHAL and L. LINSEN. **Image-space point cloud rendering**. *Proceedings of Computer Graphics International*. 2008, 136–143.

[163] G. RUFFINI, F. WENDLING, I. MERLET, B. MOLAEE-ARDEKANI, A. MEKONNEN, R. SALVADOR, A. SORIA-FRISCH, C. GRAU, S. DUNNE, and P. MIRANDA. **Transcranial current brain stimulation (tCS): models and technologies**. *IEEE Trans Neural Syst Rehabil Eng*. 3 (2013), 333–45.

[164] M. RUIZ, A. BARDERA, I. BOADA, and I. VIOLA. **Automatic Transfer Functions Based on Informational Divergence**. *IEEE Transactions on Visualization and Computer Graphics* 17 (12 2011), 1932–1941. ISSN: 1077-2626.

[165] M. RUIZ, I. BOADA, I. VIOLA, S. BRUCKNER, M. FEIXAS, and M. SBERT. **Obscurance-based Volume Rendering Framework**. *Proceedings of Volume Graphics 2008*. 2008, 113–120.

[166] M. RULLMANN, A. ANWANDER, M. DANNHAUER, S. WARFIELD, F. DUFFY, and C. WOLTERS. **EEG source analysis of epileptiform activity using a 1 mm anisotropic hexahedra finite element head model**. *NeuroImage* 44.2 (2009), 399–410. ISSN: 1053-8119.

[167] S. RUSH and D. DRISCOLL. **Current distribution in brain from surface electrodes**. *Anesth Analg Curr Res* 47 (1968), 717–23.

[168] S. RUSINKIEWICZ and M. LEVOY. **QSplat: A Multiresolution Point Rendering System for Large Meshes**. *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. *SIGGRAPH '00*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, 343–352. ISBN: 1-58113-208-5.

[169] R. SADLEIR and A. ARGIBAY. **Modeling Skull Electrical Properties**. *Annals of Biomedical Engineering* 35.10 (2007), 1.

[170] R. SADLEIR, T. VANNORSDALL, D. SCHRETLEN, and B. GORDON. **Target optimization in transcranial direct current stimulation**. *Frontiers in Psychiatry* 3.90 (2012).

[171] R. SALVADOR, A. MEKONNEN, G. RUFFINI, and P. C. MIRANDA. **Modeling the electric field induced in a high resolution realistic head model during transcranial current stimulation**. *32nd Annual International Conference of the IEEE EMBS*. 2010.

[172] P. H. SCHIMPF, C. RAMON, and J. HAUEISEN. **Dipole models for the EEG and MEG.** eng. *IEEE Transactions on Biomedical Engineering* 49.5 (2002), 409–418.

[173] M. SCHIRSKI, T. KUHLEN, M. HOPP, P. ADOMEIT, S. PISCHINGER, and C. BISCHOF. **Efficient visualization of large amounts of particle trajectories in virtual environments using virtual tubelets**. *Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry*. *VRCAI '04*. Singapore: ACM, 2004, 141–147. ISBN: 1-58113-884-9.

[174] M. SCHIRSKI, T. KUHLEN, M. HOPP, P. ADOMEIT, S. PISCHINGER, and C. BISCHOF. **Virtual Tubelets-efficiently visualizing large amounts of particle trajectories**. *Comput. Graph.* 29.1 (2005), 17–27. ISSN: 0097-8493.

[175] A. G. P. SCHJETNAN, J. FARAJI, G. A. METZ, M. TATSUNO, and A. LUCZAK. **Transcranial Direct Current Stimulation in Stroke Rehabilitation: A Review of Recent Advancements**. *Stroke Research and Treatment* 2013.170256 (2013), 31–38.

[176] M. SCHOTT, T. MARTIN, A. GROSSET, C. BROWNLEE, T. HOLLT, B. BROWN, S. SMITH, and C. HANSEN. **Combined surface and volumetric occlusion shading**. *Pacific Visualization Symposium (PacificVis), 2012 IEEE*. 2012, 169–176.

[177] M. SCHOTT, V. PEGORARO, C. D. HANSEN, K. BOULANGER, and K. BOUATOUCH. **A Directional Occlusion Shading Model for Interactive Direct Volume Rendering**. *Comput. Graph. Forum* 28.3 (2009), 855–862.

[178] T. SCHULTZ and G. L. KINDLMANN. **A Maximum Enhancing Higher-Order Tensor Glyph**. *Comput. Graph. Forum* 29.3 (2010), 1143–1152.

[179] T. SCHULTZ and H.-P. SEIDEL. **Estimating Crossing Fibers: A Tensor Decomposition Approach**. *IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008), 1635–1642. ISSN: 1077-2626.

[180] G. SCHUSSMAN and K.-L. MA. **Anisotropic Volume Rendering for Extremely Dense, Thin Line Data**. *Proceedings of the conference on Visualization '04. VIS '04*. Washington, DC, USA: IEEE Computer Society, 2004, 107–114. ISBN: 0-7803-8788-0.

[181] **SCIRun: A Scientific Computing Problem Solving Environment, Scientific Computing and Imaging Institute (SCI)**.
URL: http://www.scirun.org.

[182] S. SHAHID, P. WEN, and T. AHFOCK. **Numerical investigation of white matter anisotropic conductivity in defining current distribution under tDCS**. *Computer Methods and Programs in Biomedicine* 109.1 (2013), 48–64.

[183] P. SHANMUGAM and O. ARIKAN. **Hardware accelerated ambient occlusion techniques on GPUs**. *Proceedings of the 2007 symposium on Interactive 3D graphics and games. I3D '07*. Seattle, Washington: ACM, 2007, 73–80. ISBN: 978-1-59593-628-8.

[184] F. SHARBROUGH, G.-E. CHATRIAN, R. LESSER, H. LÜDERS, M. NUWER, and T. PICTON. **American Electroencephalographic Society Guidelines for Standard Electrode Position Nomenclature**. *Journal of clinical Neurophysiology* 2.8 (1991), 200–202.

[185] D. SHREINER, G. SELLERS, J. M. KESSENICH, and B. M. LICEA-KANE. **OpenGL Programming Guide**. Addison-Wesley Professional, 2013. ISBN: 978-0-321-77303-6.

[186] P.-P. SLOAN. **Normal mapping for precomputed radiance transfer**. *Proceedings of the 2006 symposium on Interactive 3D graphics and games*. *I3D '06*. Redwood City, California: ACM, 2006, 23–26. ISBN: 1-59593-295-X.

[187] P.-P. SLOAN, J. KAUTZ, and J. SNYDER. **Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments**. *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. *SIGGRAPH '02*. San Antonio, Texas: ACM, 2002, 527–536. ISBN: 1-58113-521-1.

[188] S. M. SMITH, M. JENKINSON, M. W. WOOLRICH, C. F. BECKMANN, T. E. J. BEHRENS, H. JOHANSEN-BERG, P. R. BANNISTER, M. D. LUCA, I. DROBNJAK, D. E. FLITNEY, R. K. NIAZY, J. SAUNDERS, J. VICKERS, Y. ZHANG, N. D. STEFANO, J. M. BRADY, and P. M. MATTHEWS. **Advances in functional and structural MR image analysis and implementation as FSL.** eng. *Neuroimage* 23 Suppl 1 (2004), S208–S219.

[189] V. ŠOLTÉSZOVÁ, D. PATEL, S. BRUCKNER, and I. VIOLA. **A Multidirectional Occlusion Shading Model for Direct Volume Rendering**. *Computer Graphics Forum* 29.3 (2010), 883–891.

[190] E. SOMERSALO, M. CHENEY, and D. ISAACSON. **Existence and Uniqueness for Electrode Models for Electric Current Computed Tomography**. *SIAM Journal on Applied Mathematics* 52 (1992), 1012–40.

[191] Y. SONG, E. LEE, E. J. WOO, and J. SEO. **Optimal geometry toward uniform current density electrodes**. *Inverse Problems* 27.7 (2011).

[192] C. STAAG and M. NITSCHE. **Physiological Basis of Transcranial Direct Current Stimulation**. *Neuroscientist* 17 (2011), 37–53.

[193] D. STALLING and H.-C. HEGE. **Fast and Resolution Independent Line Integral Convolution**. *SIGGRAPH95*. Ed. by R. Cook. *CGPACS*. New York, 1995, 249–256.

[194] K. E. STEPHAN, J. C. MARSHALL, W. D. PENNY, K. J. FRISTON, and G. R. FINK. **Interhemispheric Integration of Visual Processing during Task-Driven Lateralization**. *J. Neurosci.* 27.13 (2007), 3512–3522.

[195] K. E. STEPHAN, M. TITTGEMEYER, T. R. KNÖSCHE, R. J. MORAN, and K. J. FRISTON. **Tractography-based priors for dynamic causal models**. *NeuroImage* 47.4 (2009), 1628–1638. ISSN: 1053-8119.

[196] K. E. STEPHAN, N. WEISKOPF, P. M. DRYSDALE, P. A. ROBINSON, and K. J. FRISTON. **Comparing hemodynamic models with DCM**. *NeuroImage* 38.3 (2007), 387–401. ISSN: 1053-8119.

[197] A. J. STEWART. **Vicinity Shading for Enhanced Perception of Volumetric Data**. *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*. Washington, DC, USA: IEEE Computer Society, 2003, 47. ISBN: 0-7695-2030-8.

[198] C. STOLL, S. GUMHOLD, and H.-P. SEIDEL. **Visualization with Stylized Line Primitives**. *IEEE Visualization 2005 (VIS 2005)*. Ed. by C. T. Silva, E. Gröller, and H. Rushmeier. Minneapolis, USA: IEEE, 2005, 695–702.

[199] H. SUH, W. LEE, and T.-S. KIM. **Influence of anisotropic conductivity in the skull and white matter on transcranial direct current stimulation via an anatomically realistic finite element head model**. *Physics in Medicine and Biology* 57 (2012), 6961–6980.

[200] T. WEINKAUF and H. THEISEL. **Curvature Measures of 3D Vector Fields and their Applications**. *Journal of WSCG* 10.2 (2002). Ed. by V. Skala. WSCG 2002, Plzen, Czech Republic, February 4 - 8, 507–514.

[201] M. TARINI, P. CIGNONI, and C. MONTANI. **Ambient Occlusion and Edge Cueing for Enhancing Real Time Molecular Visualization**. *IEEE Transactions on Visualization and Computer Graphics* 12 (5 2006), 1237–1244. ISSN: 1077-2626.

[202] X. TRICOCHE. **Vector and Tensor Field Topology Simplification, Tracking, and Visualization**. PhD thesis. Germany: University of Kaiserslautern, 2002.

[203] X. TRICOCHE, R. MACLEOD, and C. R. JOHNSON. **Visual Analysis of Bioelectric Fields**. Ed. by L. Linsen, H. Hagen, and B. Hamann. Springer Berlin Heidelberg, 2008, 205–220.

[204] X. TRICOCHE, G. SCHEUERMANN, and H. HAGEN. **Tensor Topology Tracking: A Visualization Method for Time-Dependent 2D Symmetric Tensor Fields**. *Eurographics 2001 Proceedings, Computer Graphics Forum 20(3)*. The Eurographics Association. Saarbrücken, Germany, 2001, 461–470.

[205] A. TURING. **The chemical basis of morphogenesis**. *Philosophical Transactions of the Royal Society of London* 237.641 (1952), 37–72.

[206] G. TURK. **Generating textures on arbitrary surfaces using reaction-diffusion**. *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1991, 289–298. ISBN: 0-89791-436-8.

[207] K. UTZ, V. DIMOVA, K. OPPENLÄNDER, and G. KERKHOFF. **Electrified minds: Transcranial direct current stimulation (tDCS) and Galvanic Vestibular Stimulation (GVS) as methods of non-invasive brain stimulation in neuropsychology – A review of current data and future implications**. *Neuropsychologia* 48.10 (2010), 2789–2810.

[208] P. VISION PTY. LTD. **POV-Ray Raytracer**.
URL: http://www.povray.org.

[209] **VTK Visualization Toolkit**.
URL: http://www.vtk.org/.

[210] S. WAGNER, S. RAMPERSAD, Ü. AYDIN, J. VORWERK, T. OOSTENDORP, T. NEULING, C. HERRMANN, D. STEGEMAN, and C. WOLTERS. **Investigation of tDCS volume conduction effects in a highly realistic head model**. *Journal of Neural Engineering* 11.1 (2014), 016002.

[211] I. WALD, H. FRIEDRICH, G. MARMITT, P. SLUSALLEK, and H.-P. SEIDEL. **Faster Isosurface Ray Tracing Using Implicit KD-Trees**. *IEEE Transactions on Visualization and Computer Graphics* 11 (5 2005), 562–572. ISSN: 1077-2626.

[212] L. WANG, Y. ZHAO, K. MUELLER, and A. KAUFMAN. **The magic volume lens: An interactive focus+context technique for volume rendering**. *In Proc. of IEEE Visualization '05 (2005)*. Los Alamitos, CA, USA: IEEE Computer Society, 2005, 367–374. ISBN: 0-7803-9462-3.

[213] R. WANG, K. ZHOU, J. SNYDER, X. LIU, H. BAO, Q. PENG, and B. GUO. **Variational sphere set approximation for solid objects**. *Vis. Comput.* 22 (9 2006), 612–621. ISSN: 0178-2789.

[214] L. WANGER. **The effect of shadow quality on the perception of spatial relationships in computer generated imagery**. *Proceedings of the 1992 symposium on Interactive 3D graphics*. *I3D '92*. Cambridge, Massachusetts, United States: ACM, 1992, 39–42. ISBN: 0-89791-467-8.

[215] L. C. WANGER, J. A. FERWERDA, and D. P. GREENBERG. **Perceiving Spatial Relationships in Computer-Generated Images**. *IEEE Computer Graphics and Applications* 12 (3 1992), 44–51, 54–58. ISSN: 0272-1716.

[216] K. WARD, F. BERTAILS, T.-Y. KIM, S. R. MARSCHNER, M.-P. CANI, and M. C. LIN. **A Survey on Hair Modeling: Styling, Simulation, and Rendering**. *IEEE Transactions on Visualization and Computer Graphics* 13 (2 2007), 213–234. ISSN: 1077-2626.

[217] D. WEINSTEIN, G. KINDLMANN, and E. LUNDBERG. **Tensorlines: advection-diffusion based propagation through diffusion tensor fields**. *VIS '99: Proceedings of the conference on Visualization '99*. San Francisco, California, United States: IEEE Computer Society Press, 1999, 249–253.

[218] D. WEISKOPF and T. ERTL. **A hybrid physical/device-space approach for spatio-temporally coherent interactive texture advection on curved surfaces**. *GI '04: Proceedings of Graphics Interface 2004*. London, Ontario, Canada: Canadian Human-Computer Communications Society, 2004, 263–270. ISBN: 1-56881-227-2.

[219] T. WELSH, K. MUELLER, W. ZHU, N. VOLKOW, and J. MEADE. **Graphical strategies to convey functional relationships in the human brain: a case study**. *VIS '01: Proceedings of the conference on Visualization '01*. San Diego, California: IEEE Computer Society, 2001, 481–484. ISBN: 0-7803-7200-X.

[220] K. WENDEL, O. VÄISÄNEN, J. MALMIVUO, N. G. GENCER, B. VANRUMSTE, P. DURKA, R. MAGJAREVIC, S. SUPEK, M. L. PASCU, H. FONTENELLE, and R. G. PERALTA MENENDEZ. **EEG/MEG source imaging: methods, challenges, and open issues**. *Computational Intelligence and Neuroscience* 2009 (2009), 13:1–13:12. ISSN: 1687-5265.

[221] J. J. WIJK. **Image based flow visualization**. *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. San Antonio, Texas: ACM, 2002, 745–754. ISBN: 1-58113-521-1.

[222] J. J. WIJK. **Image Based Flow Visualization for Curved Surfaces**. *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*. Washington, DC, USA: IEEE Computer Society, 2003, 17. ISBN: 0-7695-2030-8.

[223] J. WILHELMS and A. VAN GELDER. **Octrees for faster isosurface generation**. *ACM Trans. Graph.* 11.3 (1992), 201–227. ISSN: 0730-0301.

[224] M. WIRTH, R. RAHMAN, J. KUENECKE, T. KOENIG, H. HORN, W. SOMMER, and T. DIERKS. **Effects of transcranial direct current stimulation (tDCS) on behavior and electrophysiology of language production**. *Neuropsychologia* 49 (2011), 3989–98.

[225] C. WOLTERS. **Influence of Tissue Conductivity Inhomogeneity and Anisotropy on EEG/MEG based Source Localization in the Human Brain**. PhD thesis. University of Leipzig, 2003. ISBN: 3-936816-11-5.

[226] C. WOLTERS, A. ANWANDER, X. TRICOCHE, D. WEINSTEIN, M. KOCH, and R. MACLEOD. **Influence of tissue conductivity anisotropy on EEG/MEG field and return current computation in a realistic head model: A simulation and visualization study using high-resolution finite element modeling**. *NeuroImage* 30.3 (2006), 813–826. ISSN: 1053-8119.

[227] X. WU. **An efficient antialiasing technique**. *SIGGRAPH Comput. Graph.* 25.4 (1991), 143–152. ISSN: 0097-8930.

[228] C. WYMAN, S. PARKER, P. SHIRLEY, and C. HANSEN. **Interactive Display of Isosurfaces with Global Illumination**. *IEEE Transactions on Visualization and Computer Graphics* 12 (2 2006), 186–196. ISSN: 1077-2626.

[229] C. YUKSEL and E. AKLEMAN. **Rendering hair with global illumination**. *ACM SIGGRAPH 2006 Research posters*. *SIGGRAPH '06*. Boston, Massachusetts: ACM, 2006, 124. ISBN: 1-59593-364-6.

[230] C. YUKSEL, E. AKLEMAN, and J. KEYSER. **Practical Global Illumination for Hair Rendering**. *Proceedings of Pacific Graphics 2007*. Maui, Hawaii, 2007.

[231] C. YUKSEL and S. TARIQ. **Advanced techniques in real-time hair rendering and simulation**. *ACM SIGGRAPH 2010 Courses*. *SIGGRAPH '10*. Los Angeles, California: ACM, 2010, 1–168.

[232] E. ZHANG, J. HAYS, and G. TURK. **Interactive Tensor Field Design and Visualization on Surfaces**. *IEEE Transactions on Visualization and Computer Graphics* 13.1 (2007), 94–107. ISSN: 1077-2626.

[233] E. ZHANG, H. YEH, Z. LIN, and R. S. LARAMEE. **Asymmetric Tensor Analysis for Flow Visualization**. *IEEE Transactions on Visualization and Computer Graphics* 15 (2009), 106–122. ISSN: 1077-2626.

[234] S. ZHANG, C. DEMIRALP, and D. H. LAIDLAW. **Visualizing Diffusion Tensor MR Images Using Streamtubes and Streamsurfaces**. *IEEE Transactions on Visualization and Computer Graphics* 9.4 (2003), 454–462. ISSN: 1077-2626.

[235] X. ZHENG and A. PANG. **HyperLIC**. *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*. Washington, DC, USA: IEEE Computer Society, 2003, 33. ISBN: 0-7695-2030-8.

[236] J. ZHOU and M. TAKATSUKA. **Automatic Transfer Function Generation Using Contour Tree Controlled Residue Flow Model and Color Harmonics**. *IEEE Transactions on Visualization and Computer Graphics* 15.6 (2009), 1481–1488. ISSN: 1077-2626.

[237] S. ZHUKOV, A. INOES, and G. KRONIN. **An Ambient Light Illumination Model**. *Rendering Techniques '98*. Ed. by G. Drettakis and N. Max. *Eurographics*. Springer-Verlag Wien New York, 1998, 45–56.

[238] M. ZÖCKLER, D. STALLING, and H.-C. HEGE. **Interactive visualization of 3D-vector fields using illuminated stream lines**. *Proceedings of the 7th conference on Visualization '96*. *VIS '96*. San Francisco, California, United States: IEEE Computer Society Press, 1996, 107–ff. ISBN: 0-89791-864-9.

[239] M. ZWAN, W. LUEKS, H. BEKKER, and T. ISENBERG. **Illustrative Molecular Visualization with Continuous Abstraction**. *Computer Graphics Forum* 30.3 (2011), 683–690. ISSN: 1467-8659.