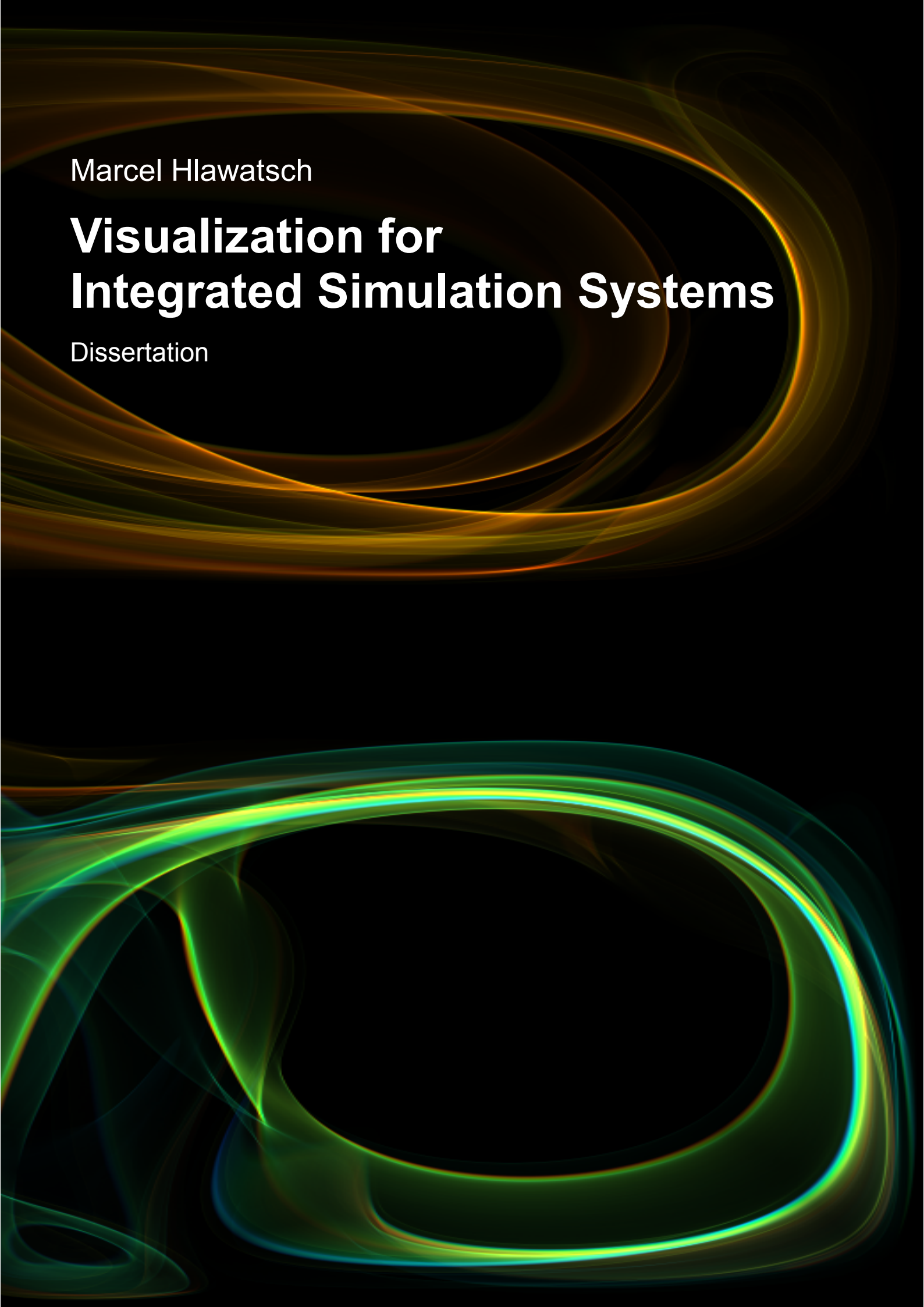
Marcel Hlawatsch

Visualization for Integrated Simulation Systems

Dissertation



Visualization for Integrated Simulation Systems

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik
und dem Stuttgart Research Centre for Simulation Technology der
Universität Stuttgart
zur Erlangung der Würde eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Abhandlung

Vorgelegt von

Marcel Hlawatsch

aus Leonberg

Hauptberichter:	Prof. Dr. Daniel Weiskopf
Mitberichter:	Prof. Dr. Rüdiger Westermann
Tag der mündlichen Prüfung:	16.11.2015

Visualisierungsinstitut
der Universität Stuttgart

2015

Acknowledgments

First of all, I am very grateful that Daniel Weiskopf was my supervisor and that he gave me the opportunity to be his PhD student. He provided me the best support and supervision I can imagine. Thank you, Daniel, for all your time, wisdom, patience, and support! Without you, I would not have been a PhD student and performed all the interesting work I present in this dissertation.

I would like to thank Rüdiger Westermann for being reviewer for this dissertation and taking part in my PhD defense. I am very glad that Joachim Vollrath supervised my student and Diplom thesis. This laid the foundation for my PhD and without this time, I don't think I would have started my PhD at VISUS. I would like to thank Claudio Silva and Juliana Freire for hosting my visit at NYU Poly in New York and working with me on the provenance visualization project. During my visit in New York, I had a great time with Claudio's group, especially the following people welcomed me warmly: Jonathas Costa, Lauro Lins, Fernando Chirigati, and Lis Ingrid Roque. I also want to thank Thomas Ertl and Daniel Weiskopf for making the fantastic working environment at VIS/VISUS possible.

It was great to have joint work and projects with the following people: Joachim Vollrath brought me to tensor field visualization; I worked with Wolfgang Nowak, Philipp Leube, and Sergey Oladyshkin on uncertainty visualization; Fabian Beck helped me with graphs and provenance; Kuno Kurzhals realized my eye-tracking ideas. Special thanks go to Filip Sadlo for introducing me to the mysteries of the FTLE and our many flowvis projects, and to Michael Burch for inspiring me with radial diagrams, all our infovis work, and giving me the chance to visit SIGGRAPH. And of course, this list is not complete without Daniel Weiskopf, who helped me with all my projects, even the crazy ones. I would also like to thank the people with whom I shared the office during my time as a PhD student, especially for tolerating my outbursts of fury when software or other things did not work the way I wanted: Filip Sadlo, Grzegorz Karch, Rudolf Netzel, Hansjörg Schmauder.

Special thanks go to Steffen Frey, Corinna Vehlow, Grzegorz Karch, Katrin Scharnowski, and Rudolf Netzel for proof reading this thesis and their valuable comments. Especially Steffen had a large bunch of pages to read, thanks for that. Furthermore, I thank Martin Falk for creating and providing the Latex template for the thesis.

What makes the time so special at VIS/VISUS—and is one of the reasons why I am still working here—are the nice people and the different activities and the fun we have together. Therefore, I want to thank the following persons for being nice colleagues and their support during my work: Joachim Vollrath, Julian Heinrich, (Tante) Katrin Bidmon, Fabian Beck, Michael Burch, Rudolf Netzel, Markus Huber, Corinna Vehlow, Marco Ament, Michael Raschke, Tanja Blascheck, Sebastian Boblest, Michael Bußler, Markus John, Florian Heimerl, Qi Han, Florian Haag, Robert Krüger, David Körner, Finian Mwalongo, Steffen Lohmann, Anette Müller, Harald Sanftmann, Christiane and Enrico Taras, Thomas Engelhardt, Gabor Liktor, Jan Novak, Christian Pagot, Benjamin and Markus Höferlin, Daniel Kauker, Sebastian Grottel, Friedemann Rößler, Frank Grave, Thomas Müller, Wolfgang Bayerlein, Marianne Jungjohann, Tina Barthelmes, Maria Schulz, Karin Vrana, Martin Schmid, Ulrike Ritzmann, Margot Roubicek, Christine Schütz, Iryna Rybak, Sergey Oladyshkin. Not only for being nice colleagues, but also for watching movies, having barbecue, playing (video) games, going out for some drinks, or having fun in any other incarnation, special thanks go to the following people: Markus Üffinger, Steffen Frey, Grzegorz Karch, Alexandros Panagiotidis, Kuno Kurzhals, Filip Sadlo, Martin Falk, Sven Bachthaler, Lena Gieseke, Oliver Fernandes, Katrin Scharnowski, Stefan Heßel, Harald Bosch, Steffen Koch, Sebastian Koch, Alexander Wender, Bernhard Schmitz, Dennis Thom, Michael Wörner, Dominik Herr, Guido Reina, Christoph Müller, Michael Krone, Hansjörg Schmauder.

Finally, I want to express my deepest gratitude to my family, especially my parents, who supported me by letting me go my way and not putting any pressure on me, and to all my friends, who keep my life in balance and give me back the energy I need for my work: Moni, Beccy, Metze, Schubi and Isa, Kunze, Gunsen, Diepie, Kalle, Sabrina, Steffen and Miri, And, Dr. Ö, Olly, Grzegorz, Kerstin, Diana, Juli, Regine, Siggi, Klaus, Peter and Sabrina.

My work at the Visualization Research Center of the University of Stuttgart was supported by the German Research Foundation DFG within the Cluster of Excellence in Simulation Technology (EXC 310) and SFB/Transregio 161.

Marcel Hlawatsch

Contents

List of Figures

Part III

Chapter 6

Chapter 7

List of Tables

Chapter 2

Chapter 4

List of Abbreviations and Units

Abbreviations

API	application programming interface
CFD	computational fluid dynamics
CPU	central processing unit
CUDA	Compute Unified Device Architecture
DT-MRI	diffusion tensor magnetic resonance imaging
FA	fractional anisotropy
FTLE	finite-time Lyapunov exponent
FSR	finite separation ratio
GLSL	OpenGL Shading Language
GPU	graphics processing unit
ISCO	in-situ chemical oxidation
LCS	Lagrangian coherent structures
LE	Lyapunov exponent
LIC	line integral convolution
ODE	ordinary differential equation
OpenGL	Open Graphics Library
RFB	remote framebuffer protocol
RMSE	root-mean-square error
SCR	separatrices of coherent regions
VBO	vertex buffer object (OpenGL)

Units

fps	frames per second
GB	gibibyte, 2^{30} byte
GHz	gigahertz, 10^{9} hertz
MB	mebibyte, 2^{20} byte
ms	millisecond, 10^{-3} seconds

Abstract

Today, a large part of science as well as many applications in industry require the usage of simulation technology. Several key elements are relevant for an expedient use of modern simulation technology. It is important that the methods for analyzing the simulation outcome are suitable for the increasing complexity of simulations and the generated data. Larger or more complex data is analyzed visually in many cases. Therefore, visualization is essential for the work with simulation technology. Furthermore, an integrated simulation system providing simulation components as well as visualization components allows a more efficient work with simulation technology. Finally, uncertainties can be introduced at different stages in the simulation and analysis process. Correct interpretation of simulation results must consider these uncertainties. The theme of this thesis is to improve the work with simulation technology by employing novel visualization approaches and techniques. Each of these approaches targets one or more of these key elements.

Since the key role of visualization in this context lies in displaying the simulation results, novel methods for visualizing different classes of data are introduced. It is discussed how scalar values covering a large value range can be visualized. Furthermore, several methods for time-dependent vector fields are presented. Another approach deals with the visualization of coherent structures in symmetric second-order tensor fields

Effective analysis of simulation results is important, but not the only aspect that can be supported with visualization. Considering integrated simulation systems, such a complex software environment is typically built in a modular way. Workflows are a common way to control modular software and define the connections between the respective modules and their execution. This thesis presents two methods for visualizing the evolution of such workflows. They can be used to recapitulate previous sessions or analyze user behavior. This can help continue previous work or improve the respective software.

Finally, three different types of uncertainty are discussed in this thesis and methods for handling them are presented: Uncertainty in the simulation model can be handled with a computational steering approach that allows the interactive change of simulation parameters. A glyph-based visualization method is introduced for time-dependent vector fields with uncertainty. Finally, the handling of uncertainty in user interaction is demonstrated in the context of flow visualization.

German Abstract
—Zusammenfassung—

Heutzutage erfordern sowohl ein großer Teil der Wissenschaft als auch viele industrielle Anwendungen den Einsatz von Simulationstechnologie. Mehrere Schlüsselelemente sind bei einem sinnvollen Einsatz moderner Simulationstechnologie von Bedeutung. Ein wichtiger Aspekt ist, dass die Methoden für die Analyse der Simulationsergebnisse mit der steigenden Komplexität der Simulationen und erzeugten Daten mithalten können. Größere und komplexere Daten werden dabei häufig visuell analysiert. Daher ist die Visualisierung essentiell für den Einsatz von Simulationstechnologie. Weiterhin ermöglicht ein integriertes Simulationssystem, das sowohl Simulations- als auch Visualisierungskomponenten bereithält, ein effizienteres Arbeiten mit Simulationstechnologie. Zu guter Letzt können Unsicherheiten in den verschiedenen Stufen des Simulations- und Analysevorgangs auftreten. Eine korrekte Interpretation der Simulationsergebnisse muss solche Unsicherheiten berücksichtigen. Das Leitmotiv dieser Dissertation ist es, die Arbeit mit Simulationstechnologie mit Hilfe von neuen Visualisierungsansätzen und -techniken zu verbessern. Jeder der hier vorgestellten Ansätze zielt dabei auf ein oder mehrere dieser Schlüsselelemente ab.

Da die Schlüsselrolle der Visualisierung im Zusammenhang mit Simulationen die Darstellung der Simulationsergebnisse ist, werden neue Visualisierungsmethoden für verschiedene Arten von Daten vorgestellt. Es wird aufgezeigt, wie Skalardaten dargestellt werden können, die einen großen Wertebereich abdecken. Außerdem werden mehrere Methoden für zeitabhängige Vektorfelder vorgestellt. Ein weiterer Ansatz behandelt die Visualisierung von kohärenten Strukturen in symmetrischen Tensorfeldern zweiter Ordnung.

Eine effektive Analyse der Simulationsergebnisse ist zwar wichtig, aber nicht der einzige Aspekt, der sich mit Visualisierung unterstützen lässt. Komplexe Softwareumgebungen wie integrierte Simulationssysteme werden häufig modular aufgebaut. Workflows sind eine gebräuchliche Technik, um modulare Software zu steuern und die Kopplung der Module und ihre Ausführungsparameter festzulegen. In dieser Dissertation werden zwei Methoden für die Visualisierung der zeitlichen Entwicklung solcher Workflows präsentiert. Die Methoden können dafür verwendet werden, frühere Arbeitsvorgänge nachzuvollziehen oder das Benutzerverhalten zu analysieren. Dies kann bei der Fortsetzung zurückliegender Arbeit oder bei der Verbesserung der entsprechenden Software helfen.

Zu guter Letzt werden in dieser Dissertation drei Arten von Unsicherheiten besprochen und Methoden für deren Handhabung vorgestellt. Unsicherheit im Simulationsmodell kann mit einem Computational-Steering-Ansatz gehandhabt werden, bei dem es möglich ist, die Simulationsparameter interaktiv zu ändern. Für zeitabhängige Vektorfelder, die mit Unsicherheit behaftet sind, wird eine Visualisierungsmethode auf Basis von Glyphen vorgestellt. Schließlich wird die Behandlung von Unsicherheit in der Benutzereingabe am Beispiel von einer Strömungsvisualisierungsmethode dargelegt.

CHAPTER 1

Introduction

Simulation has become more and more important over the last decades in science and industry. Today, simulations are commonly used tools. There are many research problems that cannot be solved without the use of simulations. In these cases, the respective experiments are too expensive or it is even impossible to implement them.

The important role of simulations in today's scientific world led to the installation of the Stuttgart Research Centre for Simulation Technology and the respective Cluster of Excellence in Simulation Technology (EXC 310)—*SimTech* [257]. The goal of SimTech is to bundle the expertise of the University of Stuttgart in different disciplines for research in the context of simulation technology.

This thesis presents the results of my research in a SimTech project related to visualization in the context of an integrated simulation system. The goal of this project was to improve the work with simulations and this system with the help of newly developed visualization techniques.

1.1 Simulation and Visualization

In general, visualization is required when "looking at numbers" is not sufficient to get insight into the data. Reasons for this can be the size or the complexity of the data. For example, important information might exist in the form of complex patterns or might be hidden by large amounts of nonrelevant data or noise. Simulations are one source for such data. For instance, many simulations

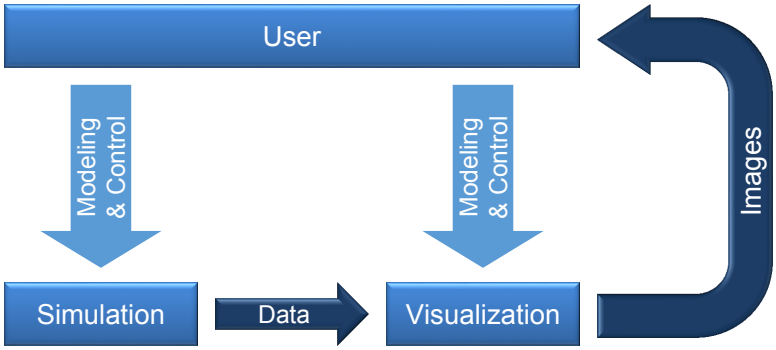


Figure 1.1 — Simulation and analysis process.

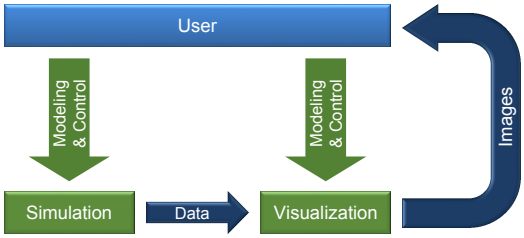
have a spatial context. They model certain processes and phenomena in a 2D or 3D domain. In these cases, the spatial context is typically important for the interpretation of the simulation results. Hence, visualization is required to provide the spatial context.

Therefore, as the basic model in Figure 1.1 shows, visualization is essential for the work with simulations because it is required for the analysis of the generated results. However, displaying simulation results is not the only part of the work with simulations that can benefit from visualization. It will be shown in this thesis that visualization can also support other stages and aspects of the simulation and analysis process.

1.2 Research Questions

The basic question for the research presented here was how visualization can improve the work with simulations. More specifically, the focus was on integrated simulation systems that enable the setup of complex simulations. An integrated simulation system is defined here as a simulation environment based on a modular software framework, i.e., it can be extended by adding new software modules. Furthermore, the setup of a simulation requires the combination of different modules that are then executed in a defined order. Such a simulation system might even offer modules for the visualization of the generated results. These modules can be directly integrated into the simulation workflow, i.e., the process shown in Figure 1.1 can be completely represented inside this system. Three major targets for visualization were identified in this context, which are discussed in the following.

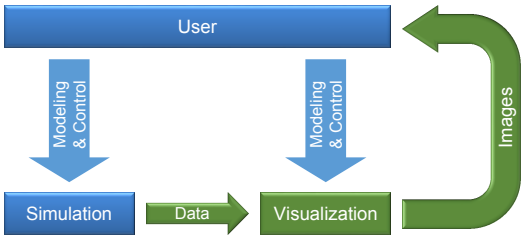
Modeling and Control This aspect deals with the setup of the simulation and the result visualization (marked green). This includes the connection and execution of software modules, and the adaptation of their parameters.



Modular software like an integrated simulation system often employs workflow-based approaches to combine the respective modules and execute them. Tools for creating and manipulating such workflows already provide a visual representation of the workflow, typically as graph visualization. However, this is only an instantaneous view of the current state of the workflow. The question is if the visualization of additional information can support the modeling and control of simulations and respective visualization of results.

Complex simulations typically have a longer life span and are often maintained and extended by multiple users. Components of such simulations may be replaced or new modules may be integrated. The simulation or parts of it might be also integrated into another simulation, e.g., to create a coupled simulation employing multiple physical scales. Visualizing provenance information, which represent the complete creation process of a simulation, could support such tasks. This could help recapitulate previous work with the simulation or reveal modifications other users applied to it. It may also support the user in understanding the overall structure of the simulation.

Data Visualization The main purpose of visualization in the context of simulation is the visual representation of simulation results to enable their analysis, i.e., data is transformed into images and provided to the user (marked green).

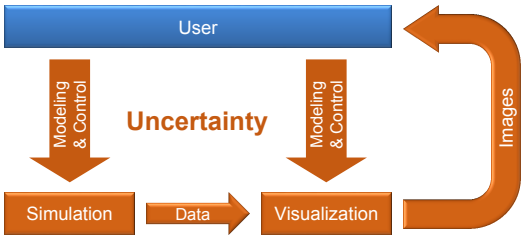


Supported by the tremendous increase of available computational power, more and more complex processes are simulated and the resolution of the models is increased. The generated data also reflects this. It increases in size and contains more complex structures. An adequate visualization of the data must consider these aspects and often requires the development of new methods.

Data resulting from simulations of temporal processes, e.g., time-dependent fluid flow in computational fluid dynamics (CFD) applications, exhibits often complex structures. The additional temporal dimension makes it difficult to analyze and understand the data. An important question is therefore if the visualization of time-dependent processes can be improved, e.g., with novel

static visual representations. Visualizing time-dependent data can also be challenging because the temporal information strongly increases the size of the data. Acceleration techniques could make the visualization on standard computers still feasible in this case. Moreover, not only an additional dimension of the data domain is an issue, but it could also be the case that complex processes can only be represented with complex data elements. For instance, anisotropic diffusion cannot be sufficiently modeled with vectors but requires one to employ higher dimensional tensors. Visualizing such tensors is difficult because they must be mapped to the limited number of visual channels. Topological methods reducing the amount of information could help in this case. Finally, even rather small and simple structured data can be challenging if it covers a large value range. While logarithmic scales are a common approach for handling large value ranges, they might not be the best solution for visual representation.

Uncertainty All stages of the simulation and analysis process including user interaction can introduce uncertainties (marked orange). Handling these uncertainties is important to allow a correct interpretation of simulation results.



With respect to uncertainty in the simulation model, two cases can be distinguished: the simulation ignores the uncertainty or it explicitly handles the uncertainty. In the first case, the result of the simulation is treated as uncertainty free but the uncertainty exists in the form of an inaccurate or incomplete simulation model, e.g., due to unknown input parameters. Fast visualization methods allowing an interactive exploration of the parameter space might provide support in this case. In the second case, the uncertainty is incorporated during the simulation process, e.g., with a Monte Carlo simulation. In this case, the resulting data can include information about the uncertainty. An adequate visual representation of the uncertainty in the data could help the user interpret the uncertainty. Displaying uncertain quantities in combination with the spatial context could ease the interpretation compared to a simple plot.

Finally, the work with simulation and visualization typically is an iterative process. Especially the visual analysis of complex data requires user interaction and is carried out in an iterative loop: parameters are adapted, parts of the data are filtered or highlighted, or enlarged for a detailed analysis, etc. Such interaction could also exhibit uncertainty, e.g., the input device has only a limited resolution, which can make the exploration of the data less efficient. Can this uncertainty be compensated in the visualization and the visual analysis process be made more efficient?

1.3 Outline and Contribution

This section provides an overview of the structure of this thesis and lists the work and publications that are the basis for the different chapters. I am the first author of all these publications and developed the respective software and tools. My supervisor Daniel Weiskopf was involved in all the work described here and is co-author of the respective publications.

The thesis is divided into three different parts that tackle each one or more of the previously discussed research questions. The first part is related to the modeling and control of simulations and the respective result visualization (Part I). Chapter 2 presents two approaches that support the work with modular software and the respective workflows. The first approach is specifically designed for the visualization and analysis of the evolution of such workflows. It is based on work together with Fabian Beck, Michael Burch, Juliana Freire, Claudio Silva, and Daniel Weiskopf [1], who all helped writing and proofreading the paper. Fabian Beck and Michael Burch provided their experiences with software visualization. Juliana Freire and Claudio Silva helped developing the basic concept and provided VisTrails and related datasets. The second approach is more general and was developed to visualize dynamic graphs. However, since this novel graph visualization is especially suitable for sparse dynamic graphs, it can be used to visualize dynamic workflow data. This method results from joined work with Michael Burch and Daniel Weiskopf [2]. Michael Burch wrote parts of the paper and provided the migration data and his experience with dynamic graphs.

Part II is related to novel techniques that can be used to visualize the simulation results. It is subdivided according to different classes of result data. The data dimensionality and complexity increases with each chapter, from scalar data, over vector data, to tensor data. In Chapter 3, the visualization of scalar data with a large value range is discussed. Multiple scales are used to provide a linear representation over the full range of values. This approach was developed and published together with Filip Sadlo, Michael Burch, and Daniel Weiskopf [3]. Besides writing parts of the paper, Filip Sadlo helped with the development of the scale selection algorithm and Michael Burch provided his experience with chart design.

Chapter 4 is related to time-dependent vector fields. Two novel glyph-based approaches and an accelerating scheme for existing flow visualization techniques are presented. The first approach—*flow radar glyphs*—uses glyphs to display the local flow direction over time with a radial mapping. This technique and the respective publication [4] result from joint work with Philipp Leube, Wolfgang Nowak, and Daniel Weiskopf. Philipp Leube and Wolfgang Nowak contributed

to the extension of the glyph to uncertain vector fields (see below) and helped proofreading the paper. While flow radar glyphs provide an Eulerian view on flow, the second approach—*pathline glyphs*—allows a Lagrangian analysis of unsteady flow. It uses a dense seeding of down-scaled pathlines to visualize particle paths without overlap. Filip Sadlo, Hajun Jang, and Daniel Weiskopf were involved in this work and the related publication [5]. Filip Sadlo provided the datasets and his experience with flow visualization and helped writing the paper. The basic approach was investigated by Hajun Jang in his student thesis ("Studienarbeit") supervised by me. I extended the approach later with improved seeding, additional color codings, and proper interaction concepts, and developed a 3D version of the glyph. The last presented approach related to vector fields is an acceleration scheme for a whole class of flow visualization techniques. With *hierarchical line integration*, the computation of dense sets of integral curves can be accelerated by using a hierarchical computation scheme. This leads to a logarithmic computational complexity with respect to the integration length and reduces the computation times of visualization techniques like *line integral convolution* (LIC) or the *finite-time Lyapunov exponent* (FTLE). This method bases on joint work with Filip Sadlo and Daniel Weiskopf [6], who wrote parts of the paper and helped proofreading it. Filip Sadlo additionally provided datasets and derived the theoretical error order of the approach.

Chapter 5 is related to tensor fields and discusses a technique for visualizing symmetric second-order tensor fields. It is based on the FTLE and is applicable to fields that describe multiple directions for every spatial position, as it is the case for tensor fields. Like the FTLE, the method shows the separation of nearby trajectories. It is demonstrated for diffusion and stress tensor fields. This chapter bases on joint work with Joachim Vollrath, Filip Sadlo, and Daniel Weiskopf [7]. Filip Sadlo provided his experience with the finite-time Lyapunov exponent and wrote parts of the paper. Joachim Vollrath provided datasets, helped writing the paper, and supervised my Diplom thesis [140], in which the basic concept was investigated. In my Diplom thesis, a method inspired by the FTLE was developed for diffusion tensor fields. Based on these results, the approach was generalized in the paper [7] and analyzed in more detail. The paper contains a formal derivation of the concept and additionally describes the application to stress tensor fields. Furthermore, a comparison with other tensor field metrics is included and the influence of noise is analyzed.

The last part of this thesis focuses on uncertainty (Part III). Different methods are presented that target at different forms of uncertainty. The first method is related to uncertainty in the simulation model. In the presented scenario, several parameters of the simulation model are unknown. An approach similar to computational steering is used to allow the user to interactively explore

the parameter space of the simulation. This work was developed together with Sergey Oladyshkin and Daniel Weiskopf [8]. Sergey Oladyshkin provided the use case and datasets, and helped implement the methods for processing the polynomial data. The next presented method was designed to visualize uncertainty in the data. In this case, uncertain vector fields are considered. They result from simulations of underwater ground water flow and the uncertainty exists in the direction and magnitude of flow. Flow radar glyphs are extended to represent the uncertainty in these unsteady flow fields. The method is again a result of joint work with Philipp Leube, Wolfgang Nowak, and Daniel Weiskopf [4]. Philipp Leube and Wolfgang Nowak provided the use case and datasets, helped evaluate the method, and wrote related parts of the paper. The last method treats uncertainty in user interaction in the context of interactive flow visualization. Using an interactive probe to explore the flow is a common approach there. Uncertainty in user interaction is introduced by the input device, a computer mouse in this scenario, and the processing of the input data by the operating system and the visualization tool. The effect of this uncertainty is especially strong in areas where the flow changes much. Therefore, a data-adaptive method is presented that changes the speed of the mouse in dependence of the flow properties at the current probe position. This lowers the risk of missing important features at positions where the flow changes much. This method was developed and published together with Filip Sadlo and Daniel Weiskopf [9]. Filip Sadlo helped formalizing the concept and writing the paper. He additionally provided the datasets and his experience with flow visualization.

I was involved in the publication of further papers during my PhD research. Two of these papers introduce methods for the visualization and analysis of eye tracking data [10, 11]. Another paper deals with the visualization of randomly generated hierarchies with a bubble tree layout [12]. In this case, the focus laid on visual aesthetics and the generation of visually pleasing images. The topics of these papers are not in the scope of this thesis and respective material is therefore not included.

1.4 Hardware and Software Specification

All results presented in this thesis were obtained with the following hardware and software environment:

- CPU: Intel Core 2 Quad Q6600 with 2.40 GHz clock rate
- RAM: 4 GB DDR 2
- GPU: nVidia GeForce GTX 560 Ti with 2 GB GDDR5 video memory
- Operating system: Microsoft Windows 7 Professional, 64-bit

All presented techniques and tools, except for the scale-stack bar charts in Chapter 3, were implemented in C++ with Microsoft Visual Studio 2008. OpenGL is used as graphics API. The scale-stack bar charts were implemented in Microsoft Excel 2010 with the integrated macro programming language VBA. The methods in Chapter 4 (vector fields) and in Chapter 6 (uncertainty) use nVidia CUDA 3.2 for computations on the GPU. The visualization technique for tensor fields in Chapter 5 uses the OpenGL fragment shader for GPU computations.

Part I

Modeling and Control

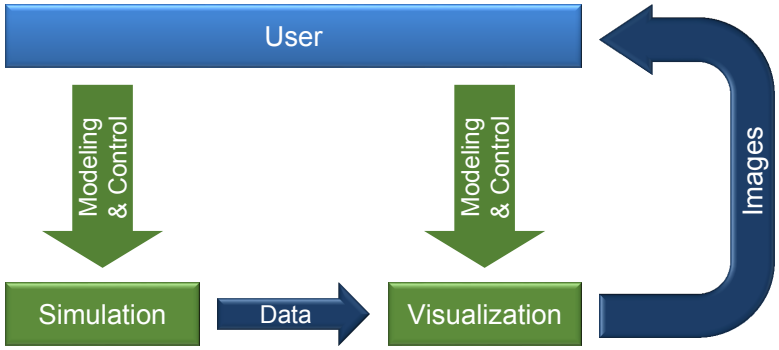


Figure I — Simulation and analysis process: the user models and controls the simulation as well as the visualization of the outcome.

Working with simulations typically requires that the user adapts the simulation to the problem under investigation (Figure I). In many cases, this adaptation requires more than just changing some parameters. For example, there might be different theoretical models or different solvers for a specific model. Therefore, simulation software or parts of it are often developed as modular framework, e.g., the PANDAS [245] and the DUNE [38] frameworks. This offers the flexibility to adapt the simulation to a specific problem or to easily exchange parts of it like, e.g., solvers. The same holds for the visualization of the simulation outcome. Creating effective visualizations strongly depends on the application and typically requires the combination of different visualization techniques. A modular visualization framework enables the user to combine different visualization modules for a specific application. The following chapter discusses visualization approaches that can improve the work with such modular software. The focus lies on workflows for modeling the interplay of the software components or modules and how to visualize the respective workflow structure.

CHAPTER 2

Visualizing Workflow Evolution

Modular software frameworks allow users to create custom software for their specific needs by combining a set of software modules. In many cases, using such frameworks requires the user to reference the respective libraries in the source code and to call their respective functions, i.e., by writing code. However, there are also frameworks that allow combining software modules on a more abstract level. Typically, a graph for the data flow between modules and parameters for executing these are specified, which can be seen as a *workflow*. For example, a module for loading the dataset is connected to a filter module removing noise from the data. This module is in turn connected to a module computing a histogram, which is the result intended by the user.

In the context of simulation and visualization, workflows can undergo many adaptation steps. Creating a suitable simulation model or visualization is typically an iterative process. Furthermore, existing module workflows are often reused for similar problems. The evolution of a workflow shows the steps users made to generate the final workflow and result. This can help recapitulate finished work and reuse parts of previously created workflows. Furthermore, it can provide information about characteristics of the underlying systems, related bottlenecks, and usability issues. This chapter[1] presents two approaches for visualizing the evolution of such workflows.

[1] **Parts of this chapter have been published in:**
M. Hlawatsch, M. Burch, F. Beck, J. Freire, C. Silva, and D. Weiskopf. Visualizing the evolution of module workflows. In *Proceedings of the International Conference on Information Visualisation (IV) 2015*, pages 40–49, 2015 [1], © 2015 IEEE.
M. Hlawatsch, M. Burch, and D. Weiskopf. Visual adjacency lists for dynamic graphs. *IEEE Transactions on Visualization and Computer Graphics*, 20(11):1590–1603, 2014 [2], © 2014 IEEE.

2.1 Module Workflows

There seems to be no common term for the data flow graph defining the interplay of different software modules. In the context of scientific data processing, e.g., in bioinformatics and other life sciences, the term *scientific workflow* [36] is used. *Business workflows* [13] usually include the execution of software components as well, but are not restricted to this. This holds even more for the term *workflow*, which describes the usage and combination of resources and tasks in general. The term *(software) pipeline* is also related, but it normally describes the more restrictive case of a linear sequence of processing steps. Therefore, the term *module workflow* is used in the following to denote the usage and combination of software components.

2.1.1 Characteristics

The usage of module workflows (e.g., the visualization workflow shown in Figure 2.1) exhibits some conceptual differences to classical software development, where source code is written and then compiled to create an application. The applications from classical software development typically offer some flexibility and a set of features that can be used to solve a variety of different problems with similar context. The work with such applications does not require that the user modifies code. The offered features are used to generate the intended result from the input data during runtime. Since code and execution are conceptually separated, analyzing the evolution of software usually does not involve information about the runtime configuration and usage of the application.

With module workflows, parts of the development process are shifted to the runtime of the software and the boundary between developing and using the software is blurred. There are similarities to working with interpreted or scripting programming languages. Module workflows are typically created for a very specific and narrowed purpose and their output directly contains the requested information. Hence, analyzing the evolution of module workflows provides information about the usage phase of the software as well. However, it must be distinguished between the execution of the framework providing the module workflow and executing the workflow itself.

The visualization technique in Section 2.2 was designed under the assumption of certain characteristics of module workflows and their usage. First, these workflows combine modules that typically provide quite complex functionality, e.g., filtering or transforming data. Therefore, this corresponds to software development on a high level of abstraction. The number of used modules is rather low compared to the number of used functions in a typical software

project. In many cases, there are no complex dependencies between modules. Even if there are branches in the workflow and modules can be connected to several other modules, the overall structure is usually quite sequential and similar to software pipelines. Furthermore, every module is employed for a single operation only. In the case that a module provides different operations, multiple instances have to be created to apply the different operations.

In typical applications, the developer and the user of the module workflow is the same person. Since the aim of this person is to generate results and not software, the term *user* is used in the remainder of this chapter. Furthermore, many systems allow the user to create the module workflow with a graphical tool, e.g., modules can be placed on the work space and connections can be drawn between them (see Figure 2.1).

Another important aspect is that all relevant parameters are set before the workflow is executed, i.e., setting parameters is also part of modifying the workflow. In many cases, this leads to repeated modifications of the workflow, e.g., when the parameter space of an operation is explored.

Finally, the use of undo operations and the reuse of previously created workflows lead to branches in the workflow evolution.

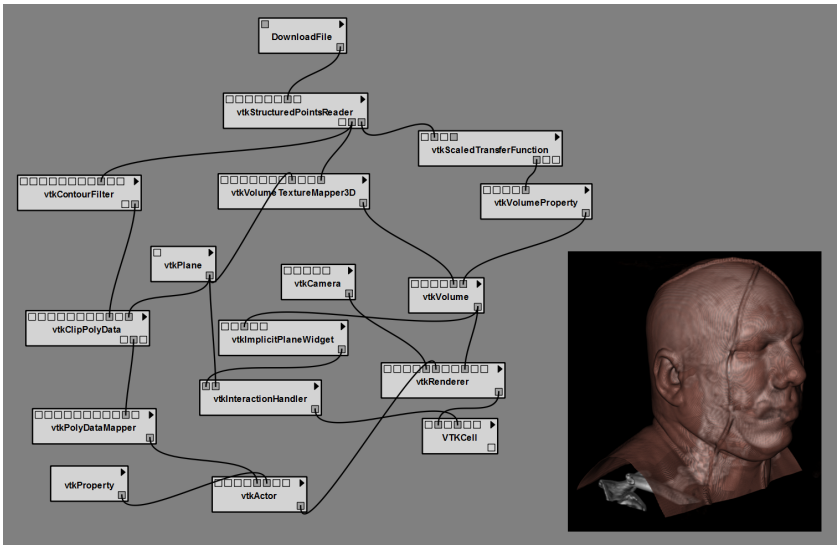


Figure 2.1 — Example of a module workflow. Different modules are connected to process data and generate the desired result. This example shows a module workflow from VisTrails [64] that generates a volume rendering of a human head (small window, bottom right).

2.2 Visualization for Module Workflows

The visualization approach introduced here depicts the evolution of workflows recorded for usage sessions of a module framework. The proposed visualization setup uses multiple coordinated views and consists of three main elements (Figure 2.2): First, a visualization of module activity and module changing events on a timeline (Section 2.2.2). Second, a visualization of the branching structure of the workflow evolution (Section 2.2.3). Third, different views for a detailed analysis of certain aspects of the workflow (Section 2.2.4).

The combination of these views is important to support the analysis of large workflow histories. The branch view gives an overview of all branches in a dataset and allows for selecting single branches or sequences of connected

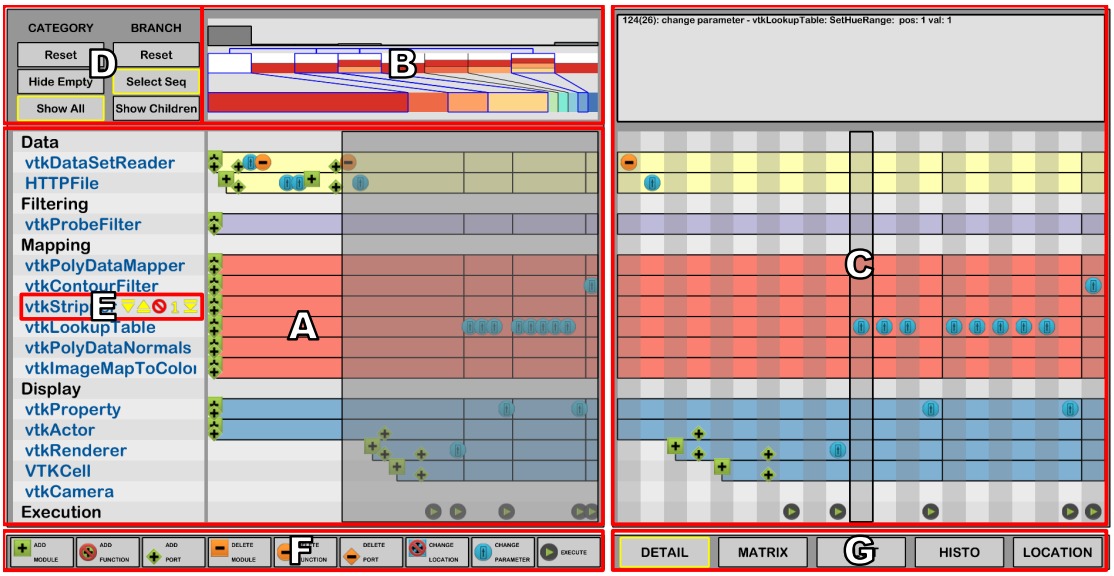


Figure 2.2 — Overview of the visualization approach with multiple coordinated views. The visualization is split in three main parts: (A) the module and event view (Section 2.2.2), (B) the branch view (Section 2.2.3), and (C) different detail views (Section 2.2.4). The module and event view and the branch view on the left side provide an overview of the workflow evolution and allow users to select and filter parts of it. The selected time span (gray area in the module and event view) can then be analyzed in detail on the right side with different detail views. Global settings and operations can be applied with the buttons in (D). The context menu in (E) appears when the mouse pointer hovers the category list and is used to (un)group modules and functions for the module and event view. (F) is the legend for the event symbols and allows filtering them. The different views for (C) are selected with the buttons in (G).

branches (see Section 2.2.3). Based on this, the module and event view shows only the selected part of the data (see Section 2.2.2). Since individual branches can still cover a large time range, the detail view is used to further drill down to a finer scale (see Section 2.2.4). The other views help analyze sequences of events, parameter changes, and the workflow layout in a graphical editor.

2.2.1 Related Work

Visualization in the context of software evolution typically focuses on modifications of the source code, for instance, the submission history of a version control system like CVS or Subversion [194, 284, 285], the changing interaction of developers with code [201, 202], or the evolution of software metrics [172, 218]. An overview of these visualization techniques can be found, e.g., in the survey by Caserta and Zendra [68] and in the book by Diehl [88]. The focus of this chapter is the evolution of module workflows: code is not changed but the configuration of the system represented by the workflow, such as module usage and module parameters. This aspect of software evolution has not yet been studied from a visualization perspective.

There is a close relation to *provenance*. The term describes all information required to reproduce a result. Hence, in the context of workflows, provenance information describes also the evolution of the workflow. Respective research [80, 81] mainly focuses on capturing provenance and accessing it with query languages. However, little attention has yet been devoted to the problem of visualizing such information. Macko and Seltzer [186] developed a tool for exploring large data provenance graphs. They use graph summarization and apply semantic zooming to enable users to incrementally explore the details of the graph. Rio and Silva [227] and Anand et al. [25] use visualization to support querying the provenance information. In contrast, the approaches in this chapter visualize the evolution of workflows and focus on the temporal aspects of the modules used and related events.

Target applications for the presented visualization approaches are all systems that allow a workflow-like combination of modules or components. Examples are systems for scientific workflows like Kepler [22], Taverna [204], or Galaxy [117]. There are also many visualization applications and frameworks available that allow for building the designated visualization by connecting and combining different modules: AVS [273], VTK [248], MeVisLab [228], the InfoVis Toolkit [101], prefuse [134], etc. VisTrails [64] has a special role in this list: besides allowing a workflow-based combination of visualization modules, it has integrated support for provenance.

The presented approach is also related to techniques for visualizing time-dependent or time-oriented data [20]. Using bars for visualizing time is a common approach, sometimes called *timelines* [267] and variations of it can be found in different works [19, 71, 163, 219]. A well-known example of this type of visualization, often applied in project management, are Gantt charts [111], which represent tasks and their temporal dependencies on a timeline. Related to the scenario of studying the evolution of modules, Viégas et al. [281] use bar-like diagrams to visualize the change history of Wikipedia. Waser et al. [292] apply enhanced bar charts with branching and symbols to control simulations in a computational steering environment. Using bars is also a common approach in trace visualization of parallel systems [126, 264]. Burch et al. [59] combine timelines with a tree visualization grouping the individual lines. With respect to indicating events on the timeline, Wang et al. [288] use symbols and histograms to display events. A combination of bars and symbols are used by Cousins and Kahn [78] to visualize different types of events in patient data.

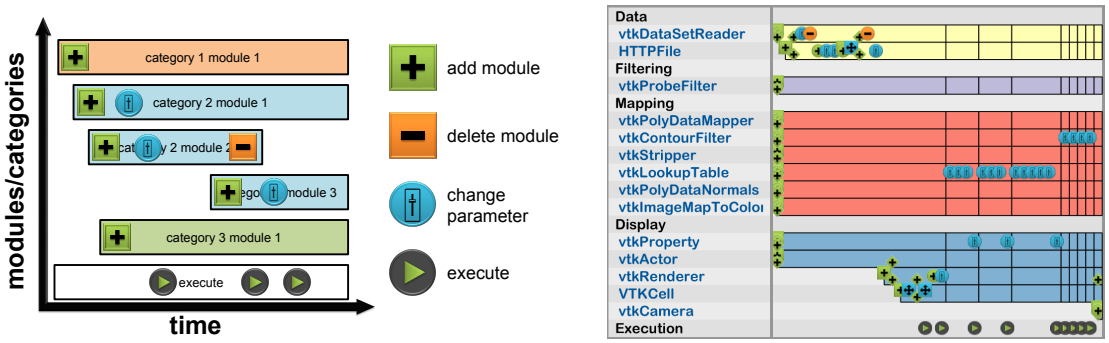
Although some of the proposed visualization techniques exhibit similarities to the ones mentioned above, a new approach for visualizing the temporal and causal relations of branches in the workflow evolution is presented. Furthermore, the overall visual design and combination of techniques together with the application to module workflow histories is novel.

2.2.2 Module and Event View

For the visualization of the module lifetime and events related to them, a central aspect is how to represent time and time spans. While events, e.g., adding a module or changing a parameter, occur at a specific point in time, the existence of modules over the workflow evolution is associated with a time span. The time span for a module corresponds to the period between the time it is added and the time it is deleted. To account for these aspects, the module and event visualization consists of two key elements: representations of time spans and representations of events (Figure 2.3).

First, as Figure 2.3(a) illustrates, bars are used to represent the time spans of the modules in the workflow. This choice was inspired by Gantt charts [111], which are effective for visualizing tasks and time spans. They allow one to easily assess temporal extents and to compare them.

Since module workflows may contain a large number of modules, they are clustered into a set of categories to improve scalability. The time spans of the categories correspond to the unity over the time spans of the modules inside the categories. The categories can be expanded to display the modules represented by them (in the implementation with a context menu, see Figure 2.2). Further-



(a) (b)

Figure 2.3 — The evolution of workflows is visualized by displaying the time spans of modules used and events related to them. The basic scheme is illustrated in (a); (b) shows its realization in the developed tool. The time spans of modules are shown as bars, where time is on the x-axis and the different modules are on the y-axis. Vertical lines on the bars indicate the beginning of a new branch. Events are represented by symbols and are superimposed on the bars—the position of a symbol marks the time of the corresponding event and the related module. Categories are used to group modules, allowing users to drill down to an appropriate level of granularity. Color coding and the positions of the modules indicate to which category they belong. Since the execute event operates always on all modules, it has its own category.

more, modules can also be isolated or removed to focus on a specific subset. In the case study (Section 2.5), workflows from the visualization framework VisTrails are analyzed. Therefore, the implementation uses a simple classification scheme for the modules with four categories that reflect high-level tasks inspired by the visualization pipeline proposed by Haber and McNabb [125]: data, filtering, mapping, and display (Figure 2.3(b)).

Second, since events are associated with a specific point in time, symbols are used to represent them (Figure 2.3(a)). By superimposing symbols on the bars of the module chart, their context is preserved both with respect to time and related modules—it is easy to see when they occur and to which modules they are related.

To ease the interpretation, a legend for the event symbols is shown below the module and event view (see Figure 2.2). There are four types of events: add, delete, change, and execution events. The execution event is special because it represents the execution of the workflow and therefore operates on all modules and functions. Thus, a separate category is used for it; otherwise, the execution symbol would always appear on all rows.

Displaying a large number of events simultaneously is problematic. Therefore, it is possible to filter events and show only the ones of interest, e.g., parameter change events. Furthermore, the symbols for different types of events were designed with different shape, color, and slight changes in position. For example, events affecting directly a module (add, delete, change position) are squares and slightly shifted to the top. Add events are green, delete events orange, and change events blue. The goal is that they can be still differentiated even when they overlap to a certain degree (see Figure 2.3(b)).

The combination of the visualization techniques for time spans and events is important. Displaying only the time span of modules would neglect events that do not have a direct influence on the time span of modules, e.g., changing parameters. In contrast, the symbol representation fully represents the evolution of the workflow because it provides a direct visualization of all user actions. However, to deduce the set of active modules, it would be necessary to follow and memorize all previous events. Only the combined visualization clearly reveals the currently active modules and related events.

To provide more information about the workflow structure, it can be considered to visually connect the modules, e.g., by drawing lines between the modules. However, visual connections clutter the visualization and increase the visual load. Therefore, module connections are not shown in the developed visualization at the cost of losing information about data flow. It is assumed that module categories and names already provide a good-enough impression of the purpose of the workflow. Furthermore, the creation and deletion of connections is displayed as events. However, there might be applications where the order of module connections has a huge influence on the outcome of the workflow. Section 2.4 therefore presents a technique for displaying the connectivity information in workflow histories.

In the case of large workflow histories, the proposed module and event view can exhibit visual clutter due to displaying a large number of overlapping symbols. Furthermore, displaying events in a linear fashion neglects the occurrence of branches, which are created when users continue from a previous state, e.g., by using undo functionality. If branching occurs, two events that follow each other in a temporal sense can operate on completely different states of the workflow. To address this problem, an additional visual representation of branches was developed that can also be used for selecting them.

2.2.3 Visualization of Branches

The second element of the proposed visualization is a visual representation of branches in the workflow evolution. The term branch is used here with respect

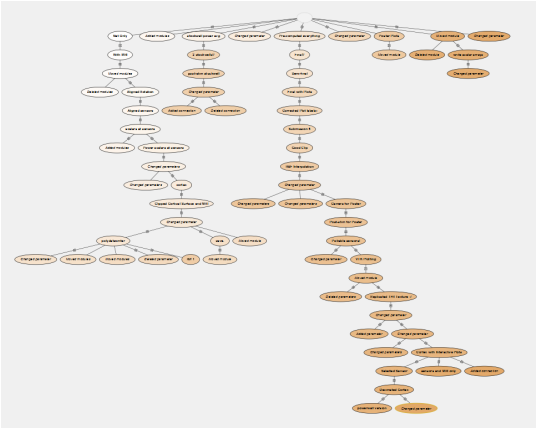


Figure 2.4 — The history view of VisTrails is an example of representing the branches of a workflow as a tree. In this case, the saturation of the nodes denotes time and the edges can be expanded to show all events between two nodes.

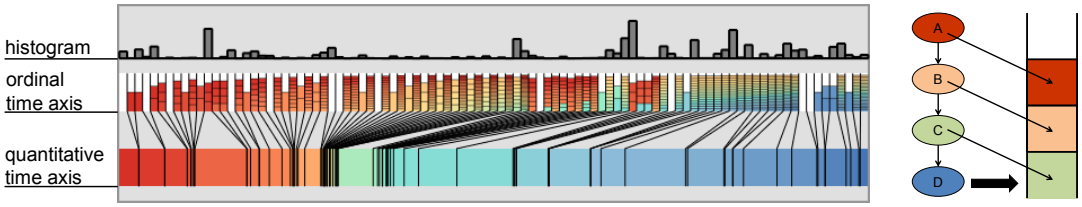
to the causality of events, i.e., they occur when the user returns to a previous state of the workflow and continues from there. Hence, while all events in a workflow history form a linear sequence with respect to the time they occur, the causal relationship between them forms a tree structure because multiple sequences of events can evolve from a single state.

In the design of the branch view, two key issues were considered: conveying the temporal order of the branches and scalability with respect to the number of branches. Since the branch view is displayed together with the module and event view (see Figure 2.2), it should exhibit a small spatial footprint.

As Figure 2.4 shows, a common approach to visualize branches—e.g., implemented in VisTrails—is a node-link representation of the branch tree. Although this representation conveys the relation between branches well, it is typically not space efficient and the temporal order of branches is difficult to obtain. Therefore, a different approach is used.

The branch visualization is divided into three parts: an ordinal time axis, a quantitative time axis, and a histogram (Figure 2.5). The primary interest lies in the temporal order of branches. Therefore, the ordinal time axis shows the branches as a linear sequence of equally sized columns to provide good scalability with respect to the dynamic range of different time spans. Since the user should still be able to retrieve the causal relationship of branches, these columns are further divided into blocks, which are used to represent relationships to previous branches (Figure 2.5(b)). Each block represents a parent branch, i.e., the stacking of the blocks represents the causal relationship of the branches. The color of the blocks indicates the time stamps of the parent branches. This approach provides the user with information about the number of parent branches and their temporal occurrence.

To also provide a quantitative representation, the time spans of the branches are additionally shown on a quantitative time axis at the bottom. This axis is



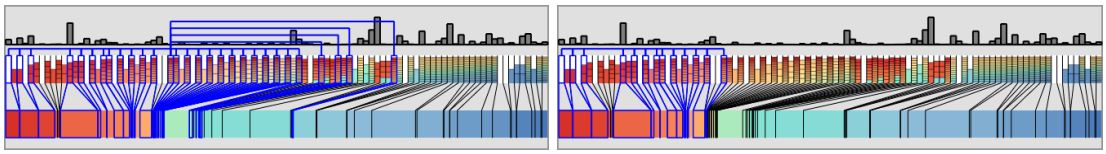
(a) (b)

Figure 2.5 — Visualizing branches in the evolution of a workflow. (a) The branch representation consists of multiple components: On the ordinal time axis in the center, the branches are shown as equally sized columns, neglecting their unequal time spans. The blocks inside these columns represent all parents of the branch. This is illustrated in (b): All parent branches (A–C) of branch (D) are shown as small blocks inside the column of branch (D). The colors of the blocks represent the time stamps of the parents. The respective color map is shown on the quantitative time axis on the bottom; white indicates that there are no further parents. The quantitative time axis additionally shows the real time spans of the branches with connected lines. The histogram on top indicates the number of events in the branches shown on the ordinal time axis below.

also used to display the color map for the time stamps of the parent branches in the ordinal axis. The combination of both axes provides good scalability with respect to branches of very different length in time without discarding their quantitative relation.

Finally, the histogram on top displays the number of events in the branches shown on the ordinal time axis. Please note that the information provided by the histogram does not necessarily correlate with the time span of the branch. Since some operations apply to a group of modules (e.g., pasting or moving a group of modules), several events may occur at the same point in time; a branch with a smaller time span may contain more events than a temporally longer one. When the number of events is changed by filtering (see Section 2.2.2), the histogram is adapted accordingly.

This visualization provides only an initial overview of the temporal aspects of the branches. It does not directly show which branches are connected. Therefore, the user can additionally show all connected branches of a selected branch in a tree visualization (Figure 2.6(a)). To reduce visual load, displaying related child branches can be deactivated (Figure 2.6(b)). Typically, the parent branches are more interesting because the current state of the workflow is a result of them.



(a) (b)

Figure 2.6 — Enhanced branch view. (a) To provide a clear view on the relationship of branches, all connected branches of a selected branch (by mouse hovering) can be shown with an orthogonal tree visualization on top of the branch representation (blue lines). (b) To reduce visual clutter, only the parents of a branch can be highlighted.

2.2.4 Detail Views

To support a detailed view without losing context and to further improve the scalability with respect to the length of the workflow history, several detail views were developed, which are placed on the right side of the visualization setup (Figure 2.2). The default view acts as a lens of the event view, i.e., it zooms in and magnifies the selected time range. Additionally, the info box at the top provides detailed information about single events (see Figure 2.2). The second view (Figure 2.7(a)) is a matrix showing the number of transitions between two modules with respect to subsequent events. If there are two subsequent events, with the first operating on module A and the second on module B, the entry in the matrix for row B and column A is increased. Hence, a row in the matrix provides all predecessor modules with respect to the occurrence of events for the respective module. Next, the plot view (Figure 2.7(b)) displays a plot over the selected time range of the parameters for all expanded functions of the modules, with each parameter shown in a separate row. As an alternative view, a histogram (Figure 2.7(c)) can also be shown for these parameters. In contrast to a classical histogram, not the recurrence of values is counted, but the accumulated time range of the respective value range is shown. The last view is a plot of the locations the modules have in a graphical workflow editor (Figure 2.7(d)). The purpose of this view is mainly to analyze usability issues with respect to placement of modules. All views relate to a user-selected area in the module and event view. Filtering events also affects these views. For instance, the transition matrix considers only parameter change events when other events are filtered out.

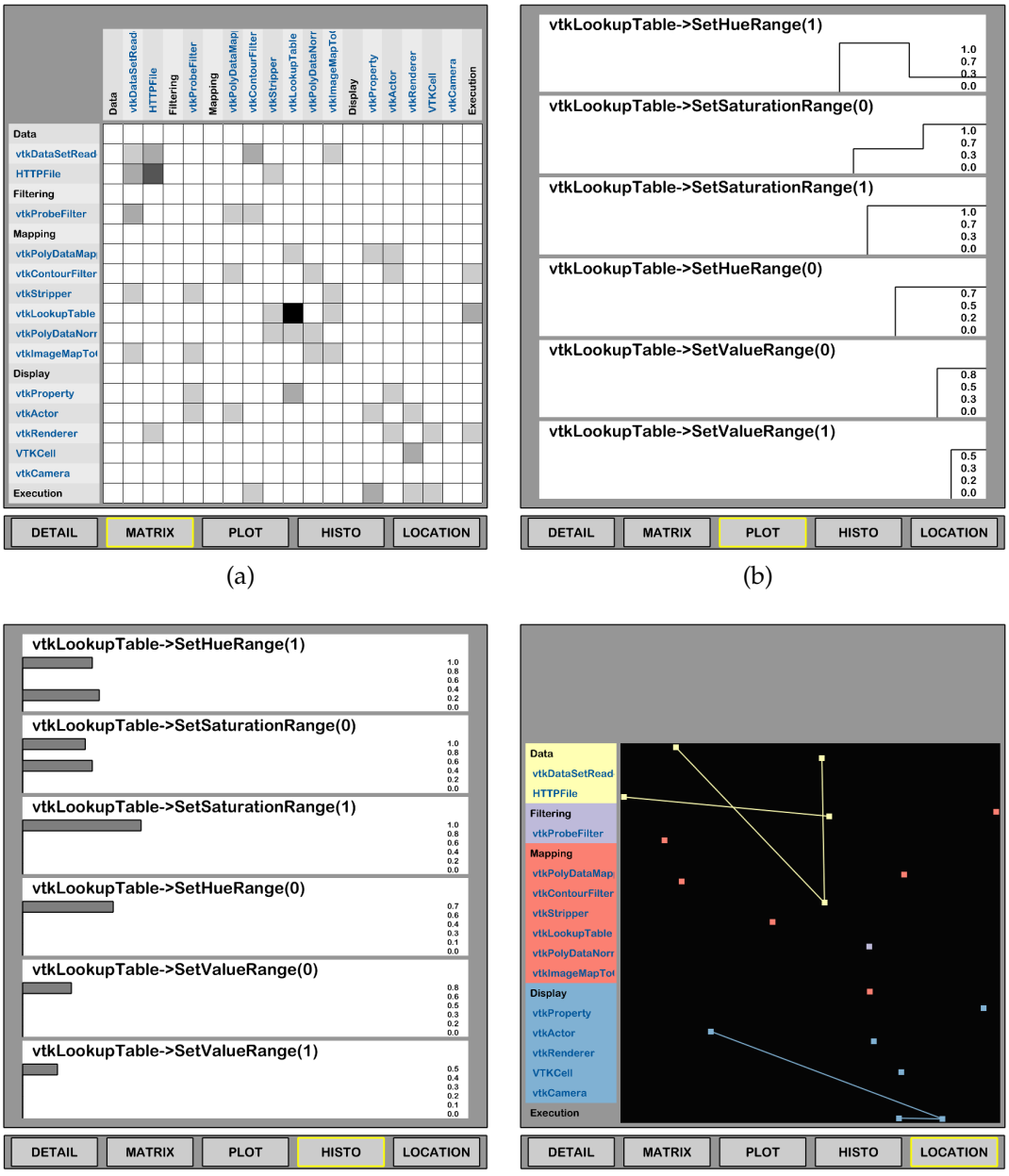


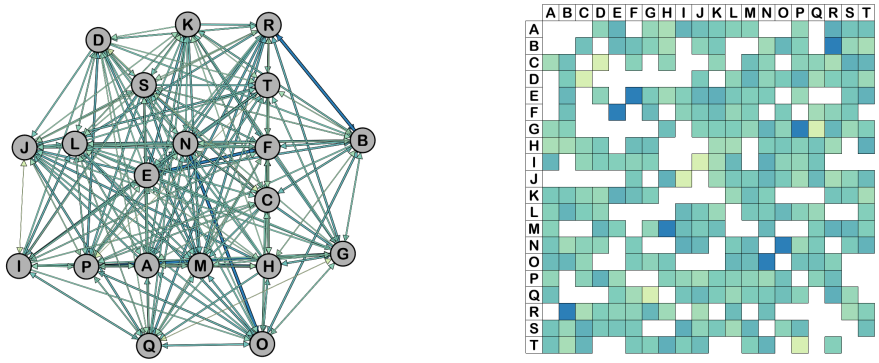
Figure 2.7 — Different views revealing specific details of the workflow. (a) A transition matrix for events, (b) a plot and (c) a histogram for parameter values, and (d) a visualization of the module positions in the graphical workflow editor.

2.3 Visualizing Dynamic Graphs

The previously described approach for visualizing the evolution of module workflows (Section 2.2) has some limitations. One limitation is that the change of the connections between modules over time is not directly represented (see Section 2.2.2). Although this might not be an issue in many application scenarios (Section 2.5), there might be applications where this connectivity information is relevant. This problem can be approached from the perspective of graph visualization, because the connectivity information can be modeled as a graph. Building on this, dynamic graphs can be used to model the change of connectivity if the module workflow is modified.

A graph G is defined as a set of vertices (or nodes) V and the edges (or links) E connecting them: $G = (V, E)$. Graphs and related problems are a fundamental research topic in mathematics and computer science [77]. Graphs are commonly used to model and represent relations or connections between entities or objects. They can be found in many different application areas, e.g., to represent computer networks, social relationships, or biological processes. Therefore, the visualization of graphs is an important visualization problem.

Many visualization techniques exist for the analysis and exploration of static graphs [85, 153]. A prominent visual metaphor is the node-link diagram (Figure 2.8(a)), which can become cluttered for dense graphs [231]. For dense graphs, matrix-based representations (Figure 2.8(b)) are more suitable [114]. However, they exhibit problems when dealing with path-related tasks such as finding a shortest route from a source to a target node. Matrix visualizations hence trade visual clutter for path reading problems. Furthermore, it is difficult to compare and aggregate the weights of links with both representations. In the



(a) (b)

Figure 2.8 — (a) Node-link and (b) matrix visualization of a static graph.

case of layered graphs, Quilts [32] are an approach to visualizing them with a compact matrix-based layout. DAGView [162] uses a combination of node-link and matrix representation.

Visual adjacency lists, which are presented in Section 2.4, share a few characteristics of matrix techniques: they avoid link crossings, but following paths is difficult (see Section 2.4.4). However, they use a list-based representation resulting in a more compact visualization, which is especially useful for sparse graphs and dynamic data, such as module workflows.

Visualizing the change of connectivity in module workflows requires visualization techniques for dynamic graphs [42]. Visualizing dynamic graphs is typically done by a natural time-to-time mapping exploiting the concept of animation. Two sub-concepts exist: The first requires the whole graph sequence in advance, denoted as off-line dynamic graph visualization [89, 98, 169]. Complementing that, online dynamic graph visualization [107, 179] requires only the next element of the sequence at the time when it is rendered. Both approaches are subject to high algorithmic complexity to produce aesthetically pleasing graph layouts [41, 43, 226, 291] that preserve the viewer's mental map [192] and achieve a high degree of dynamic stability [197]. Extraction and visualization of clusters can also be done for dynamic graphs [142, 239, 280]; recent work also considers the dynamics of associated attributes [127].

In contrast, the concept of visual adjacency lists is based on a static representation for dynamic graphs, i.e., a time-to-space mapping. The static visual representation has several benefits: Graphs can be compared visually and the viewers do not have to rely on their short-term memory [28, 269] as it would be required with animation. The mental map is preserved and dynamic stability is achieved, which also allows for an easy integration of interaction techniques.

The concept of static diagrams for time-varying node-link diagrams has been used in TimeArcTrees [120]. Another example of static visualization based on the node-link visual metaphor is parallel edge splatting [60]. A further possibility is to use a layered approach of node-link diagrams for visually encoding the dynamics of a network in a stacked 3D representation [50]. Finally, there is a hybrid approach combining animated and static node-link visualization [233].

Matrix-based techniques can also visually encode dynamic graphs [57, 59, 309] in a space-filling and clutter-free approach to illustrate time-varying weights by attaching the time axis to each of the matrix cells. However, they are not space efficient in the case of sparse dynamic graphs, such as module workflows.

2.4 Visual Adjacency Lists

In the following, a novel approach to visualizing dynamic graphs is presented which exploits the concept of adjacency lists. By using a list-based representation, a compact and clutter-free graph visualization is achieved. Furthermore, it is possible to analyze evolving graphs in a side-by-side display with the focus on the investigation of link duration.

2.4.1 Concept

The graph visualization presented in the following is based on adjacency lists, which are a well-known concept for graph representation [77]. However, adjacency lists are typically only used to internally represent graphs in computer memory and algorithms. Here, they are used to generate a visual representation of the graph, called *visual adjacency list* (Figure 2.9). Two orthogonal axes were chosen for the list, but the concept is not restricted to this; other layouts like radial approaches are possible.

Nodes are represented by graphical primitives, called *node elements*, which can have a label and/or a color. Box shapes are used for the node elements. Links are represented by the node elements of the target nodes. All nodes of the graph are listed on a vertical axis and the target nodes of outgoing links are placed on a horizontal axis. Labels specify the target nodes. With this layout,

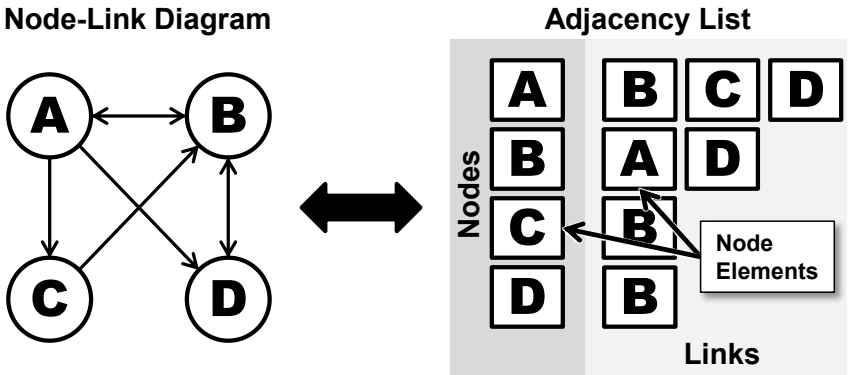


Figure 2.9 — Concept of visual adjacency lists. The example graph is shown as classic node-link diagram (left) and as visual adjacency list (right). The list-based approach uses two orthogonal axes: one axis for all nodes in the graph (vertical), and one axis for the corresponding target nodes of outgoing links (horizontal). Links are defined by their vertical position along the node axis and their label, e.g., node “A” has outgoing links to “B”, “C”, and “D”.

people used to reading from left to right can intuitively read the list.

Typical graph visualization techniques like node-link or adjacency matrix visualizations are symmetric with respect to links: source and target of a link are represented with the same visual encoding. In contrast, the proposed layout results in an asymmetric representation of links. While the source nodes are spatially arranged, the target nodes of links are only encoded with labels and/or colors. In this way, the visualization is more focused on source nodes and more suitable for tasks centered on them.

As a consequence of the underlying concept of adjacency lists, this visualization is a complete representation of an unweighted graph. It is directly applicable to both directed and undirected graphs. Furthermore, adjacency matrices and lists are strongly related (Figure 2.10). The list can be regarded as a version of a matrix without gaps because the horizontal position is not required to specify links. Therefore, the list can be derived from the matrix by shifting the entries to the node axis, removing all gaps in between, and the matrix can be derived from the list by assigning the proper horizontal positions to the node elements.

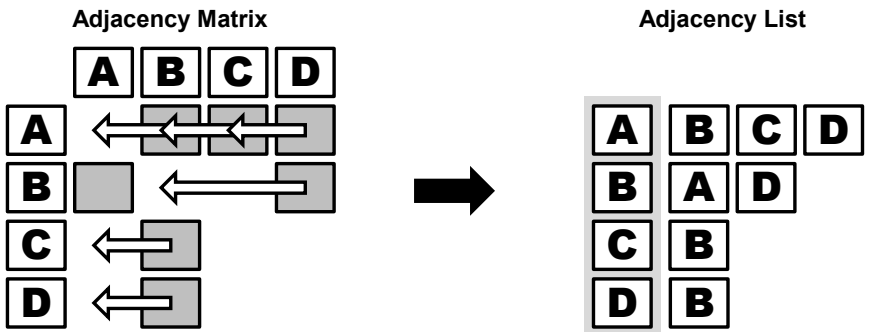


Figure 2.10 — Relationship between adjacency matrix and list. The adjacency list can be derived from the adjacency matrix by shifting all entries to the left until there are no gaps (and by labeling the entries).

2.4.1.1 Incoming Links

With the current representation, it is hard to extract all incoming links of a specific node because the whole list has to be read. Hence, this operation is linear in the number of all links, while extracting all outgoing links of a specific node is only linear in the number of the outgoing links of that node. To improve on this, incoming links are explicitly represented with an additional horizontal axis from the center to the left (Figure 2.11). In this way, the reading scheme for links remains from left to right, i.e., source nodes are always located left

from target nodes. Since the node axis is now placed in the center, it should be visually separated from the link axes. To this end, the size of the node axis is increased to support scalability to large graphs.

The explicit representation of incoming links doubles the spatial extent, but the completion of many tasks related to incoming links is accelerated. In the case of undirected graphs, the additional axis for incoming links can be omitted due to the symmetry of these graphs.

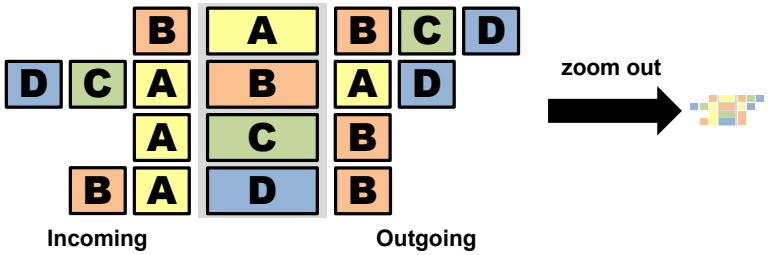


Figure 2.11 — Incoming links can be explicitly represented with an additional axis (left of the node axis here) to support tasks on them. Node “A" has, e.g., only one incoming link from node “B" (see Figure 2.9, left). To improve scalability, colors are used in addition to labels to specify nodes; they are still recognizable when scaling down the visualization. The color map is defined on the node axis in the center. Increasing the width of the node axis visually separates it from the link axes even on zoomed-out levels (right).

2.4.1.2 Color Coding and Node Order

So far, target nodes of outgoing links (and source nodes of incoming links) are only specified by their labels. This poses scalability problems: the visualization scales only to a size for which the labels are still readable. To improve scalability, color is additionally used to represent nodes (Figure 2.11). The color map is defined and shown on the node axis. With color coding, the visualization scales up to a level at which a node is represented by a single pixel. Going to the sub-pixel level, where several nodes are represented by a single pixel, requires appropriate schemes to select the color of this pixel and is outside the scope of this work.

Color coding and the order of the nodes have a huge impact on the resulting visualization and influence the identification of graph structures, especially on the overview level. They typically go hand in hand, since color coding is applied after ordering the nodes on the node axis. A good combination of node order and color coding supports the analysis of the graph.

Ordering the nodes on the node axis and the links along the link axis is independent from each other and flexible. Even ordering the links in every row differently would result in a correct representation of the graph. In this case, however, it is very difficult to see similarities in different rows and detect clusters. It is therefore important to find an adequate ordering for the node axis. Along the link axis, it is reasonable to use the same ordering as for the node axis.

A simple scheme is to order with respect to certain properties of the graph. For example, the nodes can be ordered according to the number of outgoing or incoming links. If nodes exhibit certain semantics or hierarchies, it is reasonable to order them accordingly if possible: the nodes of the visualization workflow in Section 2.5.4 are ordered according to the classic visualization pipeline [125]. More advanced ordering methods are outside the scope of this work and may be inspired by reordering methods for adjacency matrices [139, 182]. Furthermore, graph clustering methods [244] can be used to create hierarchies.

It is reasonable to use the hierarchy also for the color coding of the nodes. For the visualization workflow, e.g., four base colors are used for the four groups of nodes (see Figure 2.27(b)). Inside these groups, the brightness of the base colors are modulated according to the position of the nodes inside the group.

The node order and color coding may be interactively changed by the user. In this way, the user can filter or emphasize certain elements and may reveal different aspects of the graph, such as highlighting all links from and to a selected node.

2.4.2 Characteristics

In the following, the characteristics of visual adjacency lists and their suitability for graph visualization are analyzed. Especially their flexibility (Section 2.4.2.2) is an important aspect and is the basis for the visualization of connections in the evolution of module workflows (Section 2.4.3.2).

2.4.2.1 Scalability

For a detailed discussion of the scalability of visual adjacency lists, they are compared with adjacency matrices because of their close relationship (see Figure 2.10). Figures 2.12 and 2.13 show results for both techniques for synthetic graphs with varying node count or numbers of clusters. The graphs were initially generated with an adjacency matrix, i.e., links were created by setting the respective entry in the matrix. This assures an optimal row and column ordering for detecting clusters and assures that the adjacency matrix is not

disadvantaged in the comparison. For the basic graph structure, links between randomly selected nodes were created, using uniformly distributed random numbers, until the given link density was reached. To generate the cluster structures, this procedure was repeated in subareas of the matrix with a higher density.

In the first case, the link structure and density was kept constant, only the node count was changed (Figure 2.12). It can be seen that even in the case of small graphs (Figures 2.12(a) and (b)), identifying individual nodes or links requires interaction on typical displays—regardless of the chosen technique. large-scale structures, however, are directly visible in both visualizations. They even tend to appear clearer in graphs with higher node count (Figures 2.12(e) and (f)). The difference between both techniques is that the color coding of visual adjacency lists usually provides a lower resolution on typical output devices

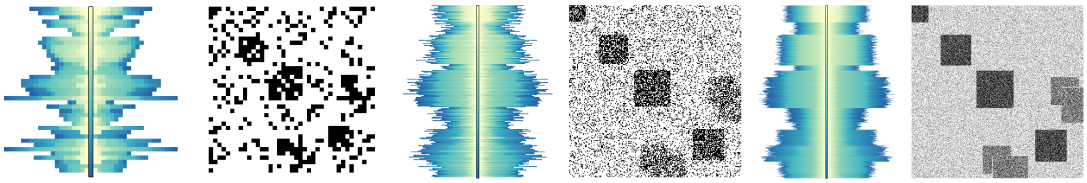


Figure 2.12 — Scalability of adjacency lists (a, c, e) and matrices (b, d, f) with respect to the number of nodes. Synthetic graphs with the same link structure, i.e., link density and cluster positions, and increasing node count ((a, b): 40 nodes, (c, d): 200 nodes, (e, f): 1000 nodes) were generated. Large-scale structures are visible independently of the node count. However, with increasing node count, the structures become clearer with both types of visualizations.

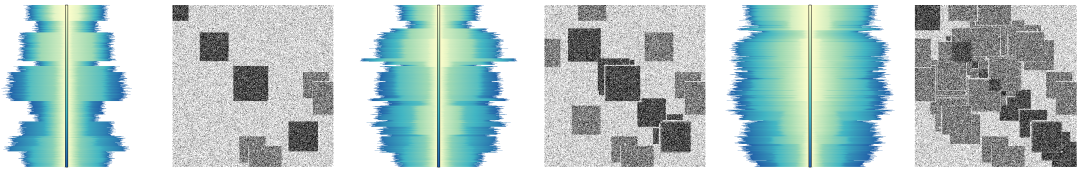


Figure 2.13 — Scalability of adjacency lists (a, c, e) and matrices (b, d, f) with respect to increasing number of clusters (from left to right) in synthetic graphs with fixed node count (1000). Even nearby cluster structures can be differentiated in adjacency matrices. This is not possible with adjacency lists due to the color encoding. Only structures with a larger distance with respect to the node order can be differentiated there.

than the spatial encoding of adjacency matrices. Therefore, the scalability of adjacency matrices with respect to the number of cluster structures (Figure 2.13) is better. While small and nearby structures are still visible and distinguishable in adjacency matrices (Figure 2.13(f)), this is not the case for adjacency lists (Figure 2.13(e)).

In summary, directly applying visual adjacency lists to large graphs can provide only an overview of large-scale structures. They offer good scalability with respect to the number of nodes. However, the representation of cluster structures is of lower resolution compared to adjacency matrices. Similar to many other techniques, a more detailed representation of large graphs requires their aggregation (Figure 2.14).

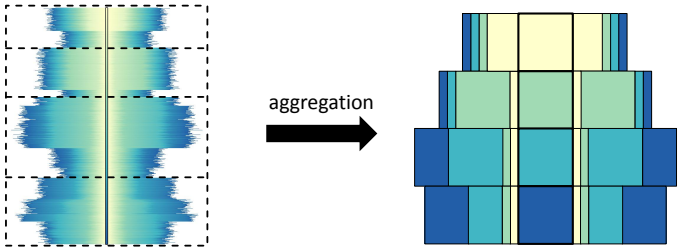


Figure 2.14 — Illustration of aggregation for adjacency lists. To improve the analysis of large graphs (left), nodes and the respective links can be aggregated, e.g., with a clustering approach. The resulting adjacency list (right) would show the representatives of the aggregated nodes and their links.

2.4.2.2 Flexibility

There are two constraints in the approach for defining links. First, target nodes must have the correct position with respect to the node axis to identify source nodes (Figure 2.15), i.e., the node element of the target node must be in the correct row to be associated with the respective source node. Second, the node element is assigned with the label and color of the target node.

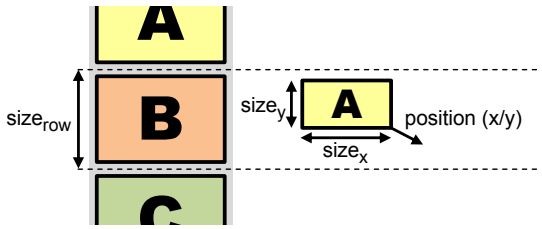


Figure 2.15 — Flexibility in link representation. Since links are defined by the associated row in the list and labels/colors, flexibility remains in the position and size of node elements. Node elements can be moved and scaled as long as they stay inside the respective row.

This leaves some flexibility to the size and position of the node element. In principle, the size in x-direction ($size_x$) is not restricted. However, the visualization becomes more difficult to read with increasing size. In y-direction, the size ($size_y$) is restricted by the size of the row, but the row size ($size_{row}$) does not have to be uniform and can vary from row to row.

Regarding the position of the node element: In x-direction, it is only required to be to the right of the node axis in case of outgoing links (and to the left in case of incoming links). If $size_y < size_{row}$, the position in y-direction is also flexible, as long as the node element stays inside the respective row.

The flexibility in the size and position of the node elements is not only the basis for the Gantt layout in Section 2.4.3.2, but can also be used to represent weights. A discussion and demonstration of weight representation can be found in the original paper [2]. Since the connections in module workflows are typically unweighted, the representation of weighted links is not further discussed in this thesis.

2.4.3 Dynamic Graphs

As previously mentioned, to represent the change of connectivity in the evolution of module workflows, dynamic graphs are required. The concept of visual adjacency lists is therefore extended to dynamic graphs. For this, the compactness of the visual representation is exploited to create a static image of the full graph sequence. It is also possible to combine visual adjacency lists with animation, with all its advantages and disadvantages. However, this work focuses on static visualization.

2.4.3.1 Discrete Time Visualization

Visual adjacency lists can be directly adopted for dynamic graphs by putting links for different points in time side-by-side (Figure 2.16). Here, it is assumed that time is discretized so that clearly separated time steps can be shown. Although the node and link structure can be dynamic, it is further assumed that only the time spans of links are of interest and therefore the union of all nodes from all time steps is used in the visualization.

The node axis is positioned in the center and the time steps of incoming and outgoing links are grouped left and right, respectively. Time evolves from the center to the outside. Slight changes in the background color and small offsets visually separate the time steps. Furthermore, by normalizing the width of the node elements with the maximum number of links, a uniform width of the time steps can be achieved.

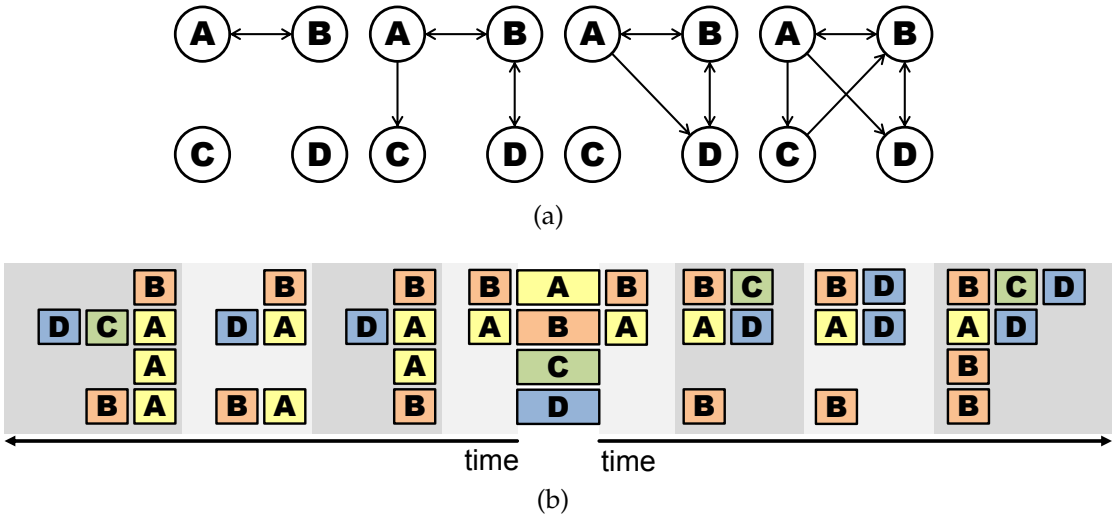


Figure 2.16 — Visual adjacency lists for dynamic graphs. (a) The example graph consists of four time steps. (b) Visual adjacency lists display the individual time steps next to each other; incoming and outgoing links are grouped and time grows from the center to the outside.

The advantage of this layout is that it is easy to analyze the evolution of incoming and outgoing links. Furthermore, the symmetry of the concept remains for all time steps. However, it is difficult to compare the incoming and outgoing links of a specific time step. Additionally, the time axes for incoming and outgoing links evolve in opposite direction, which is less intuitive to read. Therefore, other layouts are possible, e.g., the incoming and outgoing links of each time step can be grouped with time evolving from left to right.

Finally, it is difficult to track a single link over time. Its relative position on the link axis can vary in every time step due to links appearing or disappearing. Hence, at least parts of the link list have to be scanned in each time step. This can be even more difficult if links are only identified by colors. Therefore, another layout for sparse and dynamic graphs is described in the following.

2.4.3.2 Gantt Layout

With the above approach, dynamic graphs can be displayed and analyzed, but certain tasks are still time-consuming. This is especially the case for tasks related to the time span of links because all links of an individual node are horizontally stacked. Hence, deciding if a link exists at a specific time step requires scanning all links of the respective node.

To address this problem, a variant similar to Gantt charts [111] can be created for sparse graphs (Figure 2.17), which exploits the flexibility discussed in

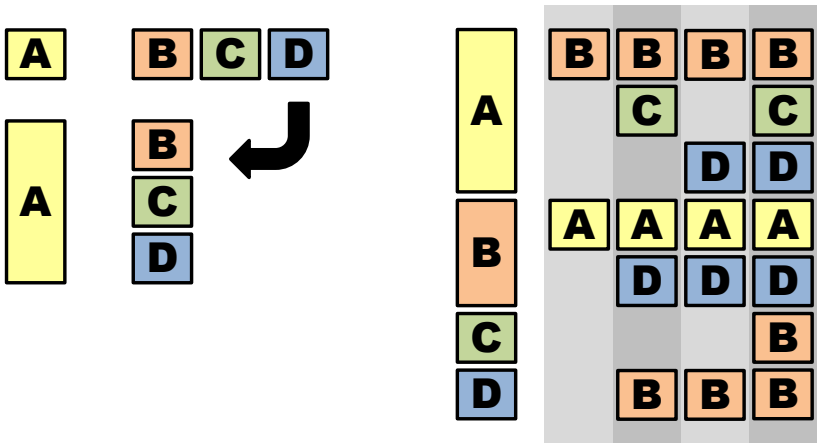


Figure 2.17 — Gantt layout. (a) It is possible to subdivide the row for a single node, so that every link has its own sub-row. (b) In this case, if a single link exists for several time steps, it is visually connected through these time steps. The result is a visualization similar to Gantt charts, displaying the time spans of the different links.

Section 2.4.2.2. As already mentioned, the size of the individual rows can vary. Therefore, it is possible to vertically stack all links of an individual node inside its respective row (Figure 2.17(a)). Drawing the time steps next to each other, time spans are represented by bars that result from the merging of neighboring node elements (Figure 2.17(b)). Time is represented horizontally from the center to the left (incoming links) and to the right (outgoing links).

With this visualization scheme, time spans can be easily analyzed even in graphs with many time steps and it is possible to detect temporal clusters, i.e., links with identical time spans (see Figure 2.20(b)). The layout scales well with the time discretization. It does not even require time discretization; it is possible to represent time spans on a continuous time scale. Another interesting property is that the height of a row corresponds to the number of links summed up over time. Hence, it can be seen which nodes have a high number of links over the full time range of the data.

However, this representation requires usually more rows, depending on the number of links per node. It is therefore only suitable for sparse graphs like the visualization workflow shown in Section 2.5.4. There, the space requirements change like this: The normal layout requires 61 rows (one for every node) and 529 columns for incoming and outgoing links. Most of the time steps require more than one column because at least one node exhibits multiple links. The Gantt layout requires 121 rows because the nodes with multiple links require

multiple rows. However, only 252 columns are required, twice the number of time steps because incoming and outgoing links are shown. Furthermore, in the case of the normal layout, some extra space in the columns is used to visually separate the individual time steps. Hence, for this dynamic graph, the Gantt layout requires twice the space in vertical direction but less than half of the space in horizontal direction (compare Figures 2.26 and 2.27).

2.4.4 Discussion

The following discussion is based on the results of a user study that was conducted to evaluate visual adjacency lists (details can be found in the original paper [2]), and general criteria like scalability, compactness, and visual clutter [41]. Table 2.1 summarizes the discussion for key aspects. Furthermore, typical patterns in visual adjacency lists are discussed.

2.4.4.1 Advantages

First, even though color coding for node correspondence is perceptually less accurate than geometric visual mapping by lines (i.e., links in node-link diagrams), some tasks benefit from the asymmetric mapping of links. This is the case for tasks with asymmetric characteristics, i.e., tasks where it is not important to identify target nodes. For example, the user study [2] included a task in which the distribution of incoming and outgoing links of all nodes had to be compared.

Second, like adjacency matrices, the proposed visualization is not cluttered: there are no overlapping visual elements. Generating the layout does require neither high computational effort nor complex algorithms. This can be the case for other graph visualization techniques such as node-link diagrams.

Table 2.1 — Comparison of graph representations.

Aspects	Adj. List	Adj. Matrix	Node-Link
asymmetric tasks	+	o	-
clutter-free	+	+	-
space-efficient	+	o	o
multigraphs	+	o	o
cluster detection	o	+	+
following paths	-	o	+
dense graphs	-	+	-

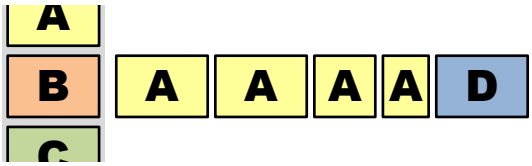


Figure 2.18 — Handling of multigraphs. The method can be directly applied to multigraphs: multiple links (links from “B" to “A" in this example) appear with multiple node elements.

Furthermore, the asymmetric mapping of links often results in a more space-efficient visual representation than adjacency matrices. Especially in the case of sparse graphs, visual adjacency lists yield a very compact visualization (see Section 2.5.4). This does not only improve the scalability but also allows a faster recognition of certain aspects of the graph, e.g., which nodes have the largest or smallest number of links. The reduced space requirements are also advantageous for the visualization of dynamic graphs.

The presented approach can directly handle multiple occurrences of the same link (multigraphs, see Figure 2.18). Node-link diagrams and adjacency matrices have to be extended for this case.

Visual adjacency lists can be easily transformed into an adjacency matrix visualization and vice versa (see Figure 2.10). When both methods are used, the transition between them can be shown, e.g., by smooth animation. This may provide further insight or support the understanding of the visualization.

Some of the properties of visual adjacency lists can be derived from the underlying concept of adjacency lists as data structures. The complexity of graph-related tasks [178] can be derived to some extent from the computational complexity of adjacency lists as memory layout. The complexity of topology-based tasks related to direct connections is linear in the number of links of the respective nodes. In the case of adjacency matrices, the complexity is linear in the number of nodes, i.e., the maximum number of possible links of a single node. The same holds for attribute-based tasks related to links. The complexity of attribute-based tasks related to nodes is linear in the number of nodes for both adjacency list and matrix.

2.4.4.2 Disadvantages

Visual adjacency lists also exhibit several drawbacks. While some tasks benefit from the asymmetric mapping of links, other tasks become more difficult.

The detection of clusters (see Figure 2.19(a)) and other graph structures can be difficult. Especially in large graphs, the color coding may hinder the recognition of nearby clusters and small structures (see Figure 2.13). One approach for reducing this problem may be a hybrid representation exploiting both concepts,

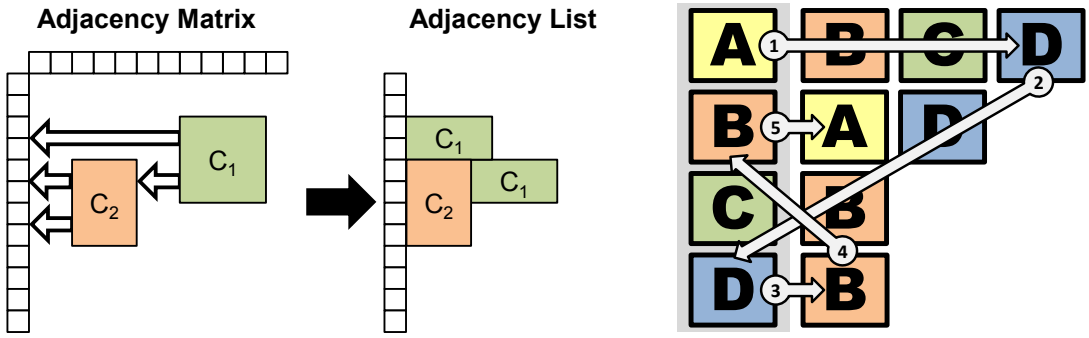


Figure 2.19 — Issues with cluster detection and following paths. (a) The clusters C_1 and C_2 are both clearly visible in the adjacency matrix (left). In the corresponding adjacency list (right, see Figure 2.10), only C_2 remains clearly visible. Cluster C_1 is distorted because the lower part is "blocked" by C_2. (b) Following a path in the adjacency list requires one to jump between the link lists and the corresponding positions on the node axis. In this example, a path from "A" to "A" over "D" and "B" is followed.

adjacency lists and matrices (see Figure 2.10). For example, clusters can be shifted closer to the node axis as long as they do not break up.

Browsing tasks or topology-based tasks related to indirect connections [178] that require one to follow paths are time-consuming (see Figure 2.19(b)). The user study showed [2] that tasks on direct connections can already be quite time-consuming. One cannot directly go to a connected node, as it is the case in node-link diagrams; the node axis has to be scanned for the respective entry. Following several links requires one to jump between the node axis and the link list back and forth. There are similar problems in visualizations with adjacency matrices [114]. The usage of adequate interaction techniques can improve the handling of paths, e.g., by highlighting possible paths after selecting a specific node.

In the case of large graphs, adjacency lists provide only an overview of large-scale structures (see Figure 2.12); small-scale details are hardly visible. Aggregation (see Figure 2.14) and interaction techniques can alleviate these issues.

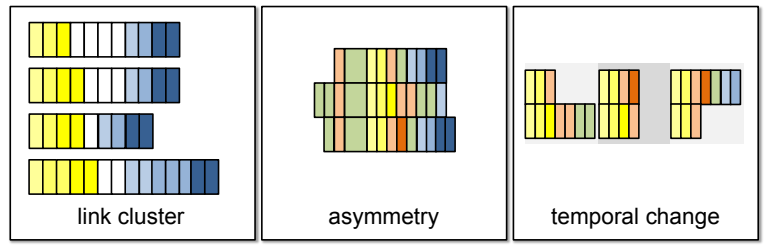
Similar to adjacency matrices, the ordering inside the node and link axes is flexible and affects the result. Adjacency lists are even less restrictive than matrices: the ordering along the link axis does not have to be identical in each row. This flexibility allows a better adaptation of the visualization to the tasks and data (Section 2.4.2.2). However, more parameter adjustment may be required to create good visualizations.

2.4.4.3 Visual Patterns

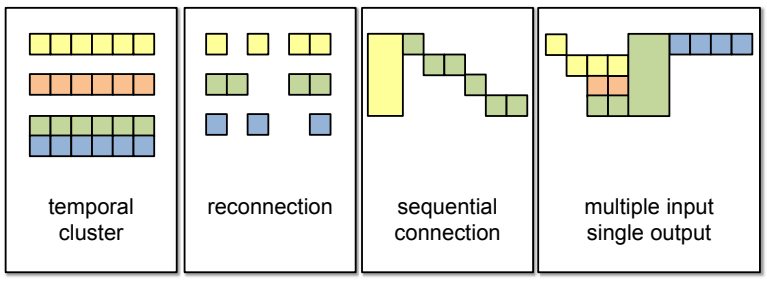
Analyzing graphs requires the detection of visual patterns and signatures. Besides clusters in general [178], typical patterns in dynamic data include trends, periodicities, shifts, and anomalies in the graph structure over time [60].

Figure 2.20 shows some of the important patterns that can occur in visual adjacency lists. With the normal layout (Figure 2.20(a)), the following patterns can be observed: Clusters are visible as patterns of nearby colors of the color map (see also Figure 2.13). Asymmetries with respect to incoming and outgoing links can appear in the link distribution. Finally, links can vary over time in dynamic graphs.

The Gantt layout (Figure 2.20(b)) emphasizes temporal dynamics of the graph. Therefore, most patterns result from changes in the graph. Links with equal time spans create a temporal cluster. Links can recur after they disappeared and a single node can also have a sequence of links to different nodes. Finally, the ratio between incoming and outgoing links (or vice versa) of a node can be large, e.g., a single node may have multiple incoming links but only a single outgoing link.



(a)



(b)

Figure 2.20 — Typical patterns in visual adjacency lists with (a) normal layout and (b) Gantt layout.

2.5 Case Study: Modular Visualization

For this case study, the presented approaches (Sections 2.2 and 2.4) were implemented to visualize and analyze the evolution of module workflows from VisTrails [64]. VisTrails is a visualization framework that allows one to combine different modules, e.g., from the visualization toolkit VTK [248], to create a custom visualization. The module workflow is also referred to as *pipeline* in VisTrails and can be created in a graphical way (see Figure 2.1). VisTrails was specifically designed to collect and provide provenance information [253], i.e., all user actions are logged. Hence, it is easy to extract the evolution of the module workflow from a VisTrails dataset.

Since visual adjacency lists (Section 2.4) are also suitable for other types of graphs like weighted graphs [2], they were implemented in a separate tool to allow a better exploration of their feature space. Both prototype tools were implemented in C++ with OpenGL for the graphics part. Since the visualizations are not computationally intensive, no special optimizations or implementations, e.g., for GPUs, were required to provide interactivity.

In this section, three different application scenarios are discussed. In the first case, the visualization of the module workflow evolution is used to identify potential bottlenecks and barriers for users of VisTrails. The second one aims at aiding in the retrospective analysis of visualization sessions with the same visualization approach. The last one demonstrates what information can be extracted from the change of connectivity during module workflow modification and uses visual adjacency lists.

Three different types of datasets were used: First, visualization experts provided several VisTrails files with the provenance of their visualization sessions. The second type of dataset contains the provenance of student assignments in a visualization course, which were obtained from Silva et al. [254]. The examples provided with VisTrails are the third type of data; the visualization of a brain scan (brain_vistrail.vt) is used as an example. For the last application scenario, a dynamic graph was extracted from one of the student datasets. This example is a sparse graph with 61 nodes, 33 links on average per time step (2 links minimum, 53 links maximum), and 126 time steps. Every time step represents a modification of the workflow by the user. In this case, the user created and modified a combined visualization of a 3D scalar and a 3D vector field.

Before the application scenarios are presented, a typical example of how the visualization approaches can be used for analysis is described briefly in the following.

2.5.1 A Usage Session

In typical cases, it is reasonable to start the analysis with the module and event visualization (Section 2.2). This provides already an overview of the modules and events, and their temporal relation in the evolution of the module workflow.

After loading the dataset, an initial look is taken at the event view to obtain a first overview of the data. If there are not too many events in the dataset, interesting details can be already recognized: If the focus lies on the visualization methods used, the "mapping" category is expanded. By filtering events, e.g., changes of the module position in the editor—these are usually not of interest, visual load is substantially reduced.

To analyze the data in detail, especially if it covers a long time range, looking at the branch view helps to select interesting parts of it. For example, branches with no or only one parent might be of interest (visible as white columns or columns with a single colored block inside, see Figure 2.5) because they indicate that the user started something new or continued from an earlier stage. If the number of modules is high, it may help to reduce it by collapsing categories or removing uninteresting modules. After selecting a branch, with or without all related parents, the workflow history can be browsed by moving the selection area and the individual steps of the visualization session can be followed. It is often interesting to look at parameter changes. For this, the functions of interesting modules are expanded. The plot and histogram view might be used to follow the parameter changes.

If the focus of the analysis lies on the dataflow and relations between modules, the graph visualization based on visual adjacency lists can be used to obtain a detailed view on the connections between modules and how they change over time. Especially the Gantt layout is suitable for this as demonstrated in Section 2.5.4.

2.5.2 Usability Issues

In this scenario, the student and expert datasets were explored at a coarse level. Anomalies and unexpected patterns are mainly of interest. Every dataset was browsed rather fast, for no longer than 5 minutes per dataset. However, even this relatively rough exploration yielded some interesting discoveries.

Figure 2.21 shows a part of an expert dataset. In this example, a large number of modules were pasted, deleted, and afterward pasted again. This is visible through bars starting at the same time and also through the aligned symbols for module creation. It was presumably easier for the user to delete all modules

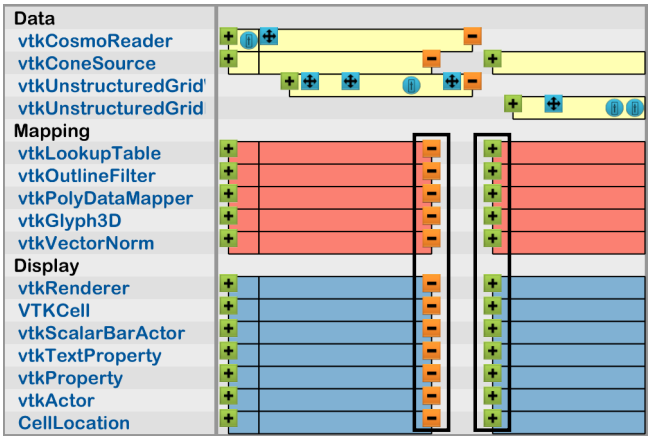


Figure 2.21 — Two branches of an expert dataset showing redo behavior, i.e., modules are deleted and then inserted again (marked black).

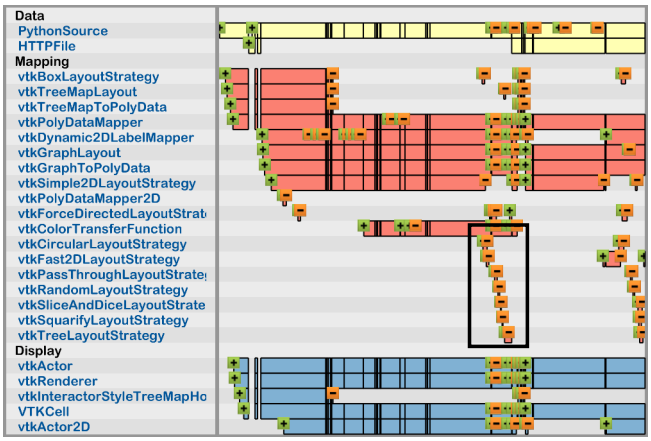


Figure 2.22 — Student dataset exhibiting many modules with very short lifetime (marked black). The module names reveal that these modules are used to apply different layout strategies.

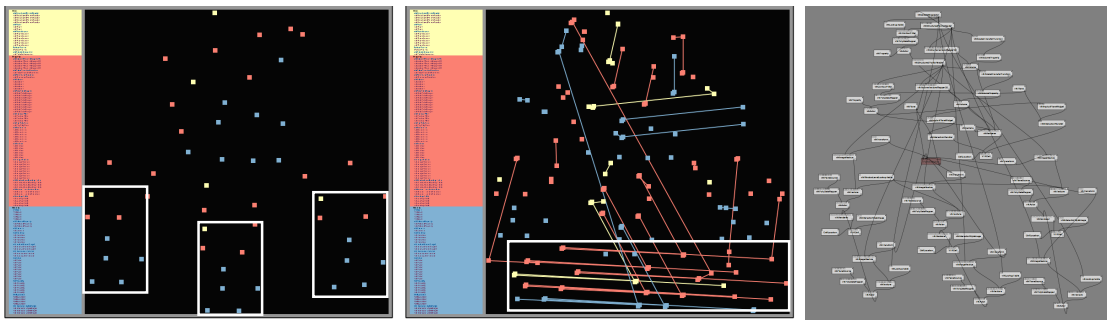
and paste them again instead of using the undo capabilities of VisTrails to return to a previous state.

The second example (Figure 2.22) shows many modules that exist only for a short period of time. One interpretation for this could be that the user tried different modules and was not able to use them for the task. However, looking at the module names reveals that every module provides a different layout method for graph or tree visualizations. Hence, the user has to repeatedly create and connect modules to test different layouts.

In the third case (Figure 2.23), the positions that the modules have in the workflow editor of VisTrails were analyzed. In the beginning, a large group of modules was pasted. Their positions are visible in Figure 2.23(a). There seems to be no overall structure in their placement, the color coding reveals that modules of all categories are mixed. Looking at the full time range of the visualization session (Figure 2.23(b)), additional modules appeared and some of them were moved. Still, no clear structure is visible in this workflow with a relatively large number of modules (Figure 2.23(c)). Hence, finding specific modules for modifications can be difficult. Of course, it is very subjective what

is perceived as a good structure. Nevertheless, this result can be an indication that the user should be better supported by the system in keeping the workflow structured.

In addition to the presented observations, a significant number of paste operations were noticed during the explorations. This seems to be independent from the experience of the user. Both, experts and students, pasted groups of modules many times.

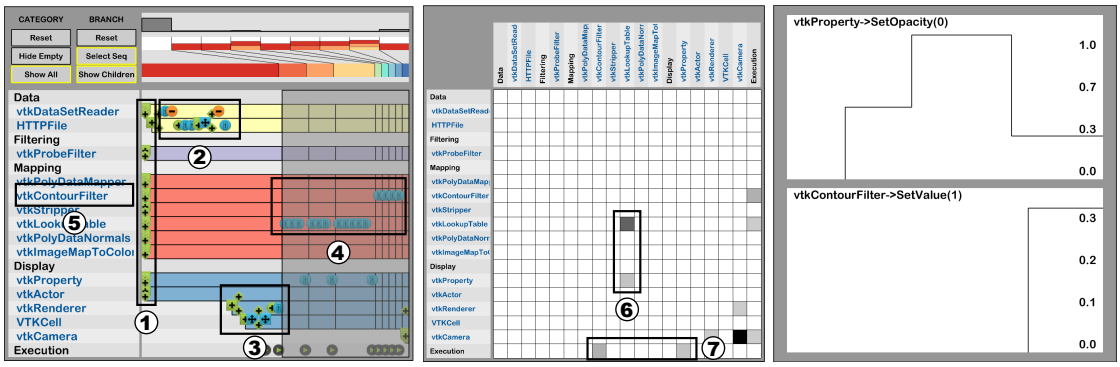


(a) (b) (c)

Figure 2.23 — Analysis of module locations in the expert dataset. (a) Some patterns can be seen in the lower part of the location plot for their initial creation. Three similar groups occur there (marked white). However, an overall structure where the modules are grouped according to their category cannot be seen. (b) The location plot for the full time range confirms this. Only some modules were moved in groups, visible in the parallel lines of their movement (marked white). (c) The complexity is also visible in the pipeline view of VisTrails.

2.5.3 Retrospective Analysis

In this scenario, a detailed analysis of a single visualization session is conducted to understand how the visualization workflow was created. Figure 2.24 shows images from the exploration of the workflow for the brain dataset. Interesting areas are marked with numbers and described in the following. The overview (Figure 2.24(a)) shows that the example was not built from scratch, but most of the modules were initially pasted including their connections (1). The creation of the visualization seems to be well structured. First, data sources were set up (2), then, the display modules were connected (3). Subsequent branches deal mainly with parameter changes of the visualization (4). Users familiar with VTK can also see that only one visualization method is used, the extraction and visualization of isosurfaces (“vtkContourFilter” module, 5). The other mapping modules extend this visualization only, e.g., by providing color mapping.



(a) (b) (c)

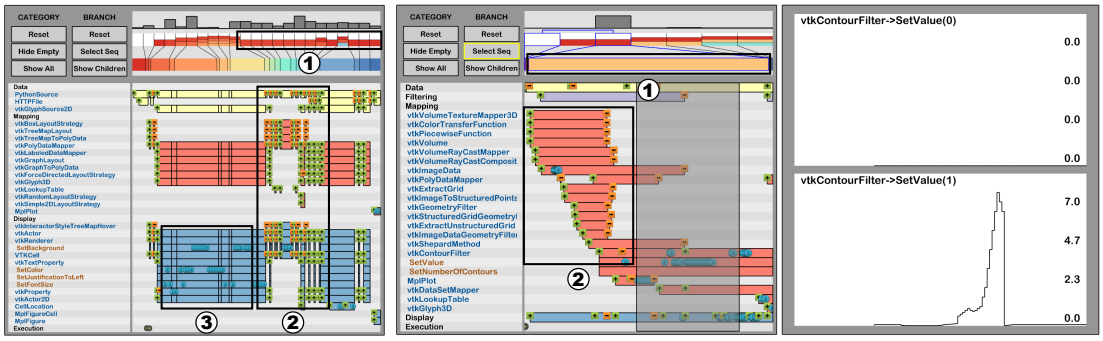
Figure 2.24 — Reconstructing the visualization of the brain dataset. (a) The overview reveals already the basic structure of the way the visualization was created. Because of the low number of different modules and events, many details can be seen without filtering. (b) The transition matrix shows typical sequences of events. (c) The parameter plot shows how the values of different module parameters were changed. Both additional views were computed for the selected time range in (a) (marked gray) covering the second half of the data. Areas of interest are marked with a black frame and numbers.

Furthermore, patterns in the parameter changes are visible in the second half of the time range. Changes of color and opacity were performed ("vtkLookupTable" and "vtkProperty"). This sequence of events affecting both modules is also visible in the transition matrix representation (Figure 2.24(b), 6). It can also be seen in the matrix that the workflow was executed after changes on two different modules (7). The parameter plot (Figure 2.24(c)) shows how the parameters were changed over time, e.g., different opacity values between 0.3 and 1.0 were used. Figure 2.24(a) shows that the isovalue was changed several times ("vtkContourFilter") at the end of the time range. However, the parameter plot (Figure 2.24(c)) shows that it was always set to the same value around 0.3. An explanation for this behavior is that the parameter changes occur in different branches of the workflow, visible through the vertical lines in the module and event view (Figure 2.24(a)). Hence, the visualizations generated in the different branches were possibly adapted so that they show the same isosurface.

The next example (Figure 2.25(a)) exhibits interesting patterns in the branch view. There are many branches with only a single parent at the second half of the time range (1): the work was restarted several times from an early stage. Looking at the events reveals additional interesting details, especially, when only events for adding and deleting modules, and changing parameters are displayed. Beside the fact that the student used many pasting operations (2),

it is interesting that parameter changes occur mainly at the beginning of the visualization session (3). The function names affected by changes show that the student spent most of the time changing the background color or the used font. In some cases, this may be enough to create a good visualization. However, together with the repeated pasting of large module groups, this can also be an indication of cheating, i.e., parts were copied from workflows of other students.

The student in the last example (Figure 2.25(b)) worked mainly in a linear way without branching (1). It can be seen that he tried out different visualization techniques (2): he first used volume rendering techniques ("vtkVolume..." modules) and then isosurface extraction ("vtkContourFilter"). Looking at the parameter plot for the "SetValue" function of the isosurface module reveals that the student put considerable effort to find a suitable isovalue. However, he used the initial value around zero at the end. This might indicate that the student possibly failed in finding a suitable isovalue for his visualization.



(a) (b)

Figure 2.25 — Two examples from the student data. (a) This usage session exhibits many branches connected to the two initial branches. (b) The other student worked mainly in a single branch, in which he tried out several methods. Areas of interest are marked with black frames and numbers.

2.5.4 Dataflow Graph

Analyzing the link structure of a module workflow, which represents data flow in this context, provides insight into user behavior and the architecture of the underlying system. Persistent links, e.g., build the backbone of the custom visualization and temporal clusters indicate a direct relation of links and the respective modules. Trial and error behavior of the user typically results in shortly existing links and applying the visualization to different data sources creates a sequential link pattern.

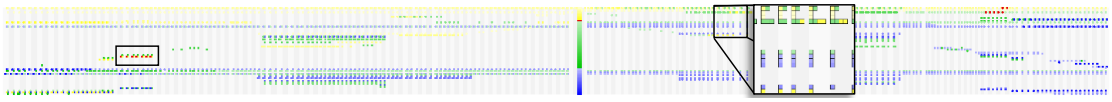


Figure 2.26 — Visual adjacency list for the respective student dataset. Node order and color coding consider the visualization pipeline (see Figure 2.27(b)). An example area is shown in an enlarged display.

Applying the visual adjacency list with normal layout to one of the student datasets results in a visualization with large horizontal extent due to the large number of time steps (Figure 2.26). Therefore, displaying the entire dataset at once provides only an overview of the graph and its connectivity. For example, it can be seen that this is a very sparse graph; most modules have only one or two connections. Patterns and outliers can be detected, e.g., the filtering module (red blocks in the visualization) is connected to a single module for a rather short time range only (marked in the image). However, analyzing individual time steps is difficult. This requires zooming-in and allows seeing, e.g., the minimum and maximum number of links as shown in the enlarged area. Still, it is difficult to determine the time spans of links because they are not aligned and can only be separated by their color.

The Gantt layout results in a more compact visualization (Figure 2.27). It does not only provide an overview of the entire dataset but also shows details without zooming, e.g., temporal clusters and outliers can be detected. Furthermore, the time spans of links and the temporal evolution of the connectivity can be analyzed. Among other things, the following interesting areas can be seen marked in the visualization:

1. The sequential connection pattern. The user probably tried out different modules as data source for this module.

2. These links exist only for a few time steps and the reconnection of a single link is visible. This can indicate trial-and-error behavior of the user.

3. A sequence of connections and partly multiple inputs, while there is only a single outgoing link for almost the full time range. Hence, this module processes data from changing sources but always provides the results to the same module. The module seems to be important for the basic visualization setup.

4. Two temporal clusters; the upper cluster consists only of links from the category “display". These modifications of the visualization persist for longer time spans and involve several modules.

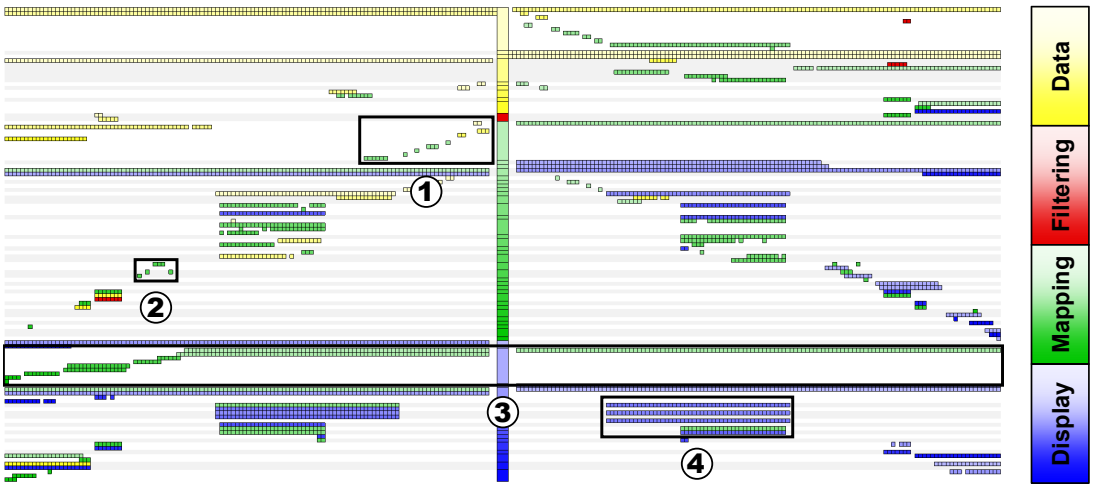


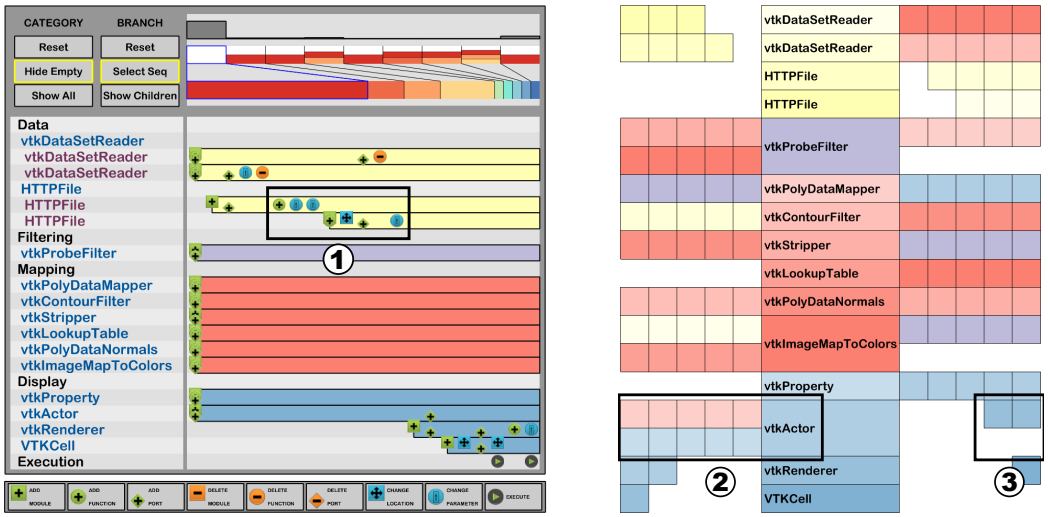
Figure 2.27 — Visual adjacency list with Gantt layout for the respective student dataset. (a) The full dataset is visualized. Areas of interest are marked with numbers and black frames. (b) The nodes are ordered and color coded with respect to their classification in the visualization pipeline by Haber and McNabb [125]. Nodes of the same classification are ordered according to the time the respective modules are created and the brightness of their color is changed to visually separate them.

In summary, visual adjacency lists with normal layout can provide an overview of the change of connectivity over time, but a detailed analysis requires zooming. The Gantt layout has a time-aligned representation of links and therefore improves the analysis of time spans and the detection of temporal clusters.

2.5.5 Comparison

The previous examples already showed that the two presented approaches are designed to analyze different aspects of a module workflow and its evolution. This is discussed in the following in more detail. Figure 2.28 shows the proposed module and event view (Figure 2.28(a)) and the visual adjacency list with Gantt layout (Figure 2.28(b)) for the same module workflow history, in this case the brain dataset from the VisTrails examples.

The module and event view provides an overview of the user actions by displaying the respective events. For example, it is easy to see when the user changed parameters of some of the modules or when the module workflow was executed. This allows, amongst others, the recapitulation of a previous session (Section 2.5.3) or the analysis of user behavior (Section 2.5.2). However, it is



(a) (b)

Figure 2.28 — Comparison of the module and event view with the visual adjacency list with Gantt layout. (a) The module and event view shows the time spans of modules and the events related to them. (b) The visual adjacency list with Gantt layout represents the evolution of the module connectivity. It considers only events related to the connectivity. The color coding of the visual adjacency list was adapted to the one of the modules and event view. Areas of interest are marked with a black frame and numbers.

hard to see the connectivity information although events related to the creation or deletion of connections between modules are displayed.

The visual adjacency list with Gantt layout is more suitable to analyze the connectivity of the module workflow and its change over time (Section 2.5.4). It shows the duration of links and when they are created. The representation with bars allows an easy comparison of the durations of different links. However, this visualization represents only the connectivity information. Other events like the change of parameters are not visible.

Therefore, both visualization approaches complement each other. In this example (Figure 2.28), it can be seen in the module and event view, for instance, that parameters of the "HTTPFile" modules were changed (1). This is not visible in the visual adjacency list. However, the visual adjacency list shows that the "vtkActor" module has two input connections from different module categories ("Mapping" and "Display"), which persist over the full time range (2). Furthermore, an output connection is later created for this module (3). It is hard to extract this information from the module and event view.

2.5.6 Summary

Some general observations can be derived from the presented examples. Looking at the way users create the workflows might help improve the usability of the workflow system (Section 2.5.2). For example, the explored data showed that pasting a group of modules is often used. The large number of layout events in the third example suggests that a useful improvement would be to support a better layout mechanism when users add and connect modules, e.g., the use of an automatic layout algorithm. This could support the understanding of the used workflow and help modify it.

There are various applications for a detailed retrospective analysis of user sessions (Section 2.5.3). One aspect is to understand or recall previous steps when continuing the work with a module workflow. This could be done for one's own work, but it could also help in collaborative projects understanding the work of the collaboration partners. The workflow history also exposes what methods were used and which parameters were changed to get the final result. It might help characterize sessions (or parts of sessions) with respect to the nature of the actions performed, for example, phases of the session that consisted mostly of parameter tweaking as opposed to workflow construction. Finally, a detailed exploration can point to characteristical usage patterns or commonly used parameters. This information may help improve the handling of the workflow system, e.g., by offering commonly used parameters in a special menu or widget.

Looking at the connections of the module workflow or rather the respective data flow graph (Section 2.5.4) provides a view on the interplay of the used modules and therefore information about the implementation. It can be seen, e.g., which modules have only output connections, i.e., they are a source for the data processed by the workflow. In contrast, modules at the end of the workflow have only input connections. In the case of visualization software, they are typically responsible for displaying the result on the screen or storing it in a file. Nevertheless, information about the user behavior can also be extracted from the change of module connections. A central data processing element in the workflow might have persistent output connections, but its input connections vary a lot because the user tries out different modules. In general, trial-and-error behavior can be visible in links with comparably short duration.

Part II

From Data to Images

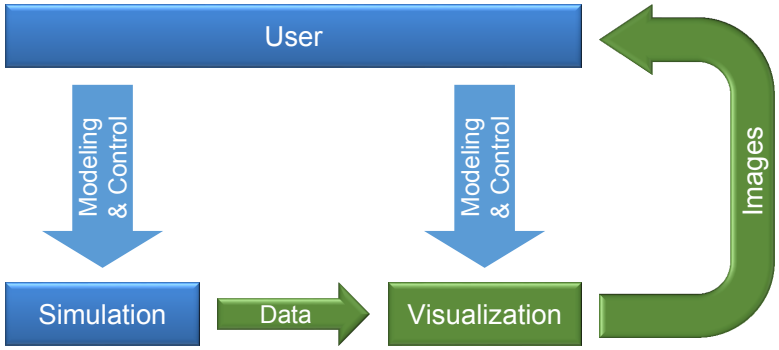


Figure II — Simulation and analysis process: visualization generates a visual representation of data resulting from the simulation.

The previous part of this thesis discusses visual support for modeling and controlling simulations and visualization methods, i.e., the respective workflow defining the interplay between the required modules was visualized. Assuming now a simulation run finished successfully, the next phase in the work with simulations starts: the resulting data must be analyzed (Figure II), which typically requires more than just looking at numbers. Hence, in the context of simulations, the key task for visualization is to enable or support the analysis of the simulation results, which is considered in the following part of the thesis: how to get from simulation data to meaningful images?

In general, visualization generates a visual representation of data, which can then be explored and analyzed by the user. Developing an effective visual representation requires us to consider the data to be visualized: a visualization of vector data typically uses different visual mappings than a visualization of scalar data. There are also different requirements for 2D and 3D domains, and time-dependent data of different dimensionality. Furthermore, the design of a visual representation should also consider the application respectively the problem under investigation. An effective visualization shows for a given problem only relevant information of the data.

In this part of the thesis, several novel visualization methods are introduced. They cover typical classes of data generated by simulations and are ordered with increasing dimensionality respectively complexity of the data. First, scalar data is considered (Chapter 3). While this is the most basic type of data and the visualization of scalar data has been under research for decades [132], there are still unsolved issues. One of them is the visualization of scalar data covering a large value range. This issue is discussed for the example of bar charts and an extension of them better representing large value ranges is presented.

After that, several methods for vector field and flow visualization are presented (Chapter 4). In particular, the visualization of time-dependent vector fields remains an important research topic with a broad spectrum of applications. Incorporating the temporal dimension in the visualization is often challenging. While many approaches use animation for this task, the presented methods provide a static representation of time-dependent vector fields respectively unsteady flow. Two methods complement each other: the first method provides an Eulerian view, whereas the second method provides a Lagrangian view of unsteady flow. Finally, a computation scheme for integral curves is presented in this chapter that can be used to accelerate different visualization methods.

The last chapter in this part of the thesis deals with tensor fields (Chapter 5). Tensors are used, e.g., to model diffusion processes or mechanical stress, and their high dimensionality leads to challenging visualization problems. The presented technique deals with symmetric second-order tensor fields and allows segmenting them into regions of coherent behavior.

CHAPTER 3

Scalar Data with Large Value Range

In scalar data, every data element represents only a single value—the respective scalar. Many phenomena in science can be described with scalar data. For example, a simulation of drug infusion processes in the human brain provides scalar fields representing the drug concentration [286]. An open issue is the visualization of scalar data covering a large value range. Such data can occur in a variety of applications. Examples include the luminosity of stars in astrophysics, the half life of isotopes, or the population of different countries. In such cases, it is challenging to create appropriate visualizations. Without interaction, only a limited resolution is available for representing the values. This makes it difficult to accurately read small values in the presence of large ones.

This chapter[1] presents a visualization approach that does not exhibit the problems of non-linear and non-proportional mappings but still allows small values to be readable. While the underlying concepts are applicable to a broader spectrum of representations, the approach was developed and is demonstrated for one of the most basic representations for scalar data: bar charts. Bar charts are an established tool to visualize quantitative data. They can be read in an intuitive and accurate way due to their encoding of quantities in the visual variables length and position [75]. Consequently, bar chart-based diagrams are a powerful and commonly used tool to communicate a set of quantities.

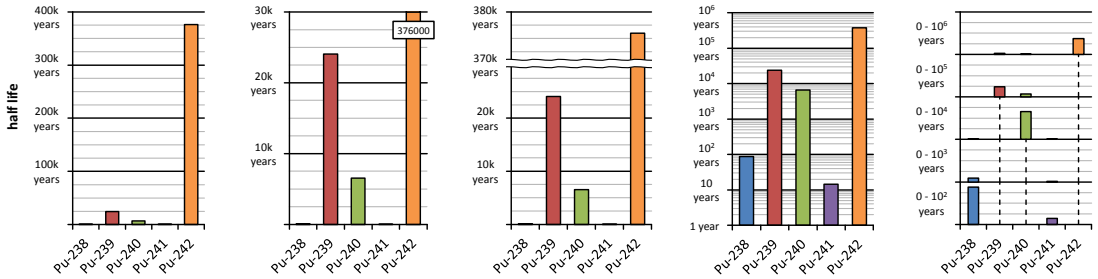
[1] **Parts of this chapter have been published in:**
M. Hlawatsch, F. Sadlo, M. Burch, and D. Weiskopf. Scale-stack bar charts. *Computer Graphics Forum*, 32(3):181–190, 2013 [3], © 2013 John Wiley and Sons.

3.1 Related Work

There are several approaches to address the problems of linear bar charts (Figure 3.1(a)) when displaying large value ranges:

- **Cut-off bars:** The bars corresponding to large values are cut off and labeled with their value (Figure 3.1(b)). This allows visual comparison of small values but not of the values exceeding the scale range.
- **Scale break:** Another approach to bridge the gap between small and very large values uses a scale break [67] (Figure 3.1(c)). This technique inserts a gap in the scale to provide more space for the representation of small values. Here, a visual comparison across the break is not possible, and only the heights of the bars below the scale break are proportional to the respective values.
- **Logarithmic mapping:** Values are mapped with a logarithmic scale (Figure 3.1(d)). This approach has the benefit that small values are still readable. In addition, it is a reasonable approach for data representing exponential processes, like growing populations. However, the non-linear mapping makes a quantitative comparison of values more difficult.

Unfortunately, the above-mentioned variants of bar charts introduce a lie factor [267]. A lie factor exists if the size of an effect in the data and in the respective visualization is not equal. The variants all have problems with respect to the quantitative comparison of values, which is one of the most important and best performing tasks when using bar charts with linear scale. Furthermore,



(a) (b) (c) (d) (e)

Figure 3.1 — Different bar charts for visualizing data with large value range. Half lives of different plutonium isotopes (see Section 3.4.1) are visualized with: (a) classic linear bar chart, (b) linear bar chart with cut-off bars, (c) linear bar chart with scale break, (d) logarithmic bar chart, and (e) scale-stack bar chart.

applying a logarithmic mapping, e.g., to a stacked bar chart results in an even more misleading visualization scenario.

There are further ways to modify bar charts. For example, the value axis does not have to be zero-based but can start at a different value. However, such modifications cannot generally solve the problems with large value ranges.

In this chapter, an extension of bar charts with different scales depending on the exponent under observation is presented. With this approach, comparisons between small values can be carried out accurately, and at the same time, larger values can be compared among each other and in the context of the smaller values. Furthermore, the proposed charts are also free of chart junk [39, 258], which refers to the fact that data is visually encoded in several redundant visual features.

This chapter focuses on bar charts, as they are a common tool and widely used, but there are of course other concepts for visualizing quantities. Many of them and general discussions of data visualization and statistical graphics can be found in Cleveland and McGill [74], Huff [143], Card et al. [65], and Ware [290]. Few [102, 103] illustrates drawbacks of radial pie charts, while in the study of Cleveland and McGill [75], the performance of several charts was measured. Also, the work of Goldberg and Helfman [119], who investigated the readability of values on linear versus radial graphs by applying eye tracking techniques, showed that choosing a non-radial diagram is beneficial with respect to user performance for many tasks. Further discussions of radial methods can be found in Draper et al. [94] and Diehl et al. [90]. Product plots [304] are a general framework for statistical graphics that can also handle bar charts. Finally, it should be noted that even well-established and rather simple techniques like bar charts still have space for improvements as the work by Talbot et al. [259] shows in the context of positioning tick labels on chart axes.

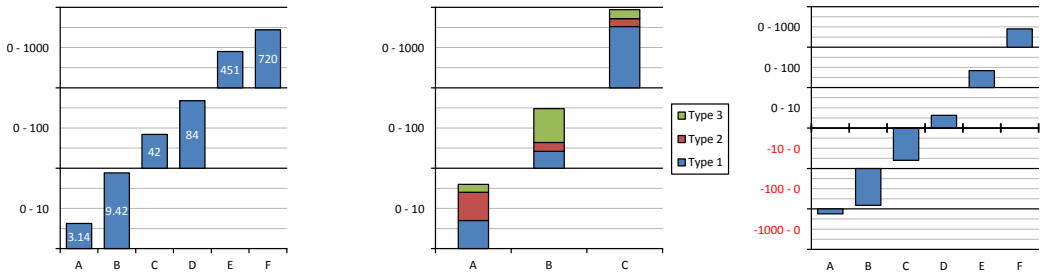
3.2 Scale-Stack Bar Charts

The approach is inspired by the scientific notation of numbers and the way floating-point values are commonly stored in computers. The values are split into two parts: their order of magnitude and their representation in this order of magnitude, similar to the exponent and mantissa in scientific notation. The result is a two-dimensional representation of numbers. There is a multitude of different two-dimensional mappings of values to visual features, even when only the bar chart metaphor is considered, e.g., different spatial dimensions or color can be used. Many different designs were tested for this work in a formative process. This resulted in *scale-stack bar charts*, which are described in the following.

3.2.1 Basic Method

The value axis of the chart is subdivided to represent different scales respectively orders of magnitude (Figure 3.2(a)). This is called a *scale-stack*. The important thing is that inside each scale, a linear mapping of the values is used. Furthermore, every scale starts again at zero. This approach preserves the advantage of linear bar charts: easy comparison of values. In summary, a two-level hierarchical representation of exponent and mantissa is used, which is possible due to the discrete nature of the exponent in the representation.

This layout has several advantages. First, both components—mantissa and exponent—can be directly compared. This would not be the case with other mappings, e.g., if two different spatial dimensions would be used. Second,



(a) (b) (c)

Figure 3.2 — Illustration of the concept. (a) Every row (delimited by bold lines) covers a complete scale; the bars always start at zero in every row. Inside each scale, a linear mapping is used. (b) Because of this mapping, the method can be used to display stacked bars. (c) For negative values, the chart is extended downward as it is also a common approach with classic bar charts.

using a different row for every scale lowers the risk of directly comparing values of different scales. Finally, the usage of length for number representation allows accurate quantification and comparison [75].

The proposed approach augments the bar chart metaphor only in the value dimension. Hence, there are only constraints in this dimension. Other dimensions and layout aspects remain as flexible as in classic bar charts. For example, the color coding of the bars as well as their sorting and shape can be adapted according to the respective needs. Since a linear mapping is used within each scale, typical variations of classic bar charts can also be used, e.g., stacked bars (Figure 3.2(b)). Furthermore, it is straightforward to display negative numbers (Figure 3.2(c)). The method is demonstrated with vertical bar charts in the following. However, the application to horizontal bar charts is straightforward and therefore not further discussed.

It is clear from the concept that the ranges of the different scales have a strong influence on the resulting visualization, as it is also the case for classic linear bar charts. If a range is too large, it is difficult or not possible to read small values. If a range is too small, only few values are captured. Because there are multiple scales, some effort is required when manually choosing a set of appropriate ranges. Therefore, a simple approach to automatically adapt the scale ranges to the data is described in the next section.

3.2.2 Automatic Scale Selection

The algorithm for automatically choosing the ranges for the different scales requires the data values for the chart and the desired number of scales as a user parameter. The problem of choosing an adequate number of scales is excluded from the algorithm to keep it simple. In general, the problem of choosing adequate scale ranges can be seen as an optimization problem. The goal is to maximize the size of the individual bars for better readability.

The height h of a single bar for a value v inside a specific range $[r_{\text{min}}, r_{\text{max}}]$ can be determined with $h = v/r_{\text{max}}$, because $r_{\text{min}} = 0$ in the proposed approach. For simplicity, the approach is described for positive v and r_{max}. It can be applied to negative values in a separate pass by using their absolute values. The algorithm assigns values to scales s for which $r_{\text{max}} - v$ is minimal and positive. To ensure that the smallest assigned values inside a scale are still visible, their height is maximized by maximizing the following objective function:

$$f = \sum_{s \in S} \min_{v \in s}(v) / \max(s),$$

where S is the set of all scales in the chart and $\max(s) := r_{\text{max}}$ of scale s.

A greedy approach is used for optimization. It is started with n scales, where n is the number of data values in the chart, i.e., every value has initially its own scale with $r_{\max} = v$. In every iteration step, two scales s' and s'' are merged to a single scale with $r_{\max} = \max(\max(s'), \max(s''))$ containing all values of both scales. To choose the best pair of scales for merging, every possible pair is temporally merged. The pair that maximizes the objective function is then finally used for merging. This is done until the desired number of scales is reached. Operating on sorted values eases the implementation and may improve the efficiency because only neighboring scales have to be considered for merging. As it is the common way for tick labels in charts, each $r_{\max}$ can be rounded, e.g., to the next order of magnitude.

This approach does not always find the globally optimal solution, but it is simple, fast, and provided good results during the experiments for this work. An improved algorithm and an approach to automatically select the number of scales exceed the scope of this work and will be part of future work.

3.2.3 Variants

Even with well-adapted scale ranges, there can be the issue that nearby values may be mapped to different scales (Figure 3.3(a)). This makes it difficult to compare them. To overcome this issue, values can be displayed on all scales with a proper range (Figure 3.3(b)), i.e., where the upper limit of the range is larger than the value. Nearby values can now be compared on the respective

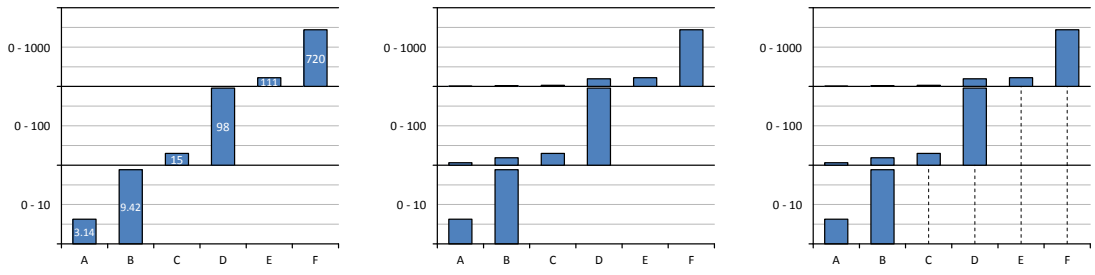


Figure 3.3 — Variations of the method. (a) Nearby values may be mapped to different scales (e.g., B and C). This makes their comparison difficult. (b) Displaying the values on multiple scales avoids this problem. The values of B and C can now be compared on the second scale. In this case, the values are shown on all scales that cover the full value range. (c) Dashed lines can be used as placeholders to show that a value exceeds the range of the respective scale.

scale. A problem of this solution is that it is not clear if a column is empty because the value is too small to be visible or because it exceeds the respective scale. Drawing all values on all scales and clamping the bars if the value exceeds the scale range would avoid this problem. However, the filled columns could be misinterpreted as large bars. This would increase the risk that bars of different scales are compared. Therefore, a placeholder is used to represent values that exceed the range of the respective scale (Figure 3.3(c)). The conducted experiments and the expert review (see Section 3.3.4) led to dashed lines as placeholders.

However, there are also drawbacks for these variants. The more visual elements are in the chart, the higher the risk of visual clutter. Furthermore, displaying multiple bars per column can additionally confuse the viewer. Therefore, these variants are used in an application specific way. In cases where values can be clearly assigned to a specific scale and comparability is only important inside these scales, the basic approach (Figure 3.3(a)) may be the best choice. It produces the clearest and simplest chart. In cases where many nearby values are mapped to different scales, it is important to show them on all possible scales (Figure 3.3(b)). The usage of placeholders (Figure 3.3(c)) can be additionally confusing and increases the visual load of the chart. They can be omitted if it is clear to the viewer on which scales the values are located, e.g., because they are already ordered by size.

3.3 Discussion and Evaluation

For a fair comparison in the following discussion, it is assumed that the complete scale-stack bar chart has the same spatial extent and pixel resolution as the other chart types.

3.3.1 Perception of Growth Behavior

Depending on the type of scale, certain types of growth behavior can be readily detected in charts. It is easy to see linear growth with linear plots. The same holds for logarithmic plots and exponential growth. Figure 3.4 shows scale-

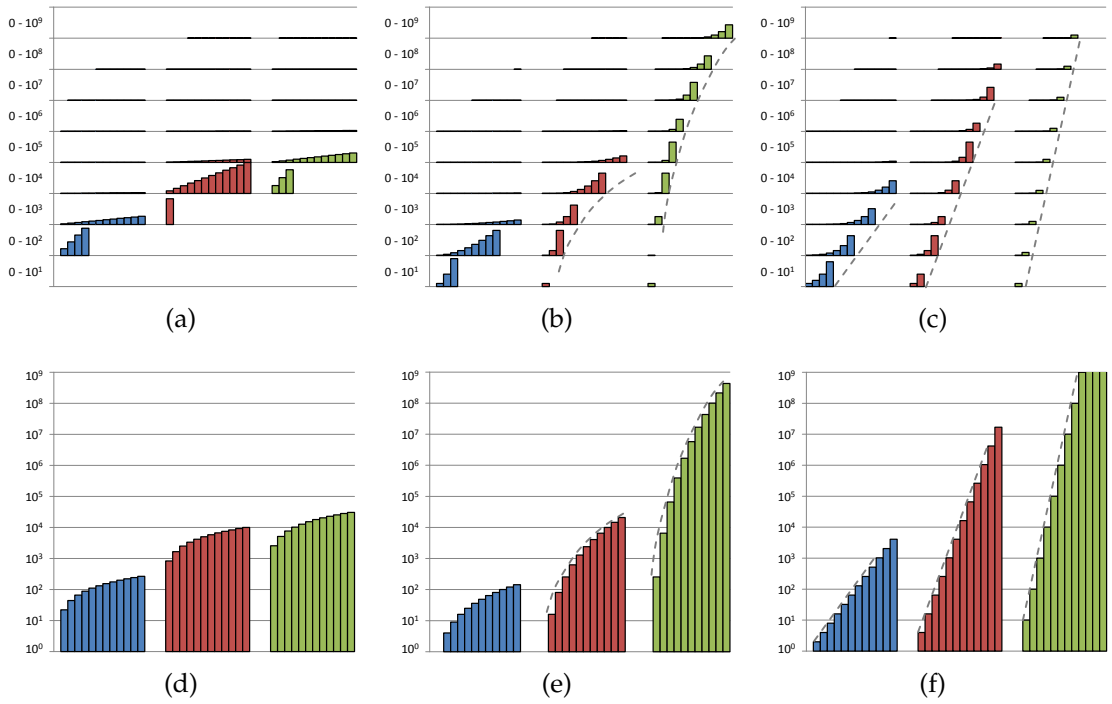


Figure 3.4 — Visualization of different growth behavior. Scale-stack bar charts showing (a) linear growth with three different slopes ($y = 22x$ (blue), $y = 831x$ (red), $y = 2546x$ (green)), (b) polynomial growth with three different degrees ($y = x^2$ (blue), $y = x^4$ (red), $y = x^8$ (green)), and (c) exponential growth with three different bases ($y = 2^x$ (blue), $y = 4^x$ (red), $y = 10^x$ (green)). The respective logarithmic plots are shown in (d)–(f). The envelopes (illustrated by dashed lines) of the logarithmic charts for polynomial and exponential growth can be approximated in scale-stack charts by connecting the last element of each row. The envelope for the blue bars in (e) is not shown due to the small number of sample points in (b).

stack bar charts and the respective logarithmic charts for three different types of growth: linear, polynomial, and exponential.

Since the proposed approach consists of linear plots on different scales, it is easy to detect linear growth (Figure 3.4(a)), in contrast to logarithmic plots (Figure 3.4(d)). The bars of the same scale grow at constant rate in scale-stack bar charts. Looking at the polynomial and exponential growth, it can be seen that the shape of the bars of the logarithmic charts also appears in scale-stack bar charts. As the figure shows, the envelopes of the logarithmic charts can be approximated in scale-stack bar charts. Therefore, it is also easy to differentiate between polynomial and exponential growth with scale-stack bar charts. In the case of polynomial growth (Figure 3.4(b)), the range of the scales grows faster than the values, i.e., the number of bars that appear with increasing row number grows faster than linear. This results in the shown curve when connecting the last bar of each row. In the case of exponential growth (Figure 3.4(c)), values and scale ranges exhibit the same growth behavior, resulting in a straight line when connecting the last bar of each row. It has to be noted that these chart properties necessitate the use of scale ranges that grow exponentially.

3.3.2 Advantages

There are several advantages of scale-stack bar charts. First of all, the approach works well with static images. The readability over the full value range does not require interaction like zooming or the display of tooltips. Furthermore, every row is self-contained. They can be viewed and interpreted independently. In contrast, extracting a subarea of a classic bar chart can lead to misleading visualizations, e.g., if the axis does not start at zero anymore. This property allows one to easily zoom, highlight, or filter parts of scale-stack bar charts. As a consequence, groups of values at certain magnitudes over very large ranges can be accurately displayed. It is also straightforward to use different units for the scales. In the case of temporal quantities, e.g., it is more intuitive to represent values with hours, days, and years, than using a single unit (e.g., seconds) for all scales (see Figure 3.5).

Scale-stack bar charts distribute the available accuracy, which depends on the resolution of the resulting image, over the full value range. The full accuracy of linear bar charts is, in contrast, only available for values of the highest order of magnitude. For example, an available resolution of $1,000$ pixels and 4 orders of magnitude of the data are assumed. In the case of a linear bar chart, values of the highest order of magnitude can be represented with 3 digits accuracy, but values of the lowest order cannot be represented anymore. With scale-stack bar charts, all values can be represented with an accuracy of more than 2 digits (250

pixels per order of magnitude). Hence, some accuracy is lost for high values but small values can be represented with the same accuracy. Logarithmic bar charts distribute the available accuracy also over the full value range, but inside the different orders of magnitude, the distribution is non-linear.

3.3.3 Disadvantages

The presented approach has also some drawbacks. First, it is clear that not all tasks benefit from it. Since scale-stack charts use a linear scale, they cannot perform better than linear charts if the values of interest are clearly visible in them. Furthermore, tasks like finding extrema can be efficiently accomplished with logarithmic charts. Another issue is that people are not used to scale-stack bar charts. Hence, they may require some training to read and correctly interpret them. There is also a potential risk that the viewer compares bars of different rows. However, similar risks exist also for logarithmic charts; not everybody is used to logarithmic scales and quantitative comparison of the bars' heights often leads to misinterpretation. Finally, the resulting chart contains more visual elements, increasing the visual load.

3.3.4 Evaluation

The previously discussed advantages and disadvantages result from theoretical considerations. To verify some of them, the method was evaluated with an expert review and a quantitative user study. First, the expert review was conducted to estimate the potential of scale-stack bar charts in general. They were then further evaluated with a quantitative user study measuring answer times and error rates. Furthermore, the expert review was also used to select an adequate placeholder (see Section 3.2.3): dashed lines as placeholders provided the best trade-off between reduced risk for misinterpretation and visual design. A summary of the results are presented in the following; a more detailed description of the expert review and the user study can be found in the original paper [3].

The experts think that linear charts work best if the data of interest are clearly visible in them, i.e., for tasks on large values. They further think that logarithmic charts are good for qualitative comparisons like finding extrema, but they may be difficult to use for quantitative tasks like estimating proportions or quantitative comparison of values. Scale-stack bar charts seem to work best when the value range is too large for linear charts and the tasks require a quantitative analysis or comparison of values. However, the experts pointed out some issues of scale-stack bar charts in their comments: The experts needed some time to get used to them and jumping between scales is time-consuming. Furthermore,

some of them see problems with the representation of negative values, because of the gap between positive and negative values (Figure 3.2(c)).

A quantitative user study was conducted with 15 participants from the Visualization Research Center of the University of Stuttgart requiring around 25 minutes to complete it. Each participant had to solve the following three tasks, each with linear, logarithmic, and scale-stack bar charts:

1. read value,
2. estimate ratio between two displayed values, and
3. determine which time step out of five exhibits the largest growth rate.

The participants could perform all tasks with scale-stack bar charts at least as fast and as accurately as with logarithmic plots. Especially, when a quantitative comparison of values is required, scale-stack charts provide higher accuracy (Task 2), or faster answer times (Task 3). Linear charts are fast to use, but they cannot accurately display values over the full range, resulting in high error rates in the study. These results coincide with the findings from the expert review.

3.4 Applications

The method was applied to data from three different fields—chemistry, social sciences, and the financial sector—which all exhibit a large value range. First, the half life of different chemical isotopes covers a large value range. As it is shown, even isotopes of the same element can already cover a range that is difficult to show in a single linear plot. Second, the populations of different countries can differ substantially. To demonstrate the ability to visualize stacked bars, the populations are divided into three different age categories in this dataset. Finally, the profits of different companies can differ largely. This dataset additionally demonstrates the representation of negative values.

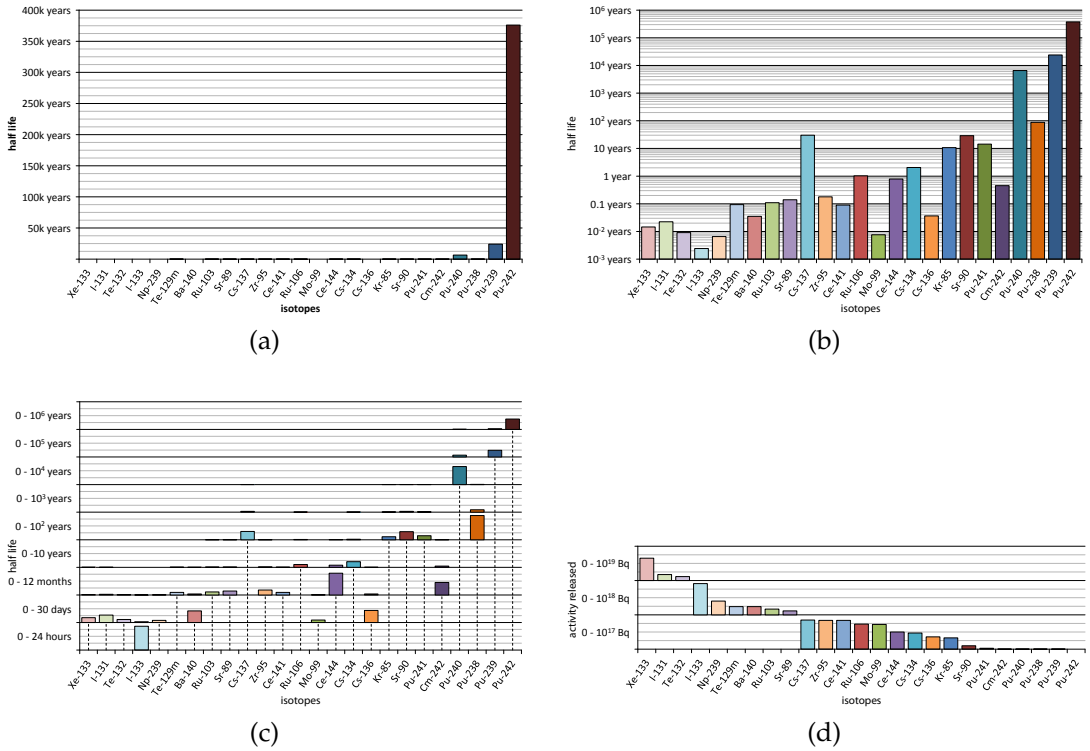


Figure 3.5 — The half life of different isotopes released to the environment during the Chernobyl disaster visualized with (a) linear, (b) logarithmic, and (c) scale-stack bar charts. In all charts, the isotopes are ordered on the x-axis according to the activity released during the accident. The respective activity values are shown in (d) with scale-stack bars. The bars are colored to be more easily distinguishable.

3.4.1 Half Life of Isotopes

The half life of isotopes is important, among other aspects, to quantify radioactive pollution. The presented data (Figure 3.5) was obtained from the UNSCEAR report [270] regarding the Chernobyl disaster in 1986. It contains estimates of the amount of the released isotopes and their half lives.

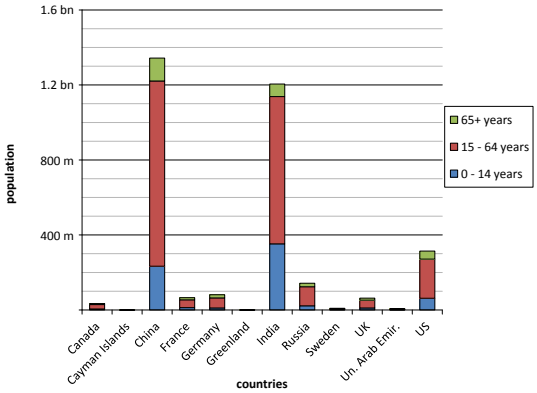
The linear bar chart (Figure 3.5(a)) is not suitable for this data. Because of the large half lives of the plutonium isotopes—more than 350,000 years for pu-242—most values are not visible in a chart of this size. With the logarithmic bar chart (Figure 3.5(b)), all values can be seen and it is easy to find the isotopes with smallest (i-133) or largest (pu-242) half life. However, reading values can already be difficult without experience with logarithmic plots. For example, it can be seen that the value of pu-238 is close to 100 years, but a more precise estimation is difficult. In the scale-stack bar chart (Figure 3.5(c)), the value lies clearly half way between 75 and 100 years, leading to an estimated value between 85 and 90 years (exact value is 87.7). Furthermore, there is a risk for the logarithmic chart that a visual comparison of values leads to wrong interpretations, e.g., one might get the impression that the three highest half lives (pu-240, pu-239, and pu-242) are quite close. In scale-stack bar charts, it is clearly visible that the three values are all in different orders of magnitude.

3.4.2 Age Distribution

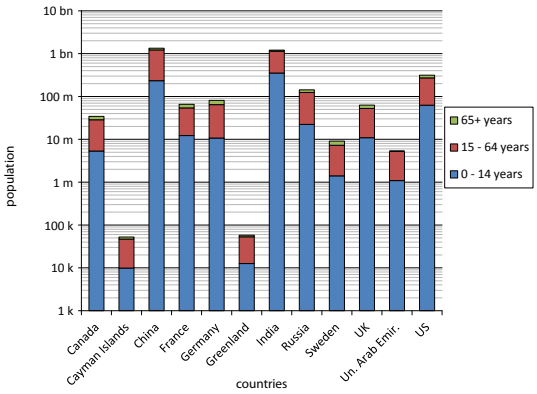
The data for the age structure of different countries are obtained from the CIA World Factbook [73] and visualized with stacked bars (Figure 3.6). Again, the linear chart (Figure 3.6(a)) allows analyzing only a subset of the data. The logarithmic chart (Figure 3.6(b)) can be used to look at the total population numbers. However, it is not very useful to analyze the distributions because the logarithmic scale distorts the visual proportions of the stacked bars. The chart provides the wrong impression that the largest age group in all countries is from 0 to 14 years. Furthermore, the respective value of a given bar depends on its position. For example, the third group (65 years and older) seems to have similar sizes for Canada, France, and Germany but in the scale-stack bar chart (Figure 3.6(c)), it can be seen that these groups have different sizes. Additionally, it is visible that the population of the United Arab Emirates consists only of a small part of old people. This is difficult to notice in the logarithmic plot.

3.4.3 Corporation Profits

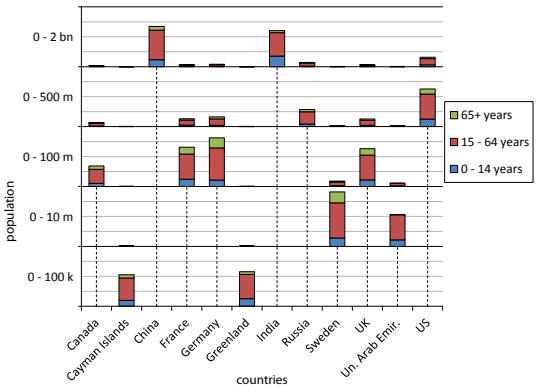
The high profits of Apple Inc. [27] can make it difficult to present them together with results from other technology companies like, e.g., Advanced Micro



(a)

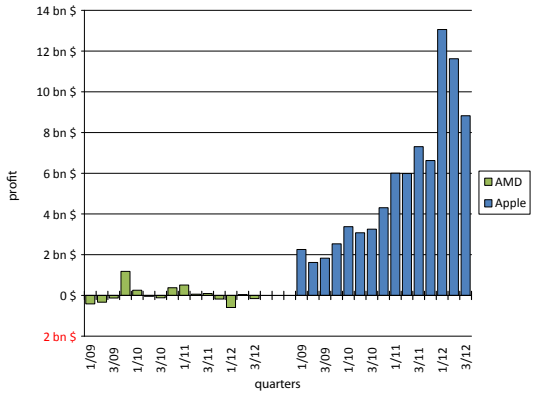


(b)

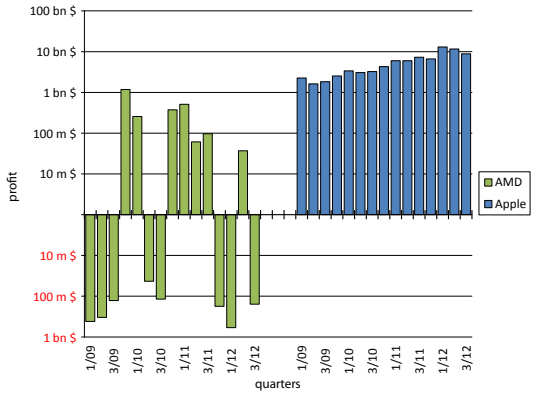


(c)

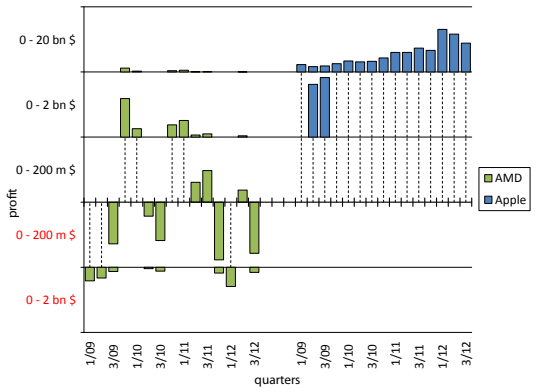
Figure 3.6 — Age structure of selected countries visualized with (a) linear, (b) logarithmic, and (c) scale-stack bar charts. The populations are divided into three age groups (see legend).



(a)



(b)



(c)

Figure 3.7 — Profits of AMD and Apple Inc. for the period from the first quarter of 2009 to the third of 2012, visualized with (a) linear, (b) logarithmic, and (c) scale-stack bar charts.

Devices Inc. (AMD) [14]. The data values are grouped by corporation, and a time range of almost 4 years is shown (Figure 3.7). The value range of this example is not as high as in the other two applications. Hence, most of the data can be analyzed with the linear bar chart (Figure 3.7(a)). However, AMD has some comparably small values and it is hard to quantify or compare them. The general tendency can also be seen in the logarithmic plot (Figure 3.7(b)) and all values are clearly visible, but the visual impression is less clear and makes a quantitative comparison more difficult. For example, the profit of AMD seems not to be that far away from Apple's profit. As the other two charts show, this is not the case. Another example is the decrease in AMD's profit from the last quarter of 2009 to the first quarter of 2010. It looks rather small in the logarithmic chart, but it is visible in the scale-stack chart that it decreased to less than a fourth. Furthermore, the peak in Apple's profit in the first quarter of 2012 is not as clearly visible in the logarithmic chart as in the other charts. The proposed approach (Figure 3.7(c)) allows also comparing the smallest values in the chart, e.g., it is easy to see that AMD's profit almost doubled from the second to the third quarter of 2011. This is hard to deduce from the linear chart, whereas it is not possible to directly compare bars in the logarithmic chart.

CHAPTER 4

Time-Dependent Vector Data

Vector data is employed in a wide range of fields in engineering and science. The spectrum includes mathematics, physics, engineering, and biosciences. For example, vector fields are used to represent fluid flow, which is not sufficiently possible with scalar fields. Often, the phenomena under investigation are time-dependent and can only be observed with appropriate visualization of the respective time-dependent vector fields.

A common approach is to create a time-dependent visualization by generating a sequence of images. In many cases, however, this is not the best approach because it relies on human memory and puts heavy load on human visual cognition. Therefore, many uses of animation in visualization are viewed skeptically by psychologists; see the discussion by Tversky et al. [269]. For example, tasks like comparing different positions over time or identifying regions of similar behavior are difficult and time-consuming (see, e.g., Figure 4.6, page 86).

The focus of this chapter[1] lies on the visualization of time-dependent vector fields describing unsteady flow, for example, from the field of computational

[1] **Parts of this chapter have been published in:**
M. Hlawatsch, P. Leube, W. Nowak, and D. Weiskopf. Flow radar glyphs—static visualization of unsteady flow with uncertainty. *IEEE Transactions on Visualization and Computer Graphics*, 17 (12):1949–1958, 2011 [4], © 2011 IEEE.
M. Hlawatsch, F. Sadlo, H. Jang, and D. Weiskopf. Pathline glyphs. *Computer Graphics Forum*, 33(2):497–506, 2014 [5], © 2014 John Wiley and Sons.
M. Hlawatsch, F. Sadlo, and D. Weiskopf. Hierarchical line integration. *IEEE Transactions on Visualization and Computer Graphics*, 17(8):1148 –1163, 2011 [6], © 2011 IEEE.

fluid dynamics. To overcome the issues with animated visualization described above, two novel glyph-based approaches are introduced in this chapter that are able to visualize time-dependent fields with static images. Furthermore, an acceleration scheme for the computation of dense integral curves is described. Dense integral curves are the basis for visualizations with FTLE (finite-time Lyapunov exponent) fields, a common approach for visualizing unsteady flow.

The first glyph-based approach (Section 4.3) uses a radial mapping of the directional information of vector fields and provides a static visualization of unsteady flow. This enables an easy analysis and comparison of the changes in local flow direction over time. Furthermore, the glyphs can also be used to visualize the uncertainty in vector fields, as discussed later in Chapter 6 (Section 6.3).

While this first approach deals with the local flow direction (Eulerian view), the second approach (Section 4.4) is based on pathlines and provides a Lagrangian view of the flow, i.e., a representation of the fluid motion. A novel type of glyphs was designed to improve pathline visualization of unsteady 2D flow. By using down-scaled pathlines as glyphs, visual clutter is reduced, allowing a dense coverage of the domain.

Both glyph-based techniques provide a visualization of time-dependent vector fields on multiple scales. While the overview scale allows identifying different regions in the flow, zooming-in enables the detailed analysis and comparison of time-dependent flow behavior in these regions.

Finally, an acceleration scheme that reduces the computational complexity of computing dense integral curves is presented at the end of this chapter. The algorithm is designed for high intrinsic parallelism, making it well-suited for many-core hardware architectures like GPUs. The scheme can be applied to time-independent and time-dependent vector fields alike and the main benefit is that dense visualization, e.g., using LIC (line integral convolution [63]) or the FTLE (Section 4.1.2), can be generated faster, or at higher resolution with the same computation time, respectively. Although this is achieved at the cost of increased memory consumption and reduced accuracy, it is shown that the advantages outweigh the disadvantages in typical applications.

4.1 Basics

A vector field $\boldsymbol{v}$ maps points in the domain to vectors. In the case of time-dependent vector fields, this mapping depends not only on the position $\mathbf{x}$ but also on time t:

$$\boldsymbol{v}(\mathbf{x}, t) : \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}^m. \tag{4.1}$$

In many applications and in the work presented here, the spatial dimension of the domain and the dimensionality of the vectors are equal, i.e., $n = m$.

4.1.1 Characteristic Curves

A general curve or *trajectory* in a vector field can be defined as follows:

$$\boldsymbol{\xi}(t; \mathbf{x}_0, t_0) : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}^n, \tag{4.2}$$

with time t being the curve parameter, $\mathbf{x}_0$ being the starting point or *seed* of the curve, and t_0 being the starting time.

Several types of characteristic curves in vector fields that are commonly used in flow visualization can be described by such trajectories. They are not only directly visualized but are also the basis for other visualization methods like LIC or FTLE-based techniques.

Pathlines $\boldsymbol{\xi}_p$ are solutions to the initial value problem for the following ordinary differential equation (ODE):

$$\frac{\mathrm{d}\boldsymbol{\xi}_p(t; \mathbf{x}_0, t_0)}{\mathrm{d}t} = \boldsymbol{v}(\boldsymbol{\xi}_p(t; \mathbf{x}_0, t_0), t), \tag{4.3}$$

with initial condition $\boldsymbol{\xi}_p(t_0; \mathbf{x}_0, t_0) = \mathbf{x}_0$. In other words, the tangents at all points of the pathlines are equal to the vectors defined by the vector field at respective positions. Pathlines can be interpreted as the trajectories of massless particles in the flow.

Streamlines $\boldsymbol{\xi}_{sm}$ are the instantaneous version of pathlines. They are obtained at fixed physical time, no matter if the vector field is time-dependent or not. They are solutions for the respective initial value problem for a slightly modified version of Equation (4.3) incorporating the fixed time t_0:

$$\frac{\mathrm{d}\boldsymbol{\xi}_{sm}(t; \mathbf{x}_0, t_0)}{\mathrm{d}t} = \boldsymbol{v}(\boldsymbol{\xi}_{sm}(t; \mathbf{x}_0, t_0), t_0), \tag{4.4}$$

with initial condition $\boldsymbol{\xi}_{sm}(t_0; \mathbf{x}_0, t_0) = \mathbf{x}_0$.

Streaklines $\boldsymbol{\xi}_{sk}$ are another type of characteristic curves commonly used in flow visualization. They are constructed by connecting massless particles that are continuously released at a fixed position in the domain. For example, continuously releasing dye into the flow results in streaklines. A streakline can be defined with points on a set of pathlines passing the same point $\mathbf{x}_0$ at different times:

$$\boldsymbol{\xi}_{sk}(s; \mathbf{x}_0, t_0) = \boldsymbol{\xi}_p(t_0; \mathbf{x}_0, t_0 - s), \tag{4.5}$$

with $s \in [0, T]$ being the parametrization of the streakline and T being the time span covered by the streakline. The respective pathlines used for constructing the streakline start in the time range $[t_0 - T, t_0]$ but are all evaluated at fixed time t_0. The end point of the streakline ($s = T$) corresponds to the point for time t_0 on a pathline passing $\mathbf{x}_0$ at time $t_0 - T$.

In the case of steady flow, i.e., the vector field is not time-dependent, these three types of curves are identical. Furthermore, the computation of these curves requires the integration of the vector field. This typically necessitates the use of numerical integration methods like Runge-Kutta methods [224].

4.1.2 Finite-Time Lyapunov Exponent

The *Lyapunov exponent* (LE) is a concept from dynamical systems theory measuring the exponential rate of divergence of two close orbits (trajectories in phase space) of the system. An alternative interpretation of the LE is that it measures the exponential growth of an infinitesimal perturbation.

The LE was originally introduced for predictability analysis in temporally unconstrained dynamical systems. For an n-dimensional system, or vector field, it is a spectrum of n exponents. The largest exponent σ_1 is of major importance since it represents the upper bound for the growth of a perturbation and it is therefore often denoted as the Lyapunov exponent itself. It is defined as

$$\sigma_1(\mathbf{x}) = \lim_{T \to \infty} \lim_{\|\boldsymbol{\delta}(\mathbf{x}, t_0)\| \to 0} \frac{1}{|T|} \ln \frac{\|\boldsymbol{\delta}(\mathbf{x}, t_0 + T)\|}{\|\boldsymbol{\delta}(\mathbf{x}, t_0)\|}. \tag{4.6}$$

In this equation, $\boldsymbol{\delta}(\mathbf{x}, t)$ represents a perturbation at time t, having originated at position $\mathbf{x}$ and time t_0. The perturbation is oriented at time t_0 such that σ_1 becomes maximal after time T. In other words, the perturbation $\boldsymbol{\delta}(\mathbf{x}, t) = \boldsymbol{\xi}(t; \mathbf{x} + \boldsymbol{\delta}(\mathbf{x}, t_0), t_0) - \boldsymbol{\xi}(t; \mathbf{x}, t_0)$ represents the deviation of the perturbed trajectory $\boldsymbol{\xi}(t; \mathbf{x} + \boldsymbol{\delta}(\mathbf{x}, t_0), t_0)$ from the reference trajectory $\boldsymbol{\xi}(t; \mathbf{x}, t_0)$, both started at time t_0 but at positions $\mathbf{x} + \boldsymbol{\delta}(\mathbf{x}, t_0)$ and $\mathbf{x}$, respectively.

When computing the LE for real-world data, one issue arises: integration of infinite length is usually not applicable. Furthermore, the spatio-temporal domain of real-world data is often finite. This led to the idea of replacing the infinite time range in Equation (4.6) by a finite one, resulting in a counterpart for time-constrained problems: the *finite-time Lyapunov exponent* (FTLE).

The FTLE is commonly computed using the *flow map*

$$\boldsymbol{\xi}_{t_0}^{T}(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^n, \tag{4.7}$$

mapping from the seed $\mathbf{x}$ at time t_0 of a trajectory to its end point after advection for finite time T. In practice, the flow map is typically obtained by integration of trajectories seeded on a regular grid.

The direction of maximum perturbation growth corresponds to the major eigenvector of the right Cauchy-Green deformation tensor

$$\mathbf{C}_{t_0}^{T}(\mathbf{x}) = (\nabla \boldsymbol{\xi}_{t_0}^{T}(\mathbf{x}))^{\top} \nabla \boldsymbol{\xi}_{t_0}^{T}(\mathbf{x}). \tag{4.8}$$

In other words, a perturbation oriented in this direction at time t_0 and position $\mathbf{x}$ experiences maximum growth under the action of the flow for time T. In the context of predictability analysis, this means that predictability is lowest with respect to these perturbations at time t_0 and position $\mathbf{x}$. The maximum growth factor can be obtained using the spectral norm of the flow map gradient $\nabla \boldsymbol{\xi}_{t_0}^{T}(\mathbf{x})$, which is defined as the square root of the major eigenvalue $\lambda_{\max}(\cdot)$ of $\mathbf{C}_{t_0}^{T}(\mathbf{x})$. Hence, the maximum FTLE, in this work simply denoted as the FTLE or σ, at position $\mathbf{x}$, time t_0, and advection time T, is defined as

$$\sigma_{t_0}^{T}(\mathbf{x}) = \frac{1}{|T|} \ln \sqrt{\lambda_{\max}(\mathbf{C}_{t_0}^{T}(\mathbf{x}))}. \tag{4.9}$$

4.2 Related Work

The huge variety of different techniques for flow visualization can be classified in four different categories: texture-based and dense flow visualization (Section 4.2.1), geometric flow visualization (Section 4.2.2), feature- and topology-based techniques (Section 4.2.3), and glyph-based techniques. Since the glyphs introduced in this chapter are not only inspired by flow visualization techniques, a broader spectrum of related work is discussed in this context (Section 4.2.4). Furthermore, the visualization methods presented in this chapter are also related to methods for the static visualization of time-dependent data (Section 4.2.5) and hierarchical computation schemes (Section 4.2.6).

4.2.1 Texture-Based and Dense Flow Visualization

Texture-based methods have a long tradition in flow visualization. They typically generate a dense visualization of the flow [196] and often provide a visual impression that is similar to physical experiments, e.g., like releasing dye into a fluid. Laramee et al. [175, 176] give a survey of such visualization techniques.

The most prominent example of texture-based flow visualization is LIC by Cabral and Leedom [63]. This method convolves a texture along densely seeded streamlines. Hierarchical line integration presented in Section 4.5 is able to accelerate LIC computations without the need for filter kernels of lower quality (Section 4.5.3.2). In particular, it reduces the computational complexity in contrast to most previous work on acceleration of texture-based methods that employ a mapping to GPUs [136, 147, 279, 299].

Fast LIC by Stalling and Hege [135, 256] is to the author's knowledge the only other example that reduces the number of LIC filter operations. Fast LIC avoids multiple computations of similar streamlines by scattering streamline information across the grid. Scattering methods are less cache-friendly and not well suited for parallel execution on multi-core hardware architectures. Additionally, atomic operations are required to avoid race conditions between parallel threads. Hierarchical line integration (Section 4.5) uses a gathering approach to avoid redundant computations and is therefore more suitable for GPUs and multi-core CPUs. Furthermore, it supports computations beyond LIC, integration of further quantities, time-dependent data, and higher-dimensional domains.

Li and Shen [181] use so-called trace slices to accelerate view-dependent texture advection. Similar to the concept of *coordinate maps* in Section 4.5.1.1, their trace slices provide a mapping from starting positions to positions on reverse-time

pathlines. Instead of building a hierarchy with fixed resolution and increasing trajectory length as in the case of coordinate maps, Li and Shen create a set with different resolutions similar to mipmapping for multi-resolution texture advection.

4.2.2 Geometric Flow Visualization

The survey by McLoughlin et al. [190] provides an overview of geometric flow visualization in general, while the overview by Weinkauf and Theisel [295] focuses especially on streak- and timelines. Weinkauf et al. [296] present a generalization scheme that allows them to describe the common types of characteristic curves (see Section 4.1.1) as tangent curves of a derived vector field. While methods based on pathlines typically employ the model of massless particles, Günther et al. [116] propose an approach with mass-dependent integral curves for certain applications.

In the field of geometric flow visualization, line placement and selection techniques are important research topics. Recent work on streamline placement methods includes the method by Yu et al. [310], who use flow saliency to generate streamline bundles, and an approach to parallelize placement in 2D fields by Zhang et al. [312]. While placement methods adapt the seed positions, selection methods reduce a set of given lines, e.g., McLoughlin et al. [191] reduce streamline sets for interactive seeding, while Tao et al. [260] treat streamline selection as a joint problem with viewport selection. A comprehensive overview of streamline placement and selection can be found, e.g., in the work by Günther et al. [122].

In contrast to streamlines, pathlines can intersect; hence, placement and selection of them are more difficult and only few papers consider this topic. Günther et al. [122] present a general method for sets of lines—including pathlines—which adapts the opacity to avoid the occlusion of important features. Pathline selection can also be achieved with the methods by Falk et al. [99], Salzbrunn et al. [240], and Weinkauf et al. [297].

4.2.3 Feature- and Topology-Based Flow Visualization

An overview of feature- and topology-based techniques can be found in several surveys [220, 221, 241, 246] on these topics. Typically, these methods are aimed at structuring the vector field into areas of similar behavior or extracting the relevant features of the flow. Topological methods in flow visualization were established by Helman and Hesselink [137, 138] with their work on vector field topology for 2D and 3D flow. Vector field topology segments the vector field

into different regions of coherent flow behavior. Since it is based on streamlines, it can only provide an instantaneous view on unsteady flow fields.

The concept of Lagrangian coherent structures (LCS) represents an alternative to vector field topology that is also well applicable and interpretable for transient vector fields because it is based on true advection (pathlines) instead of instantaneous streamlines. Coherent structures have been roughly defined as regions of coherent motion or where a flow variable exhibits significant correlation with itself or another variable [229]. Haller has defined LCS to be a time-dependent analog of separatrices [128, 131] (hence separating regions of coherent motion) and has shown that they can be obtained using the FTLE (see Section 4.1.2). A detailed analysis of LCS in n-dimensional systems is presented by Lekien et al. [180]. In visualization, Garth et al. [113] introduced direct visualization of the FTLE, whereas Sadlo and Peikert [235] extracted height ridges from FTLE fields and compare them to vector field topology for the case of stationary vector fields. An advantage of LCS is their robustness under the effects of noise, as analyzed by Haller [129] for approximated velocity data. A vector field topology for time-dependent 2D vector fields based on the FTLE and streaklines was introduced by Sadlo and Weiskopf [236]. This approach was later extended by Üffinger et al. [105] to time-dependent 3D vector fields.

There are other approaches for feature extraction and topological analysis of vector fields, e.g., recent work by Obermaier and Joy [199] based on function fields or cell tracking for LCS approximation by Kuhn et al. [168]. However, FTLE-based visualization methods have become very popular over the last decade. Besides the traditional flow map-based method, there are also other approaches for computing FTLE fields: FTLE computation with renormalization [109, 195], localized FTLE [152], and extensions of flow map-based FTLE [104]. However, comparisons of these methods [104, 167] show that the traditional approach has some advantages, e.g., small features in the flow cannot be missed. Since it requires the time-consuming computation of a large number of integral curves, acceleration approaches are an important topic in flow visualization.

Early examples of FTLE acceleration strategies are presented by Garth et al. [112] and Sadlo and Peikert [234], who use adaptive refinement of the computational grids. Later, Agranovsky et al. [15] and Barakat and Tricoche [34] presented acceleration and refinement approaches with sparse samples. Barakat et al. [35] present an approach for the interactive exploration of FTLE fields that employs GPU-computations and hierarchical data structures. Using GPUs for FTLE computation is also discussed by Ameli et al. [23]. Several publications [70, 124, 198] focus on the computation of FTLE fields on cluster architectures for very large datasets.

However, all these approaches are still subject to linear computational complexity with respect to the integration range. For the special case of FTLE time series, Brunton and Rowley [55] show that computations from previous time steps can be reused to reduce computational effort. Hierarchical line integration (Section 4.5) shares this basic concept. However, it is more general and allows the accelerated evaluation of quantities along trajectories, e.g., required for LIC.

4.2.4 Glyph-Based Visualization Techniques

Glyph-based techniques use symbolic elements to represent properties at selected discrete locations. Their flexibility is one of their main advantages and allows encoding very different types of information. A comprehensive overview and discussion of glyph-based techniques for visualization is provided by Borgo et al. [46].

In flow visualization, glyphs have been used for a long time to visualize vector fields. Using arrows to depict flow direction is a standard method and many extensions of this basic idea have been proposed, including the combination of arrow glyphs and streamlines by Bertrand and Tanguy [45] and arrow glyphs on streamsurfaces according to Löffelmann et al. [184]. Van Walsum et al. [276] present concepts for combining glyphs with feature extraction. Pickett and Grinstein [215] introduced stick-figures, a more complex glyph for multidimensional data. De Leeuw and Van Wijk [82] also describe glyphs with higher complexity for flow visualization. Kirby et al. [159] demonstrate how to combine different glyphs encoding flow properties. Peng et al. [212] use glyphs to represent local flow properties of their vector field clusters. An augmentation of non-scaled streamlines with glyphs is presented by Pilar and Ware [217].

Ward [289] discusses glyph placement for multivariate data and dense seeding to visualize spatial relationships. Chung et al. [72] discuss the sorting of glyphs along coordinate axes for multivariate data. Dovey [93] shows that jittering the seeding positions by a small offset can avoid visual patterns induced by seeding on regular grids. Highlighting certain glyphs can reduce visual overload as in the work by Boring and Pang [48]. Klassen and Harrington [160] describe how shadows can improve perception of glyphs for 3D flow. Schultz and Kindlmann [249] discuss properties of a glyph for visualizing higher-order tensors. The glyph was designed to preserve certain characteristics of the underlying data (e.g., symmetries) in the visualization. Their design strategy is very generic; it was applied to design the flow radar glyphs in Section 4.3.

In information visualization, radial mapping and ring-shaped glyphs are already

a common tool. Bak et al. [33] use glyphs inspired by the growth rings of trees to visualize sensor data in a spatiotemporal way. Spatiotemporal data is also visualized with the ringmaps by Zhao et al. [313]. Chen [69] employs ring glyphs to detect patterns in scientific literature. Carlis and Konstan [66] show that mapping data onto a spiral helps analyze periodicity in data. Similar concepts are presented by Weber et al. [294]. Tominski et al. [261] use helix-shaped glyphs to visualize spatiotemporal data. Burch and Diehl [57] use radial visualizations to display dynamical hierarchical information. An overview of these and further techniques is given by Draper et al. [94] and Burch and Weiskopf [58]. While flow radar glyphs (Section 4.3) share the radial layout with the above techniques from information visualization, they differ in the actual mapping of data because they target flow field information.

There are some more general concepts from information visualization that can also be applied to glyph-based techniques. Drawing multiple plots into a single chart often leads to visual clutter; the small multiples approach [266] splits such a chart into a grid of multiple charts for better comparison. Sparklines [268] use downscaled plots for integration into continuous text. However, the glyphs presented in this chapter employ spatial sampling of a domain that is typically of much higher density than considered in these approaches.

4.2.5 Static Visualization of Time-Dependent Data

Scientific visualization often employs a direct mapping of data dimensions (typically, 2D or 3D space, and time) to analogous visualization dimensions. However, there are also approaches that generate a static visualization of time-dependent data. Examples include space-time approaches for unsteady flow: Falk et al. [99] combine FTLE fields with pathlines, Sadlo et al. [238] augment FTLE ridge surfaces with LIC, and Karch et al. [151] introduce several streamline-based concepts. Woodring and Shen [306] present static visualization of time-dependent volumetric data. This approach is generalized by Woodring et al. [307], treating 3D time-dependent data as 4D data. A survey of techniques employing space-time cubes is given by Kristensson et al. [165]. Joshi and Rheingans [150] also discuss methods for visualizing time-dependent data with static images, but with the focus on illustrative rendering techniques.

4.2.6 Hierarchical Computation Schemes

Algorithms in computer science often rely on hierarchical data structures or computation schemes to reduce computational complexity. Typical examples are divide-and-conquer algorithms. Hierarchical line integration (Section 4.5)

particularly resembles the acceleration strategy of pyramid algorithms known from digital image processing.

Pyramid algorithms were introduced by Burt [62] for image filtering. The idea behind pyramid algorithms is to iteratively construct a pyramid of images with decreasing resolution. For example, Gaussian blur can be efficiently implemented by repeated application of Gaussian filters. Similarly, image operations for computer graphics can be accelerated by pyramid techniques [203]. The filter principle relies on repeated application of filter operations, which can also be used in the form of general repeated integration according to Heckbert [133]. Due to the direct mapping of regular image grids to textures, pyramid filters lend themselves to efficient GPU implementations; a summary of GPU filters is presented by Kraus and Strengert [164]. An application of hierarchical computation in visualization is described by Lum et al. [185], who use a hierarchical method for calculating pre-integration tables for direct volume rendering with $O(N^2)$ complexity, instead of $O(N^3)$ for naive implementations.

Although hierarchical line integration (Section 4.5) adopts the acceleration strategy of pyramid and hierarchical methods, it exhibits the following structural differences. First, filtering along curved 1D trajectories is considered, leading to a more demanding filter process. Second, to increase accuracy, the resolution is not decreased while progressing to higher levels of the compute hierarchy.

4.3 Flow Radar Glyphs

This technique considers local flow direction and velocity magnitude at discrete sampling points, i.e., it provides an Eulerian view on unsteady flow. This glyph-based approach visualizes time-dependent fields with static images. The glyphs—called *flow radar glyphs*—use an intuitive mapping of the directional information that preserves certain data properties like periodicity. Furthermore, the glyphs can be extended to incorporate uncertainty (see Chapter 6, Section 6.3) or for other cases, in which a single direction per time-step is not sufficient.

4.3.1 Concept

The basic concept of flow radar glyphs is the radial mapping of temporal variations in vector directions (see Figure 4.1). Using a polar coordinate system whose center coincides with the center of the glyph, vector direction is mapped to the angle and time is mapped to the radius. A temporal sequence of directions results in a spatially, radially organized sequence of points in the coordinate system. These sample points are connected according to the interpolation scheme employed for the dataset. Linear interpolation results in the curve shown in Figure 4.1. Hence, the directions are encoded in the positions on the curve relative to the center of the glyph. To obtain the directions from the glyph, lines can be mentally drawn from the positions on the curve to the center. This kind of mapping preserves the intuitive human notion of directions, as it is also the case with hedgehog or arrow glyphs. Figure 4.2(a) illustrates how to read and interpret flow radar glyphs, and Figure 4.2(b) shows an example for a simple analytic vector field.

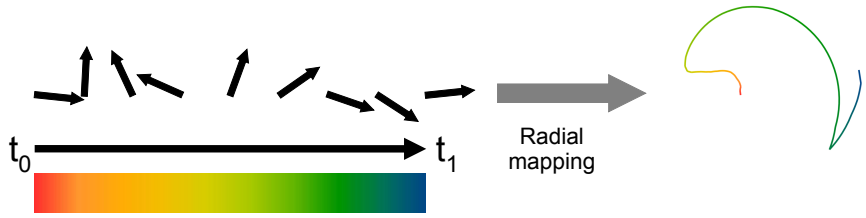
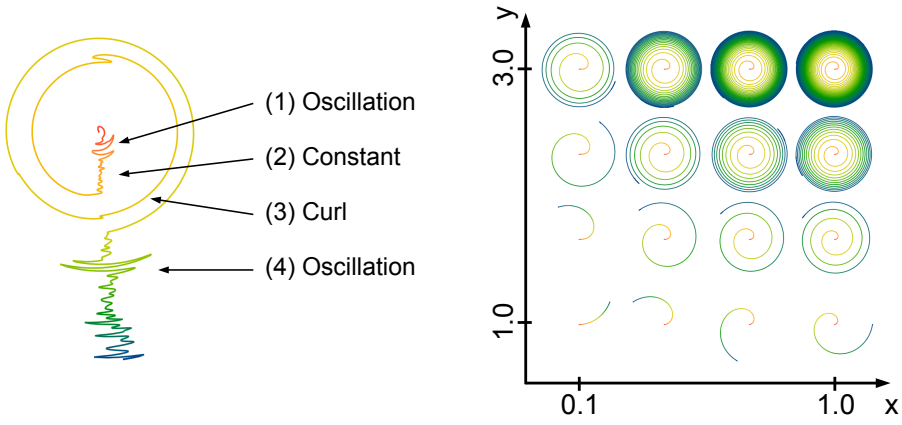


Figure 4.1 — Basic principle of flow radar glyphs. Direction is mapped to the angle, and time is mapped to the radius of a polar coordinate system. The left sequence of directions results in the right glyph. Color mapping enhances the visualization of the temporal behavior. The color map from this figure is used for all images in this section.



(a) (b)

Figure 4.2 — (a) How to read flow radar glyphs. (1) The flow starts with short oscillations in downward direction, (2) followed by almost constant downward motion. (3) A curl with two cycles follows. (4) At the end, the flow direction is again downwards with oscillations of varying intensity. (b) Visualization example of the flow (v_x, v_y) with $v_x = \cos(t^y \cdot x)$, $v_y = \sin(t^y \cdot x)$ and $0.0 \leq t \leq 2\pi$, $0.1 \leq x \leq 1.0$, and $1.0 \leq y \leq 3.0$. The number of periods increases in x-direction and the acceleration of the rotation increases in y-direction.

Further advantageous properties result from the used radial mapping. The periodicity of angles is preserved, i.e., there is no discontinuity in the visualization of angular ranges larger than 2π. This is, e.g., not the case for Cartesian plots. Symmetries in the directions are also preserved, facilitating the comparison of neighboring positions. This allows the visual analysis of data on multiple scales (Section 4.3.2). Mapping time to the radius is also beneficial: time has monotonic behavior and is constant over space, which results in constant size for all glyphs. An additional aspect of the glyph is that it converges to the classic arrow glyph as the time range covered converges to zero (Section 4.3.1.2) or in the case of steady flow. The local flow direction is constant in these cases, resulting in a flow radar glyph with a single line. As with arrow glyphs, it is possible to mentally reconstruct streamlines by connecting these lines in these cases. This is, however, not possible in unsteady flow with the glyph covering a non-zero time range.

Mapping local vector directions is intuitive because of the direct relationship to the glyph positions, and thus, only respective examples are presented here. However, the concept is not restricted to this and mappings of other directional information are possible, such as the tangential direction along pathlines.

4.3.1.1 Magnitude-Scaled Glyphs

The mapping described above has the issue that the vector magnitude is ignored. This leads to an overestimation of areas with low magnitudes. Therefore, a more general mapping to the glyph radius is defined in the following. The idea is to map the product of time and vector magnitude, i.e., the path length, to the glyph radius. This leads to the following ordinary differential equation:

$$\frac{dr(t)}{dt} = \frac{|\mathbf{v}(t, x_0)|}{v_{\text{norm}} T}, \quad r(t_0) = 0, \tag{4.10}$$

where $r(t)$ is the radius at time t, $|\mathbf{v}(t, x_0)|$ is the magnitude of the vector at position x_0 and time t, v_{norm} is a normalization factor to control the glyph size, and T is the time range covered by the glyph. The following equation provides then a first-order approximation of Equation (4.10):

$$r_i = \frac{|\mathbf{v}_i| \, \Delta t}{v_{\text{norm}} T} + r_{i-1}.$$

Here, r_i is the radius after the i-th iteration, $|\mathbf{v}_i|$ is the magnitude of the local vector, and Δt the time range of an iteration step.

In this form, the radius is not only increased with time, but also in dependency of the vector magnitude. Hence, positions with higher average magnitude exhibit glyphs of bigger size (Figure 4.3). The drawback of this new mapping is the ambiguity of the radius because different combinations of vector magnitudes over time can lead to the same radial position. However, this is compensated with color mapping of the temporal range.

(a) (b)

Figure 4.3 — Comparison of magnitude-scaled and normalized flow radar glyph. (a) The magnitude-scaled glyph depicts directional variation over time, including flow magnitude. (b) The normalized glyph discards the magnitude and shows only the directional variation. The difference is visible around the center of the glyphs. Because of low velocities at the beginning, the curl disappears in the magnitude-scaled glyph.

A glyph for which the magnitude is discarded can be derived from the magnitude-scaled glyph by using normalized vectors. This leads to $r_i = \Delta t/T + r_{i-1} = i\Delta t/T$. This variant is called normalized glyph in the remainder of this chapter. Figure 4.3 shows a comparison of the magnitude-scaled and the normalized glyph. The visual difference between both types for a full dataset can be seen in Figures 4.9(a) and (b).

4.3.1.2 Time Range

An issue of mapping time to radius is the uneven angular resolution, which increases with increasing radius. Thus, the visual accuracy of the displayed directions is not constant over the time range. The mapping of time to increasing radius therefore emphasizes the behavior at the end of the time range (Figure 4.4(a)). This is suitable for applications in which events that are more recent are of greater interest or importance.

However, the information at the end of a temporal range in datasets is not always the most significant one. To account for this, the time range covered by the glyphs can be adapted. This helps analyze the flow behavior in more detail (Figure 4.4(b)). Keeping the color map invariant when changing the time range has a similar visual effect as zooming-in on the glyph. This can be advantageous when comparing glyphs for different time ranges. However, the color map can also be adapted to the selected time range such that the full color range remains available (Figure 4.4(c)). This supports the interpretation of the glyphs for smaller time ranges, and helps compare neighboring glyphs in detail or detect patterns in coarse-scale visualizations (see Section 4.3.2).

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

Figure 4.4 — Different mappings of time for the same spatial position. (a) The full time range is mapped to increasing radius. (b) A smaller time range (marked in (a)) is mapped to increasing radius. (c) The color map is adapted to the selected time range from (b). (d) The full time range is mapped to decreasing radius.

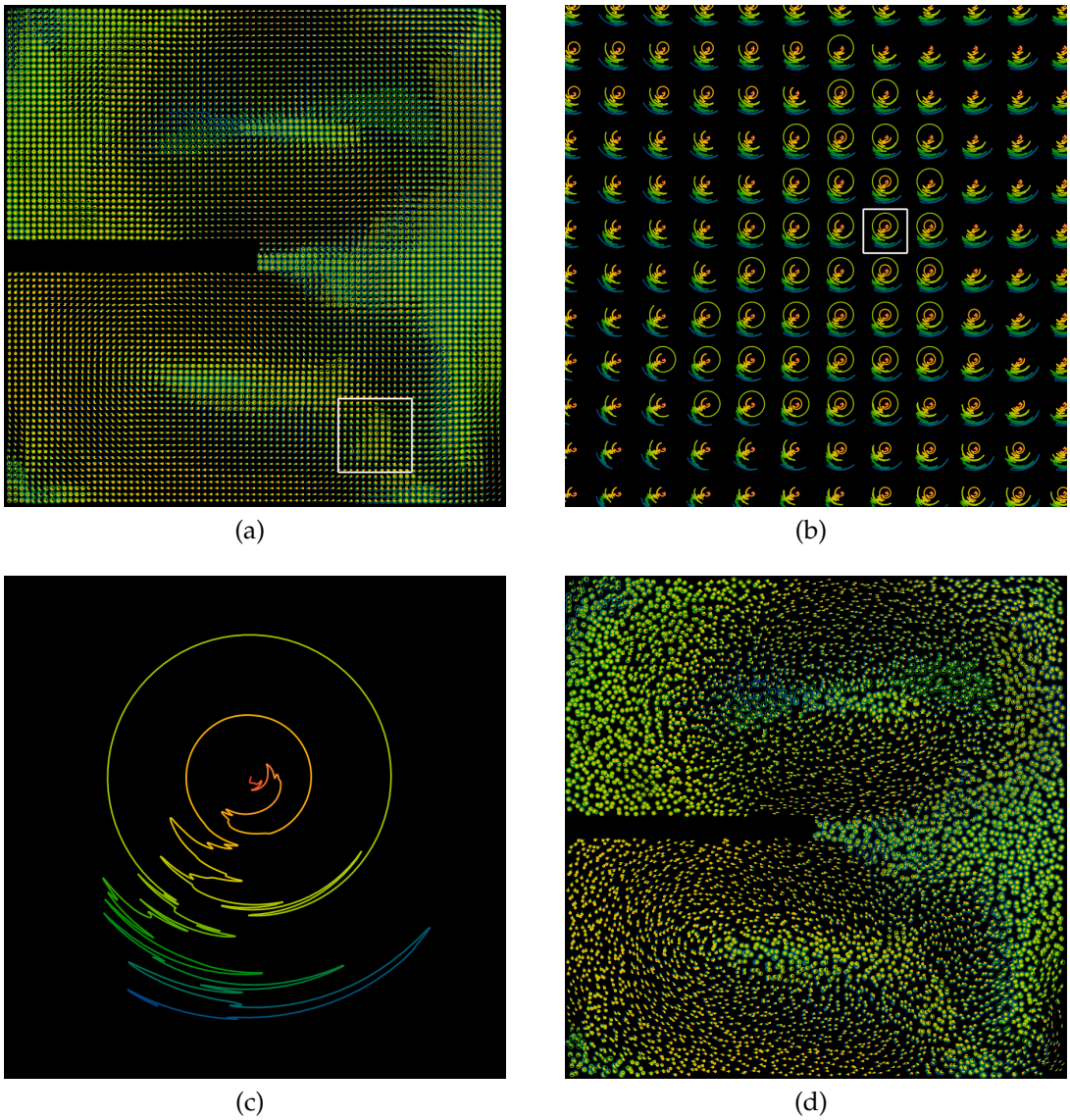
However, the problem of low angular resolution at the beginning of the time range still remains. Therefore, it is also possible to invert the mapping of time to radius. In this case, time is mapped to decreasing radius and the beginning of the time range is emphasized (Figure 4.4(d)). Analyzing the data with both mappings prevents the user from missing important details at the beginning of the time range. It is also possible to use a non-linear mapping of time to emphasize certain sections of the time range. However, it is recommended that the user starts the analysis with a linear mapping of time to increasing radius, which is more natural than an inverted or non-linear mapping.

All of these variants can be applied in an interactive way with the proposed implementation for GPUs (see Section 4.3.5).

4.3.2 Multi-Scale Visualization

The visualization of flow data with densely seeded flow radar glyphs allows an analysis of the data on multiple scales. Three qualitatively different scales for the resulting visualization were identified (Figure 4.5). On the coarsest scale, many small-sized glyphs visualize a large region of the data and a partition of the data into coherent regions is visible through visual fusion (Figure 4.5(a)). On the medium scale, fewer glyphs are visible with larger extent, allowing direct comparison of neighboring positions (Figure 4.5(b)). The finest scale shows the zoomed-in view of single glyphs (Figure 4.5(c)). This enables a detailed analysis of the temporal evolution of the flow direction. In the case of dense seeding, visual patterns can occur due to regular seeding. Jittering the seeding positions randomly by a small offset can remove these patterns. However, detecting the edges of coherent regions can also be more difficult with jittered seeding (Figures 4.5(a) and (d)). Further examples of multi-scale visualization are presented in the result section (Section 4.3.6).

The following work flow for the visual analysis with flow radar glyphs is suggested: The user starts with a zoomed-out view to detect interesting areas and patterns in the data. Switching between regular and jittered seeding can help classify the occurring patterns. Zooming-in allows then a more detailed analysis of the spatiotemporal behavior in these areas. If interested in the detailed temporal evolution of the flow direction, e.g., to observe the effects of certain events on it, the user can examine single glyphs. Changing the time range (Section 4.3.1.2) can further support the user in understanding temporal processes in the flow.



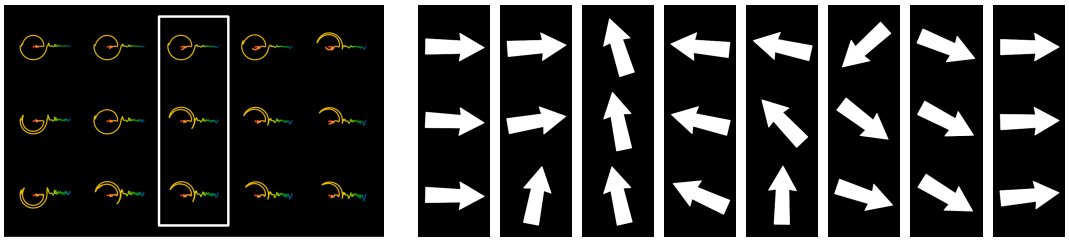
(a) (b) (c) (d)

Figure 4.5 — Multi-scale visualization with flow radar glyphs. (a) On the coarsest scale, patterns in the data are visible and different regions of similar behavior can be distinguished. (b) On the medium scale, neighboring positions and glyphs can be compared and the spatiotemporal behavior becomes visible in more detail. (c) On the finest scale, the temporal progression of the flow direction at individual positions can be analyzed. (d) Jittered seeding can remove patterns induced by regular seeding.

4.3.3 Comparison to Existing Techniques

Common methods for the visualization of 2D unsteady flow are animated arrow glyphs and the representation by path- or streaklines (as geometrically rendered lines or indirectly in the form of texture-based flow visualization). Arrow glyphs show directions of a single time step only. An animation of them enables the analysis of temporal processes. Furthermore, it is also possible to display a sequence of arrow glyphs in a static image (see Figure 4.6). In both cases, the analysis and comparison of multiple spatiotemporal positions are difficult. Figure 4.6 shows that the differences of the neighboring positions can easily be detected with flow radar glyphs. The same information can be gained with arrow glyphs, but more effort is required to compare neighboring positions. In this example, only the upper position shows a full rotation. With animated classic vector glyphs, the same task requires watching the animation multiple times. Still, with animated vector glyphs it is hard to analyze and compare the flow behavior over a large time range of the flow, e.g., to detect the stronger oscillations in the bottom region at the end of the time range. Even when analyzing only a single position, small temporal details can be missed, like the small oscillation in the upper row before the direction turns downward and back to the right.

Flow radar glyphs cannot be directly compared with path- or streaklines because the latter visualize Lagrangian properties. However, they are common tools for analyzing unsteady flow and a qualitative comparison can be done (Figure 4.7). The figure shows that it is quite difficult to analyze the full spatiotemporal domain of the field due to the susceptibility to visual clutter of these methods. Even with the method by Weinkauf et al. [297] that reduces



(a) (b)

Figure 4.6 — (a) Flow radar glyphs in comparison to (b) an animation of classic vector glyphs. The marked area in (a) is shown as a sequence of arrow glyphs (from left to right in (b)). The flow radar glyphs cover the full time range of the dataset; the sequence of arrow glyphs covers only a smaller time range, colored yellow in the flow radar glyphs.

(a) (b) (c)

Figure 4.7 — Flow radar glyphs in comparison to path- and streaklines. (a) Regularly seeded pathlines with a tenth of the full time range used as integration range. (b) Regularly seeded streak lines with half of the integration range of (a). (c) Magnitude-scaled flow radar glyphs covering the full time range.

clutter for characteristic curves, the visual signature of flow radar glyphs fundamentally differs from line-based methods. Another problem is that path- or streaklines depend on seeding in space and time. Therefore, they are typically used with animation—with the problems of cognitive load inherent to animated visualizations. In contrast, flow radar glyphs can display the full temporal range of the data without visual clutter. As a further advantage, the glyphs can incorporate uncertainty with all spatiotemporal details (see Section 6.3.1).

4.3.4 Three-Dimensional Flow

The visualization of 3D data is challenging because of occlusion and projection issues, which are inherent to any 3D flow visualization method. In the following, only regular seeding on a plane through the 3D domain is considered. Other seeding strategies on planar or non-planar 2D manifolds through 3D space should be feasible without necessitating any changes to the glyph. Seeding at arbitrary 3D positions would be more challenging due to issues with occlusion and depth perception. It is important to note that the glyphs, even if seeded on 2D manifolds, show the full 3D vector data at the sample positions.

Instead of polar coordinates, spherical coordinates are now used for the glyphs. The direction is mapped to the two angles ϕ and θ, and the time is mapped to the radius (Figure 4.8(a)). Usually, depth perception can be improved by appropriate lighting. A simplified shading model is employed that only takes the flow's normal component relative to the seeding plane (or tangent plane of the 2D manifold) into account (Figure 4.8(c)). This shading model preserves

the visual fusion for zoomed-out images, similar to the 2D case. In this way, temporal and spatial patterns in the flow can be recognized. For best perception of the tangential flow components, the viewing direction should be close to the direction of the normal of the seeding plane (Figure 4.8(d)). However, there is still the problem of distortions and overlap with neighboring glyphs through to the projection of 3D geometry, especially if the viewing direction is far from the normal of the plane. This affects the perception of patterns in the data. To address this issue, the glyphs are projected onto the seeding plane by neglecting the flow's normal component for the geometry of the glyphs (Figure 4.8(e)). The normal component is now only used for the shading, controlling a factor for adding white color to the color of the glyphs. The result is that glyphs with normal components antiparallel to the viewing direction are lighter, and normal components parallel to the viewing direction make the glyphs darker.

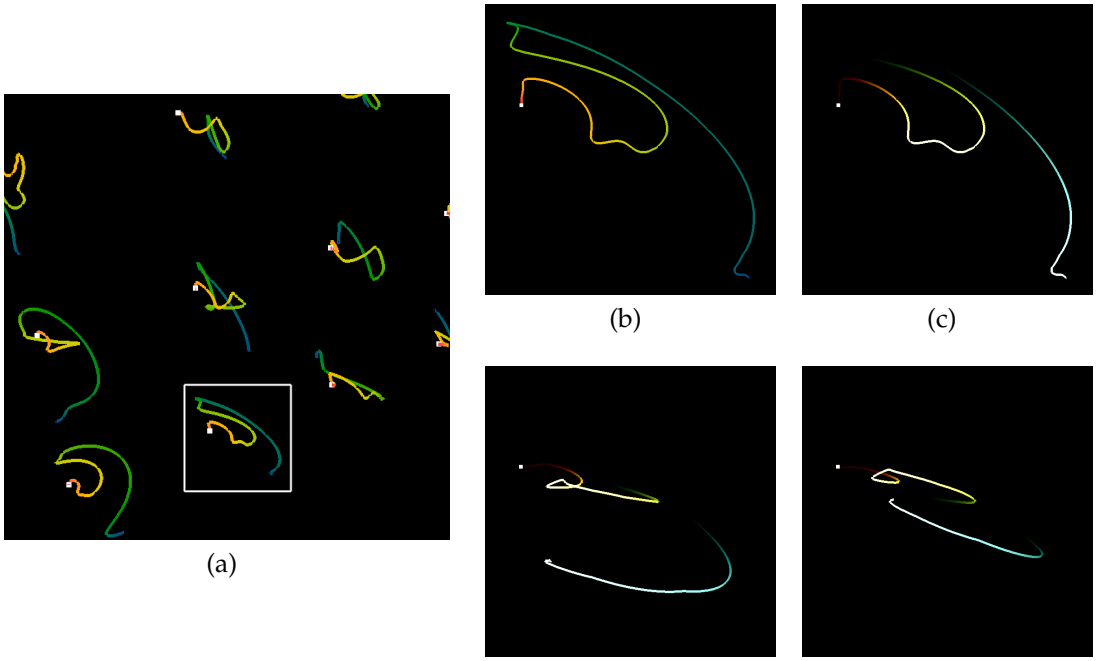


Figure 4.8 — Extension of the glyph to 3D. (a) A 3D version of the glyph is defined by mapping direction to the two angles of spherical coordinates and time to the radius. (b) Close-up of a single 3D glyph (marked in (a)). The glyph is hard to interpret without lighting. (c) Improved depth perception with a simple shading scheme—the normal component relative to the seeding plane controls the shading intensity. (d) The same glyph now viewed along the normal direction of the plane. (e) The glyph projected onto the plane. The shading still allows perception of the normal component.

This 3D version of flow radar glyphs visualizes the directional component of the flow in the plane by geometry and the component normal to the plane through shading. The capabilities of the glyph are retained: patterns are visible on zoomed-out images and close-ups allow a detailed analysis of 3D flow directions (see Section 4.3.6.2).

The approach discussed here is meant as a proof-of-concept showing that it is possible to extend the concept of flow radar glyphs to 3D flow. There is still potential for improvements of the 3D glyph. For example, advanced shading models, shadows, or depth of field could improve the perception of them. However, even with such extensions, typical problems of 3D applications like occlusion and projections issues cannot be completely avoided. Therefore, it is clear that flow radar glyphs work better for 2D applications

4.3.5 Implementation

Flow radar glyphs were implemented with OpenGL, GLSL, and CUDA. The basic implementation is straightforward: it is sufficient to draw the respective curve (Section 4.3.1) for every glyph. Since the glyphs allow a static visualization, their geometry can be pre-computed and only the rendering has to be fast enough to allow interactive navigation (zooming etc.). However, it is beneficial if the user can interactively change certain parameters of the glyphs, like their size or the time range covered by them. In this case, the glyph shapes have to be re-computed fast enough for interactivity, which is more challenging.

Two approaches were implemented for this: a geometry-based approach and an image-space approach. The geometry-based approach renders a geometric curve for every glyph. This allows high visual accuracy even when strongly zooming-in on individual glyphs. To generate the geometry fast enough for interactive updates of the glyph, CUDA is used to compute the vertices of the glyphs in parallel on the GPU and to write them into an OpenGL Vertex Buffer Object (VBO). OpenGL can directly render the geometry in the VBO without additional memory transfer. If the number of glyphs is not too large, this is fast enough to change parameters interactively. For example, each result in Figure 4.10 with 83×83 glyphs covering 320 time steps was computed in less than 50 ms (see Section 1.4 for hardware specification). However, GPU memory consumption is high, in particular for a large number of glyphs or if the glyphs cover a wide time range. A level-of-detail approach could reduce this problem. Beside high accuracy, other advantages of this approach are the easy handling of non-regular seeding schemes and overlapping glyphs.

The image-space approach is similar to local GPU ray casting, a successful generic strategy in computer graphics; see, for example, ellipsoid ray casting

by Gumhold [121]. Bounding geometry is rendered and fragment shader programs are executed for the covered pixels. Inside the shader program, it is checked if the current pixel is part of the glyph. This is done by computing the temporal and directional range covered by the pixel and comparing it to the corresponding range in the data. If these ranges overlap, the pixel is drawn. However, since these computations have to be performed for each update of the screen content, fast computations are required. Therefore, the temporal and directional range covered by a pixel is only approximated resulting in reduced accuracy of the glyphs, which is visible especially when strongly zooming on parts of a single glyph. Additionally, variants of the glyph and of the glyph rendering are quite complex to implement, e.g., rendering magnitude-scaled glyphs or overlapping glyphs. However, there are several advantages of this solution. The rendering time depends only on the output resolution; a huge number of glyphs can still be rendered interactively. Since the glyphs are computed on-the-fly, all parameters of the glyph, like the time range covered, remain interactively changeable. Furthermore, the extension to uncertainty only requires checking a different angular range in the shader.

4.3.6 Results

To demonstrate the capabilities of flow radar glyphs, examples of two different types of data are presented in this section. The first example visualizes 2D data from a turbulent CFD simulation of air in a closed room, which is subject to heating. It demonstrates how to visually compare different simulation runs for parameter studies. The second example is a 3D CFD simulation of an overflowed cuboid that illustrates the extension of the glyphs to 3D. A further example demonstrating the visualization of uncertain vector fields with flow radar glyphs can be found in Section 6.3.2.

4.3.6.1 Two-Dimensional CFD Data and Parameter Studies

This CFD dataset denoted "Hotroom" (uniform grid with a spatial resolution of 41×41 and 320 time steps) results from a 2D finite volume simulation of air in a closed room. The bottom is heated and the top cooled, both with constant temperature. The room is partially divided by a rectangular barrier in the middle of the room.

Single-Parameter Setup In this setup, the temperature boundary condition was 250 °C at the bottom and 5 °C at the top. Figure 4.9(a) shows an overview of the dataset with normalized flow radar glyphs. Large areas of different behavior are visible. Whirls with moving centers are below and above the barrier.

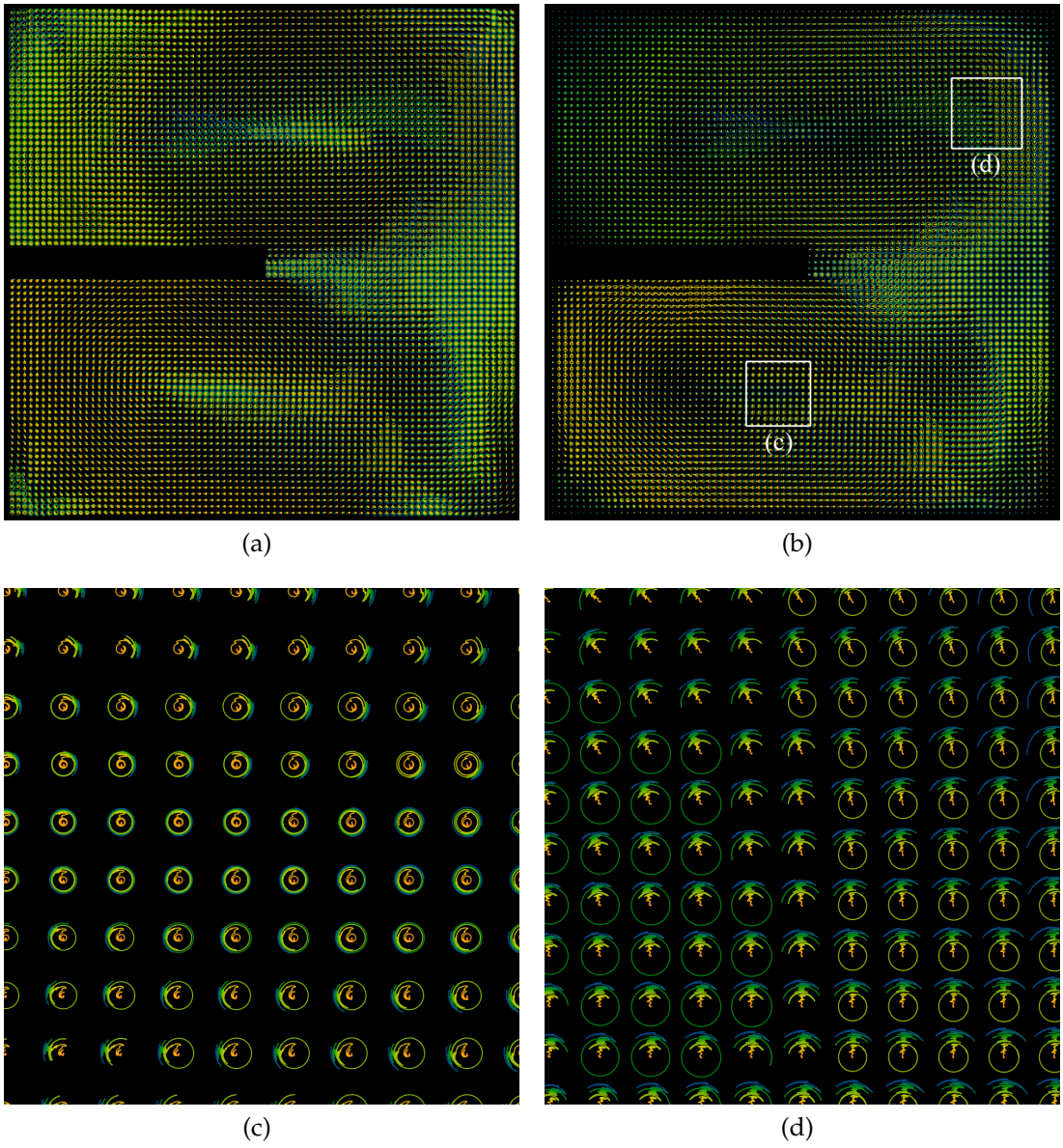


Figure 4.9 — Visualization of the Hotroom dataset for the single-parameter setup. (a) Overview with normalized glyphs. (b) Overview with magnitude-scaled glyphs. (c) Close-up on the center of the lower whirl. (d) Close-up on the region in the top right corner. The close-up regions are marked in (b).

Another area grows from the barrier to the right with extensive directional variations. The magnitude-scaled glyphs (Figure 4.9(b)) additionally depict the variation in flow velocity, e.g., low velocities in the center of the whirls and in the corners.

The close-ups (Figures 4.9(c) and (d)) allow a more detailed analysis of flow behavior. Figure 4.9(c) shows the center of the lower whirl. This area exhibits uniform behavior in the first half of the simulation. After that, the moving center of the whirl induces a significant change of the flow behavior above and below its path: the flow oscillates heavily with an average direction to the right (upper part of Figure 4.9(c)) respectively to the left (lower part). For Figure 4.9(d), a region with two smaller areas in the top right was chosen. A small, separating area between them is visible (Figure 4.9(b)). Left and right from this separating area, a single curl occurs at different points in time. Apart from that, the flow behavior is almost uniform there.

Parameter Study Figure 4.10 shows results for three examples from the parameter study, where the temperature of the bottom was varied, while the temperature on the top was kept unchanged. Comparing the overview images (Figures 4.10(a)–(c)), it can be observed that the structures of the two high temperature results are more consistent and differ qualitatively from the low temperature result. Both high temperature cases exhibit the large areas described in the previous paragraph. The low temperature result exhibits two whirls at the bottom but the changes in flow behavior are overall less intense. In the result for the highest temperature (Figure 4.10(c)), the area of high eddy intensity starting at the barrier extends farther and the flow in the top left corner exhibits more pronounced turbulence. A possible conclusion is that high temperature generates stronger turbulence.

In the close-up areas of the second row, the two lower temperature results exhibit similar behavior: curls followed by constant flow direction. With the highest temperature at the bottom, curls appear only at the left border at the end of the time range. In the third row, the flow for the configuration with lowest temperature exhibits less turbulent behavior than the high temperature configurations. In these two configurations, flow directions toward the barrier are visible. The last row again shows high turbulence for the two high temperature configurations. The flow of the low temperature configuration has only little turbulence there, similar to the region in Figure 4.10(d).

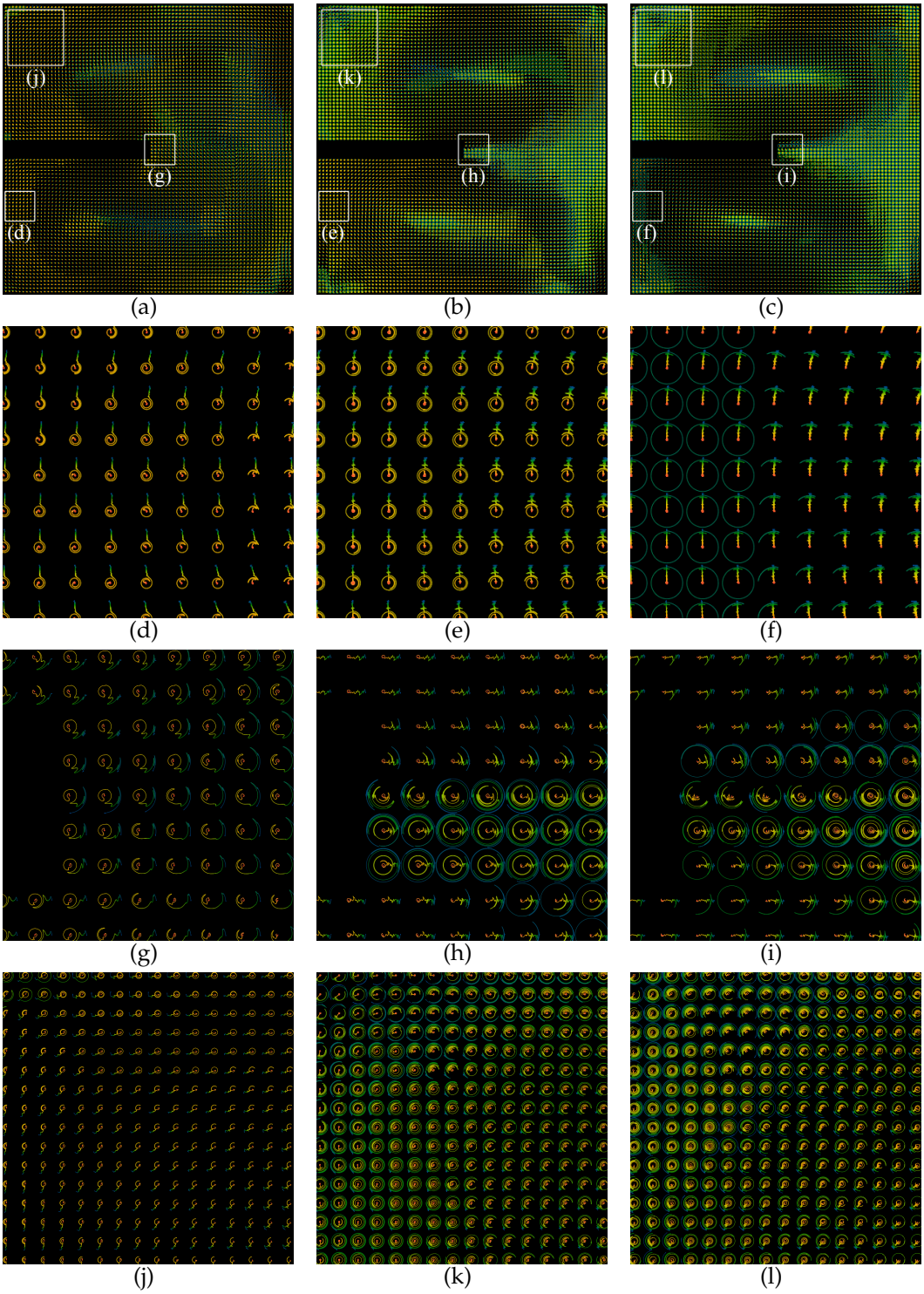


Figure 4.10 — Visualization for the Hotroom parameter study with normalized glyphs. Each column belongs to one configuration of the parameter study with different temperature at the bottom (from left to right: 25 °C, 250 °C, and 500 °C). The temperature at the top was 5 °C for all configurations. The close-ups show the same areas in each row (marked in (a)–(c)): (d)–(f) lower left area, (g)–(i) area at the barrier, (j)–(l) area in the top left corner.

4.3.6.2 Three-Dimensional CFD Data

The dataset denoted "Overflow" results from a 3D CFD simulation of the turbulent flow over a cuboid and is represented on a uniform grid with spatial resolution $121 \times 51 \times 101$ and 70 time steps. The cuboid on the left side (Figure 4.11) is overflowed from left to right. Behind the cuboid, a von Kármán vortex street appears.

The glyphs in Figure 4.11(a) depict the oscillation of the flow direction in the von Kármán street. The region, where the flow is not affected by the cuboid, can be identified. The shading of the glyphs (Section 4.3.4) in the close-up (Figure 4.11(b)) reveals the huge normal component in this area with oscillating directions in the x/z-plane. In Figure 4.11(c), the von Kármán street causes visible patterns right of the cuboid. The dark shading of the glyphs near the cuboid indicates normal components parallel to the viewing direction. The downward direction of the flow behind the cuboid (see Figure 4.11(a)) is the reason for this. The oscillating directions in the x/z-plane are visible in detail in Figure 4.11(d).

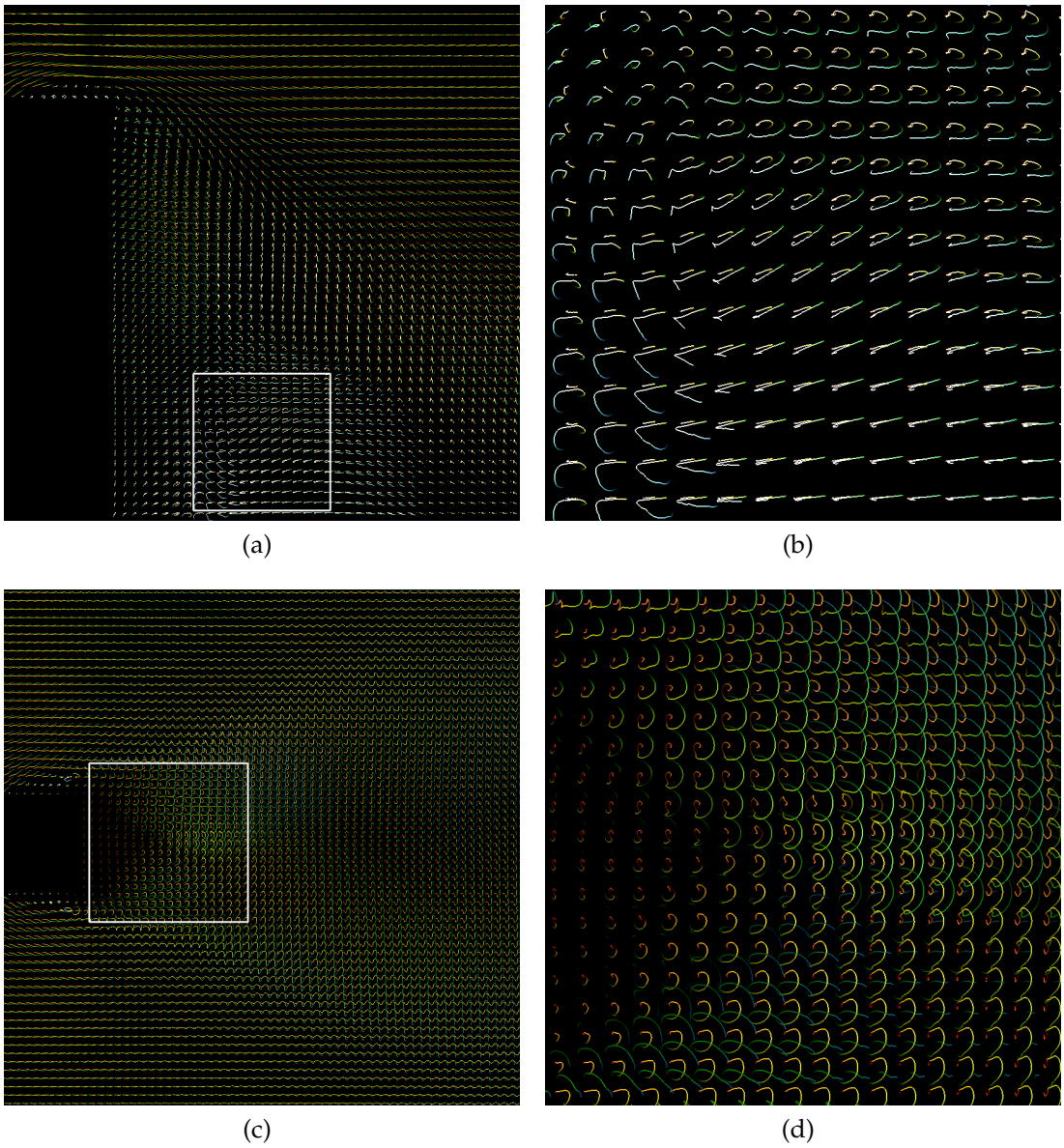


Figure 4.11 — Visualization of the Overflow dataset with magnitude-scaled 3D flow radar glyphs regularly seeded on a plane. (a) Zoomed-out image of glyphs seeded on a plane in x/y-direction. (b) Close-up of the region marked in (a). (c) Zoomed-out image of glyphs seeded on a plane in x/z-direction. (d) Close-up of the region marked in (c).

4.4 Pathline Glyphs

The previously described flow radar glyphs (Section 4.3) provide an Eulerian view on unsteady flow. However, to analyze transport processes, it is important to visualize the flow in a Lagrangian way, e.g., with pathlines. One of the main issues with pathline visualizations is that they quickly become cluttered with an increasing number of lines—amongst others due to intersection. Interactive exploration of the flow field with a low number of pathlines reduces these issues, but analyzing the complete domain with this approach manually is time-consuming and does not provide a comprehensive view of the flow.

The technique presented in the following improves pathline visualization of unsteady 2D flow by "uncluttering" it. The approach encodes the same data as a direct visualization by an arrangement of pathlines, but with less visual clutter. This is achieved by using downscaled pathlines as glyphs—*pathline glyphs*. Furthermore, similar to flow radar glyphs (see Section 4.3.2), the resulting visualization allows the analysis on multiple scales (Section 4.4.1.2).

4.4.1 Concept

To achieve a pathline-based visualization with less visual clutter, pathlines are transformed into glyphs—pathline glyphs—by downscaling each of them (Figure 4.12). To this end, the domain is subdivided into cells, each representing

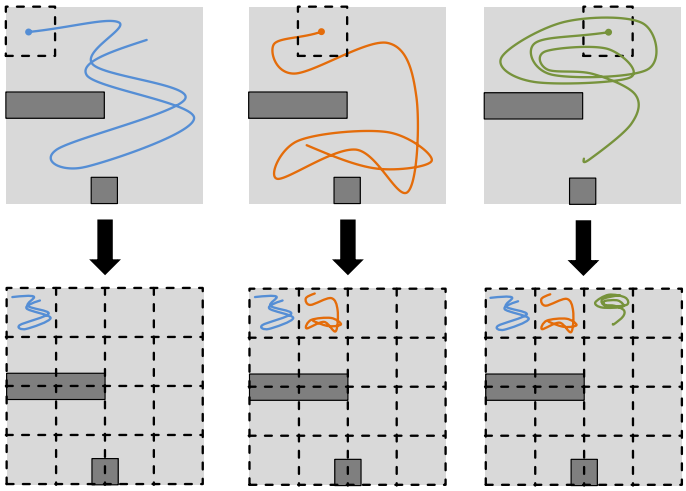


Figure 4.12 — Basic concept of pathline glyphs. The domain is partitioned into non-overlapping cells (dashed). Each cell represents the full domain and contains a single downscaled version of a pathline (colored).

the domain and containing a single pathline. The shape of the pathline glyph inside each cell corresponds to the shape of the original, non-scaled pathline in the entire domain. Using non-overlapping cells avoids intersection of different lines; only self-intersection is possible, which is still allowed because the shape of individual pathlines should be preserved.

4.4.1.1 Seeding

Like for all glyph-based visualizations, different seeding strategies are possible. Here, the focus lies on regular seeding grids because they are widely applied and easy to interpret. For regular seeding, a straightforward choice would be to use the center of each cell. However, this would be problematic, as Figure 4.13

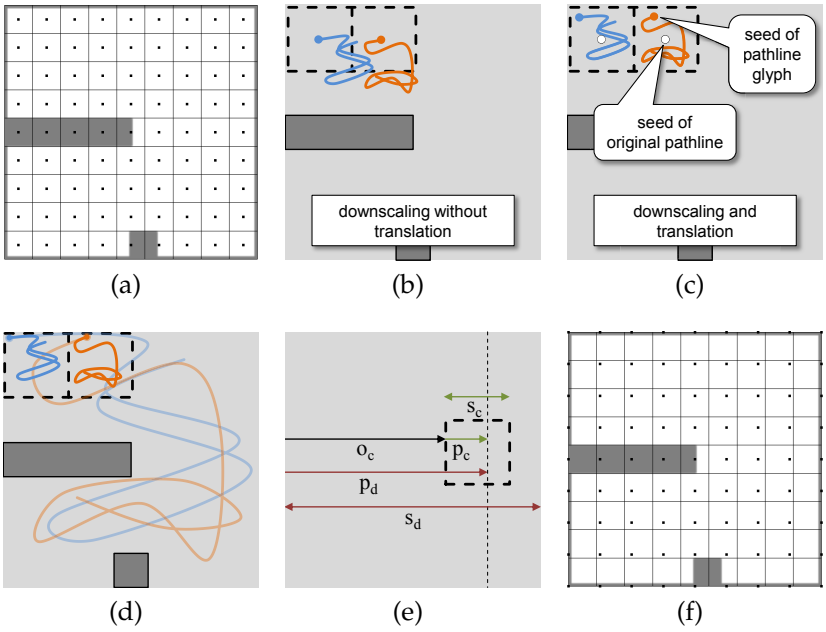


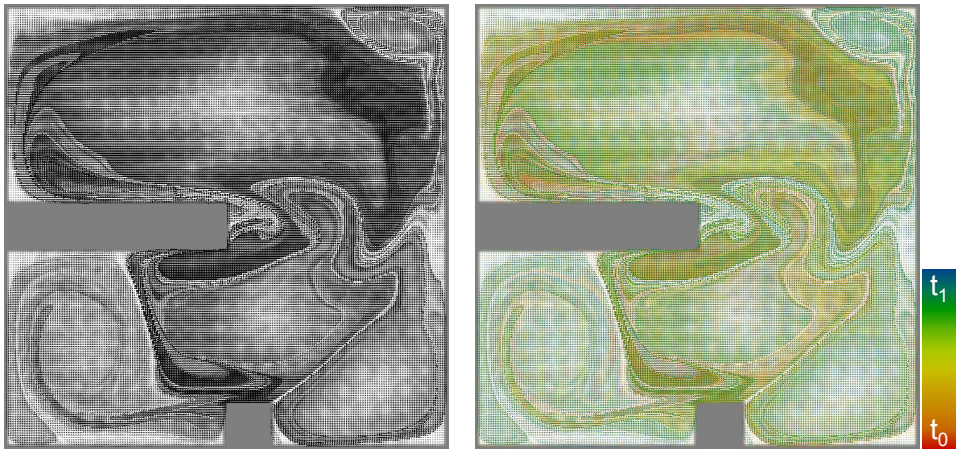
Figure 4.13 — Seeding strategies. (a) Using the center of each cell as seed point and downscaling the pathline without translation would result in an inconsistent visualization: (b) the cell would not represent the domain with respect to the downscaled pathline. (c) The respective translation would restore the alignment with the cell, but the seed point of the pathline glyph would not match the seed point of the original, non-scaled pathline. (d) The goal is now that the seed point of the pathline glyph is identical to the seed point of the original pathline (low saturation). (e) This requires that the relative position with respect to the domain (p_d/s_d) equals the relative position inside the cell (p_c/s_c). For simplicity, only the x-coordinate is illustrated; the scheme is identical for all coordinates. (f) Resulting seed points used in the approach.

illustrates. Simply downscaling the pathline would lead to an offset of the line with respect to the cell (Figure 4.13(b)) and an additional translation would introduce inconsistency between the seed points of the pathline glyph and the original pathline (Figure 4.13(c)). This would make their correct interpretation difficult, especially when combining the glyphs with interactively seeded pathlines (Section 4.4.2.2).

For these reasons, seed points are used that have the same relative position in the domain *and* the respective cell (Figure 4.13(d)). The respective equations are derived only for one dimension because they can be directly applied to the other dimensions as well. To obtain such a seed with identical relative position, the ratio between the position p_d in the domain with size s_d and the position p_c inside the cell with size s_c must be equal: $p_d/s_d = p_c/s_c$ (Figure 4.13(e)). Furthermore, the position inside the domain must equal to the sum of cell offset o_c and cell position p_c, i.e., $p_d = o_c + p_c$. Deriving $p_c = s_c(p_d/s_d)$ from the first equation and inserting this in the second leads to $p_d = o_c + s_c(p_d/s_d)$. Solving this for p_d results in $p_d = \frac{o_c}{1-s_c/s_d}$. The resulting seed points are shown in Figure 4.13(f). It can be seen from the last equation that the offset between neighboring points is constant.

4.4.1.2 Visual Signature on Different Scales

Patterns are visible in densely seeded pathline glyphs due to their variation in local shape and, thus, texture (Figure 4.14(a)). This effect can be enhanced

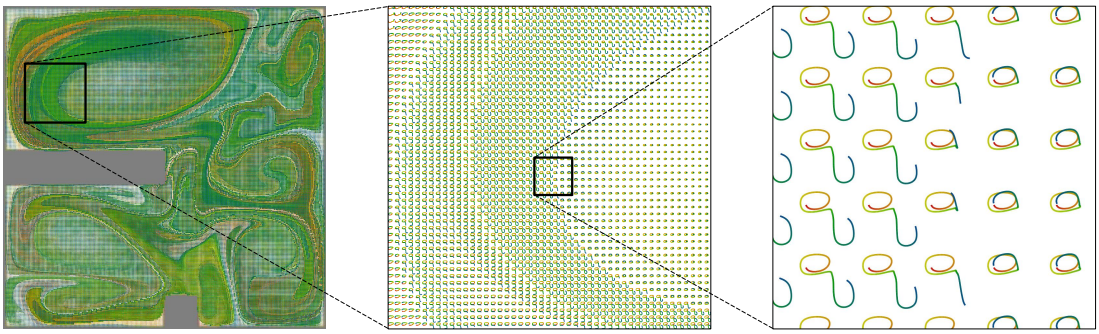


(a) (b)

Figure 4.14 — (a) Patterns in densely seeded pathline glyphs. (b) Mapping integration time to color enhances these patterns. The color map shown is used for all images with pathline glyphs, unless otherwise noted.

by color mapping. Similar to the previously introduced flow radar glyphs (Section 4.3.1), color encodes by default the integration time (between t_0 and t_1) along the pathline glyph (Figure 4.14(b)). In contrast to flow radar glyphs, which employ a radial mapping of direction and time, pathline glyphs have no constraints regarding their shape besides their overall size, e.g., they can exhibit self-intersecting lines. Furthermore, they do not represent quantities at fixed positions in space. Therefore, color mapping is an important enhancement of pathline glyphs facilitating their interpretation. With the integration time mapped to color, the time span that is covered by the represented pathline can be seen, e.g., if pathlines stop at domain boundaries. This exhibits similarities to the level set method by Westermann et al. [300]. However, other encodings are possible as well (Section 4.4.2.3).

It becomes apparent that densely seeded pathline glyphs provide a multi-scale visualization of the flow data (Figure 4.15), another similarity to flow radar glyphs (see Section 4.3.2). On the overview scale, they make salient patterns in the data visible (Figure 4.15(a)). Regions with pathlines of similar behavior, both with respect to space and time, can be identified. Zooming-in allows analyzing the respective flow behavior (Figure 4.15(b)). In this example, a region with pathlines of compact shape can be detected on the right, but more complex glyphs are also visible. Zooming-in further provides a detailed view on pathlines of two types of time-dependent flow behavior (Figure 4.15(c)):



(a) (b) (c)

Figure 4.15 — Multi-scale visualization with pathline glyphs. (a) Emergent patterns visible on the overview scale correspond to structures in the flow and provide the basis for detailed exploration, but individual pathline glyphs cannot be perceived. (b) On an intermediate scale, patterns remain present, but the shapes of the glyphs become visible. (c) A close-up facilitates detailed analysis of time-dependent transport: particles started in the left area later move downward (blue), while particles from the right area stay in the whirl.

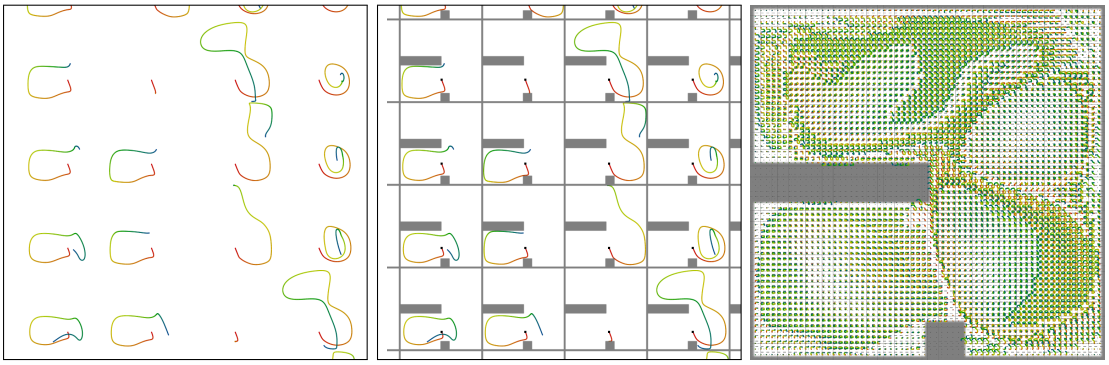
while those on the right are part of a whirl over the entire time range, those on the left leave the whirl later in downward direction. This example shows that pathline glyphs do not only work on different scales but also reveal qualitatively different information on them.

4.4.2 Extensions

Even though the presented concept reduces the visual clutter in pathline-based visualization and enables flow analysis on multiple scales already, several extensions can further complement the work with pathline glyphs.

4.4.2.1 Domain Grid

The shape of a pathline glyph is given by the corresponding pathline. Hence, the glyph shows the trajectory of a particle started at the respective seeding position. However, displaying just the glyphs makes it difficult to interpret the trajectory with respect to the domain (Figure 4.16(a)). This can be improved by additionally displaying information about the domain and the seeding grid. This supports the understanding of single glyphs (Figure 4.16(b)). The drawback is that this can interfere with the perception of patterns on the overview scale (Figure 4.16(c)). Therefore, the best approach is to display the grid only on zoomed-in scales.



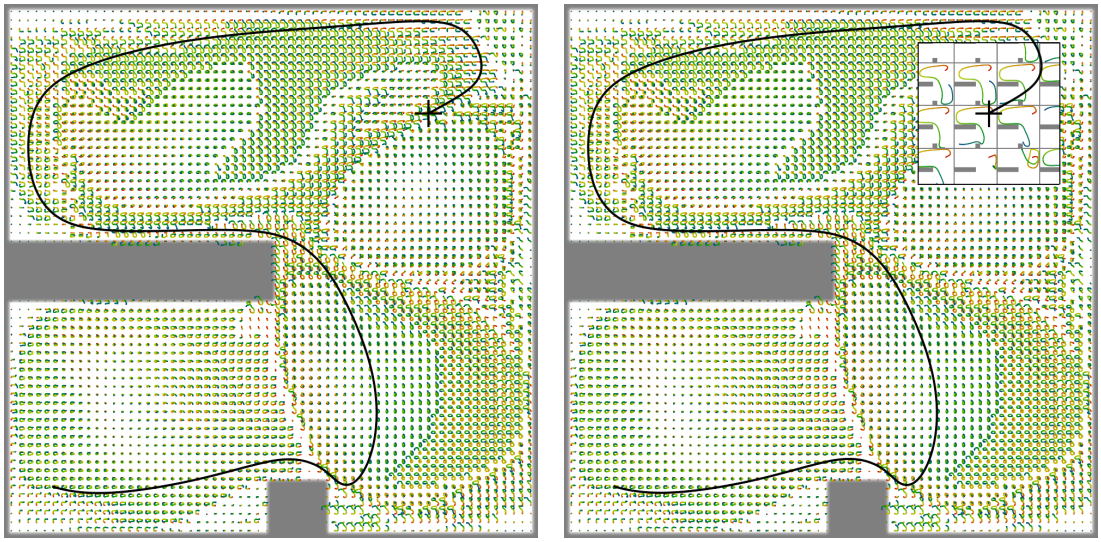
(a) (b) (c)

Figure 4.16 — Additional visual elements to support the understanding of pathline glyphs. (a) Displaying pathline glyphs alone makes it difficult to interpret the pathlines with respect to the domain. (b) Displaying the boundaries of the data domain together with the respective seed points allows an easy understanding of the shape of the pathline with respect to the domain. (c) However, this interferes with the recognition of patterns on the overview scale.

4.4.2.2 User-Controlled Pathline and Zoom Lens

Working with pathline glyphs typically requires frequently changing between scales: identifying interesting patterns on the overview scale, and then analyzing them in detail on a zoomed-in scale. Additional interaction techniques can make this more efficient. Displaying additional full-sized pathlines that can be interactively controlled allows analyzing single pathlines without leaving the overview scale (Figure 4.17(a)). A region of interest can be explored by interactively seeding these pathlines. However, this alone would end up in a trial-and-error approach because the glyphs are usually too small on the overview scale to provide hints for the exploration. Zooming-in would not help because parts of the pathline can be located outside the zoomed-in area.

A solution for this problem is a zoom lens showing the area around the seed point of the pathline (Figure 4.17(b)). This provides the local context of the pathline and allows a more efficient exploration. It has to be noted that due to the pathline glyphs, the zoom lens provides the local context over the full time range. This can typically not be accomplished when using a zoom lens for classic pathline visualization. In that case, many neighboring pathlines could leave the zooming area and only an initial part of the time range would be



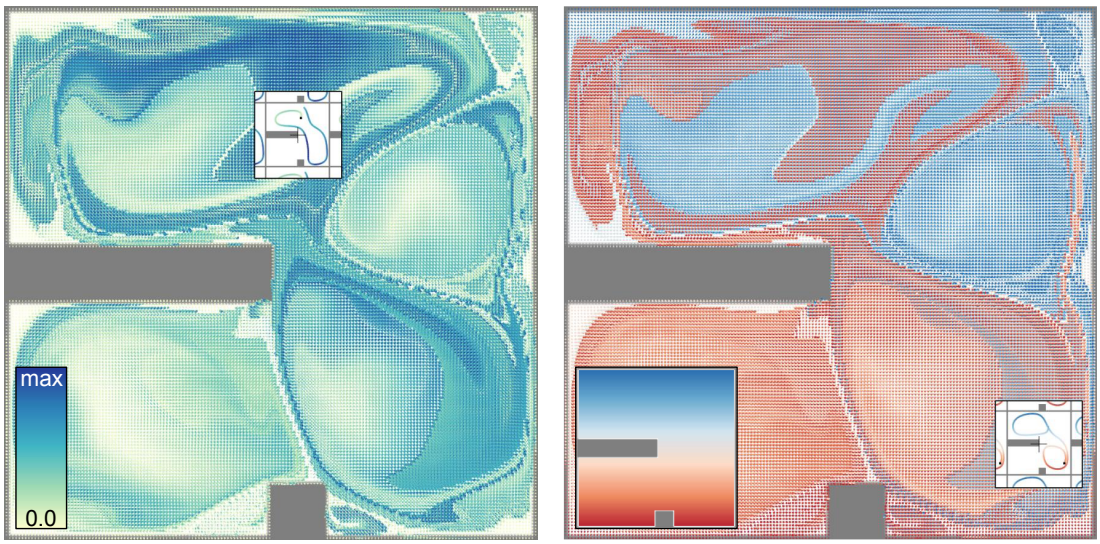
(a) (b)

Figure 4.17 — Interaction for pathline glyphs. (a) An interactively controlled, full-sized pathline (black curve) enables the analysis of individual pathlines without leaving the overview scale. (b) An additional zoom lens, magnifying the area around the seeding point of the interactive pathline, provides the local context of the pathline.

visible. Figure 4.16 shows that an additional grid can help analyze pathline glyphs. However, it interferes on the overview scale (Figure 4.16(c)). Showing the grid only in the zoom lens is a good solution for solving this issue.

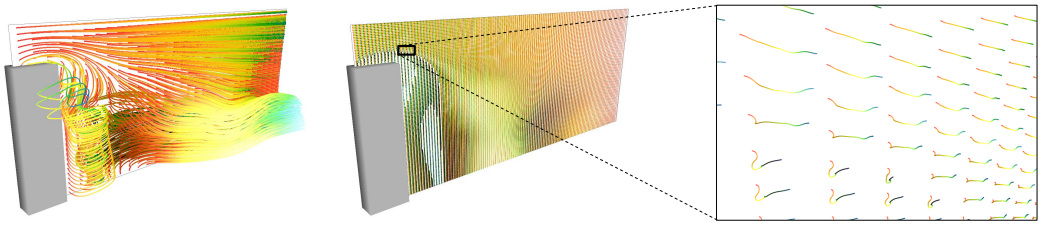
4.4.2.3 Quantity Encoding

So far, color coding with respect to time was employed (Figure 4.14(b)). This is the default color coding for pathline glyphs because the temporal structure of pathlines is important for the analysis of unsteady flow. Nevertheless, other quantities can also be mapped to color (Figure 4.18), depending on the underlying problem. For instance, if areas of high velocity are of interest, it can be helpful to map flow velocity to color (Figure 4.18(a)). In that example, pathlines started in a large area in the upper part of the domain exhibit high velocities. However, if the spatial context of the pathline is of interest, e.g., to analyze transport phenomena, the spatial position can be mapped to color (Figure 4.18(b)). The image shows that large groups of pathlines travel mainly in the area where they were seeded.



(a) (b)

Figure 4.18 — Possible encodings of quantities. (a) Velocity magnitude along the pathlines can be mapped to color. This mapping allows detecting regions that give rise to pathlines with high velocities (dark blue). Looking at individual glyphs, the velocity variation along the pathline can be seen. (b) Another possibility is a spatial mapping. In this example, the y-coordinate is represented with the color map (bottom left). In this case, the glyph visualization shows in which area the pathlines travel most frequently.



(a) (b) (c)

Figure 4.19 — Extension of pathline glyphs to 3D demonstrated with the Overflow dataset (Section 4.3.6.2). (a) Traditional pathline visualization of 3D data suffers additionally from visual clutter due to projection, even when the lines are only seeded on a plane. (b) Pathline glyphs also reduce visual clutter in this case. Besides the pattern from the color mapping, the shading according to the distance to the seeding plane induces additional patterns. (c) A closer view shows the transition in the marked area. Parts of the upper lines stay in front of the seeding plane (brighter segments), while parts of the lower lines are behind the plane (darker segments).

4.4.2.4 Extension to 3D

The basic concept of pathline glyphs—downscaling pathlines around their seed points—can be easily transferred to 3D applications. However, in this case, the glyph visualization will suffer from typical problems of 3D visualization, including issues with occlusion and depth perception. Furthermore, in contrast to the 2D case, the downscaling cannot avoid visual clutter resulting from projection. To address these issues, glyphs should only be seeded on a plane or surface. This approach is exemplified in Figure 4.19, showing visualizations of the Overflow dataset from Section 4.3.6.2. Shading is used to improve depth perception with respect to the seeding plane: the distance to the plane modulates the brightness of the vertices, i.e., parts of the line in front of the plane become brighter, parts behind the plane darker. A similar, simple shading model was also introduced for the 3D flow radar glyphs in Section 4.3.4.

Seeding traditional pathlines on a plane can already result in a cluttered image (Figure 4.19(a)), because lines that do not intersect can still overlap after projection. Due to the downscaling of the lines, the glyphs can avoid this issue. The image becomes less cluttered and patterns are better visible (Figure 4.19(b)). For example, due to the shading model, changes in the distance of the pathlines to the seeding plane can be seen. Zooming-in allows investigating this behavior in detail (Figure 4.19(c)), revealing that the upper lines stay in front of the seeding plane, while the lower lines are partly behind the plane.

4.4.3 Implementation

A visualization tool for pathline glyphs was developed in C++. CUDA was used for computing the glyphs on the GPU and OpenGL was used for rendering. The used computer system is described in Section 1.4. Every pathline glyph can be computed independently from the others. Hence, parallel computation on GPUs is straightforward. One GPU thread is started for every glyph and employs the 4^{th}-order Runge-Kutta scheme with fixed step size for numerical integration. The GPU implementation allows interactively changing glyph parameters like the starting time, at least for shorter time ranges and lower seeding resolutions. For instance, computing the glyphs in Figure 4.21(a) requires around 200 ms and they can be displayed with 19 fps on the used computer system.

It was possible to reuse existing code for pathline computation on the GPU, which shows how easy it is to implement the approach. It was only necessary to add the proper scaling functionality. The glyphs are stored in two Vertex Buffer Objects, one for the vertex coordinates and one for additional quantities (integration time, velocity etc.) stored as texture coordinates for every vertex. Color mapping is applied in the fragment shader using this information. Hence, the user can switch between different color mappings without re-computation. For the zoom lens, the respective part of the frame buffer is overwritten with the rendering for zoomed-in camera settings. Drawing the glyphs with anti-aliasing improves the visibility of patterns at overview scales but drastically reduces the frame rate. Therefore, anti-aliasing is only applied when there currently is no user interaction.

4.4.4 Comparison and Examples

In the following, pathline glyphs are compared to other flow visualization techniques and temporal analysis of flow with them is exemplified. Furthermore, they are compared with a pathline selection method [297]. Results for more datasets can be found in the original paper [5].

Additionally, pathline glyphs were evaluated with a qualitative user study. A detailed description of the study and its results is presented in the original paper [5]. In summary, the involved domain experts think that the glyphs provide a clear overview and visually separate the flow into different areas, offering a good basis for exploration. They believe that pathline glyphs may help accomplish their work more efficiently.

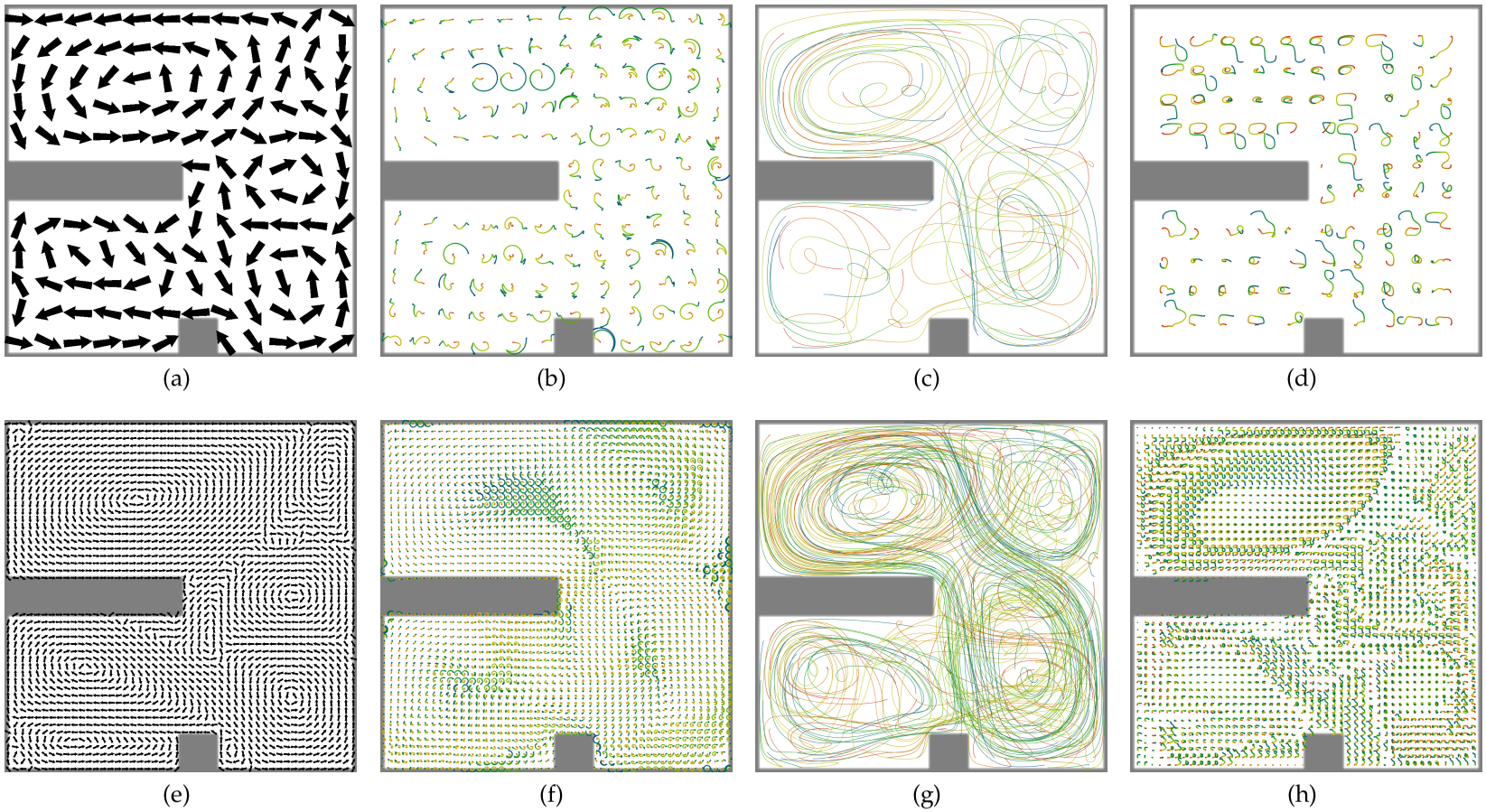


Figure 4.20 — Comparison with other flow visualization techniques for the Hotroom2 dataset. (a) While classic arrow glyphs provide only an instantaneous view, (b) flow radar glyphs (Section 4.3) represent local flow direction over time. In contrast, (c) pathlines and (d) pathline glyphs provide a Lagrangian view on the flow. The lower row ((e)–(h)) shows the respective images for an increased seeding resolution.

4.4.4.1 Comparison to Other Visualization Techniques

Figure 4.20 shows pathline glyphs together with three other flow visualization techniques for the "Hotroom2" dataset (101×101 spatial resolution, 1001 time steps), which represents the simulated buoyant flow in a closed room with heated bottom and cooled top. This is a variant of the Hotroom dataset from Section 4.3.6.1 with two obstacles. In flow visualization, glyphs typically represent local flow direction (Figures 4.20(a) and (b)). Being based on pathlines (Figure 4.20(c)), pathline glyphs (Figure 4.20(d)) provide a complementary Lagrangian view and allow for the analysis of transport by the flow. With increasing seeding resolution, visualization of pathlines typically becomes more cluttered (Figure 4.20(g)), while patterns emerge in the glyph-based visualizations (Figures 4.20(e), (f), and (h)). However, only the patterns in the pathline glyph visualization represent structures from transport processes.

4.4.4.2 Relationship to FTLE

FTLE fields measure the separation of nearby particles in the flow and therefore also represent transport processes in the flow (see Section 4.1.2). Figure 4.21

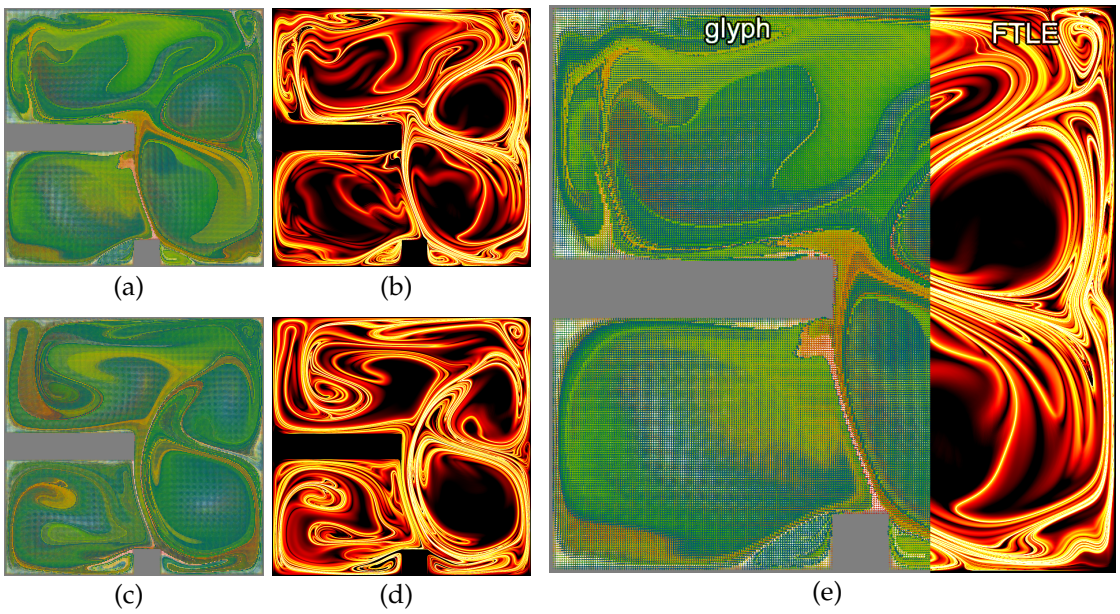


Figure 4.21 — Comparison with the FTLE for the Hotroom2 dataset. Pathline glyphs were computed in (a) forward and (c) backward time (seeding resolution 256×256). The FTLE fields (resolution 2048×2048—(b), (d)) for the same integration parameters show consistent structures. (e) The combined image of (a) and (b) shows that the shape and position of the structures match.

shows a comparison of the structures visible in pathline glyphs and FTLE fields. Many of the structures correspond in shape and position (Figure 4.21(e)). Using temporal color coding results in different colors for pathlines of different length. Neighboring pathlines of different length also correspond to separation and therefore high FTLE values. Hence, an abrupt change in color of neighboring glyphs typically corresponds to high FTLE values. The similarity between the FTLE field and pathline glyphs can be understood from their principal computational approaches: both techniques focus on Lagrangian transport. However, FTLE only takes into account the maximum deviation of end positions, while pathline glyphs show the full trajectory. Therefore, some differences already become apparent at the overview scale. Of course, for more zoomed-in views, the pathline glyphs become very different, as demonstrated in the other examples.

4.4.4.3 Temporal Analysis

By changing the starting time and time range of pathline glyphs, a temporal analysis of flow behavior is possible. This is demonstrated for heat transport in the Hotroom2 dataset. By using pathline glyphs for backward time and spatial color coding (Section 4.4.2.3), it can be seen from and through which areas particles flow. By changing the starting time (Figures 4.22(a)–(f)), it can be analyzed how the mixing of particles from both areas changes over time. Initially, most particles stayed only in one area for most of the time. This is visible in their clearly separable color. At the end of the examined time range, the number of particles passing both areas increased considerably, visible in the mixing of both colors. With increasing time range (Figures 4.22(g)–(i)), the probability that particles travel through both areas increases. This is also visible in the glyph shape (Figure 4.22(l)): some of the particles staying in the whirl in the lower area came originally from the upper area.

4.4.4.4 Pathline Selection

Pathline selection is one approach to reduce visual clutter in pathline visualizations (see Section 4.2.2). Therefore, pathline glyphs are compared with the pathline selection method by Weinkauf et al. [297]. Pathline glyphs were applied to one of the datasets used in their paper (Figure 4.23). The dataset contains the flow above a heated cylinder and has a spatial resolution of 41×70 and 241 time steps.

Their result (Figure 4.23(a)) contains only a few intersecting pathlines and has therefore low visual clutter; it provides a good impression of major pathline structures. However, due to the long pathlines (green lines) starting at the

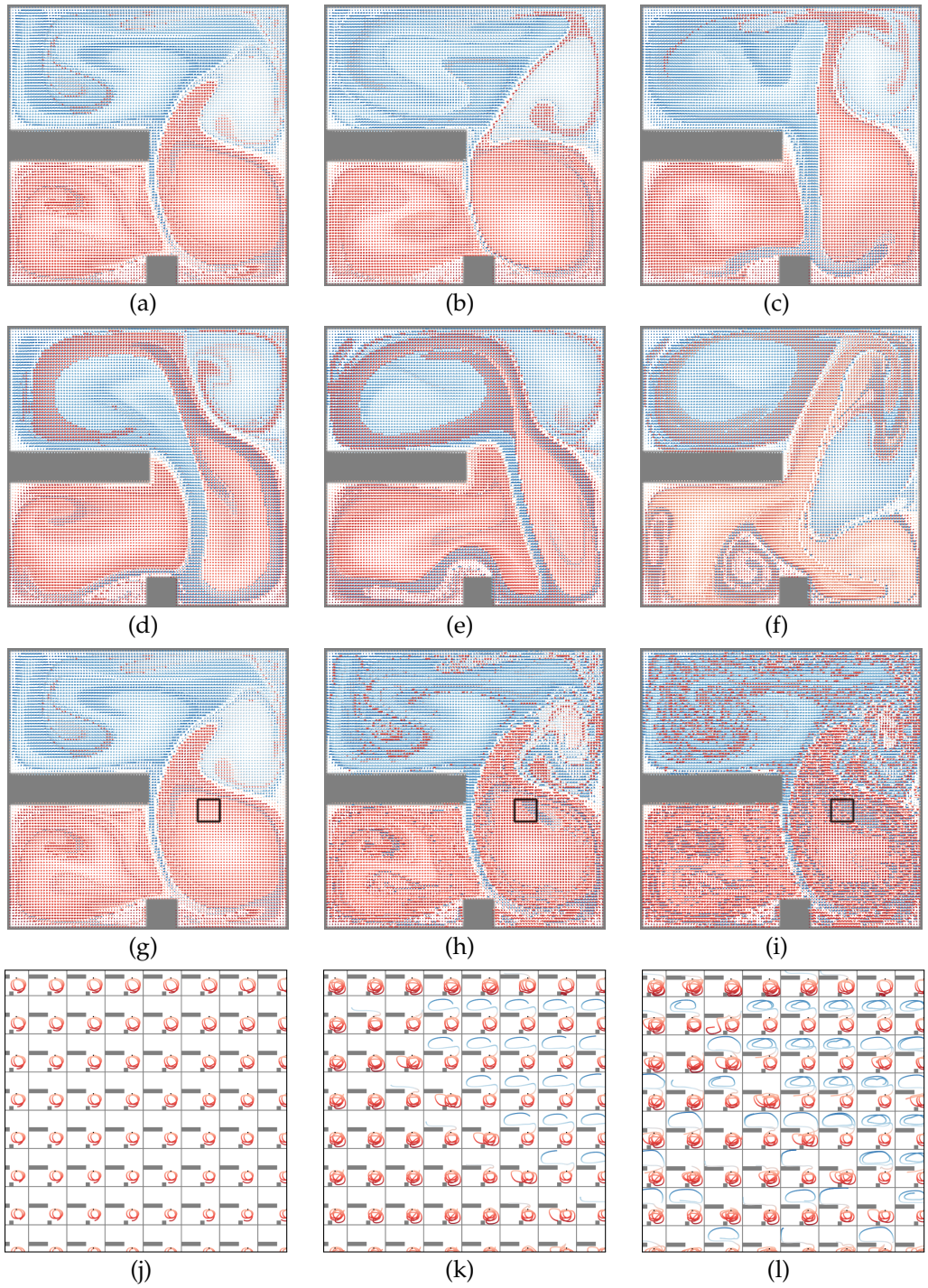
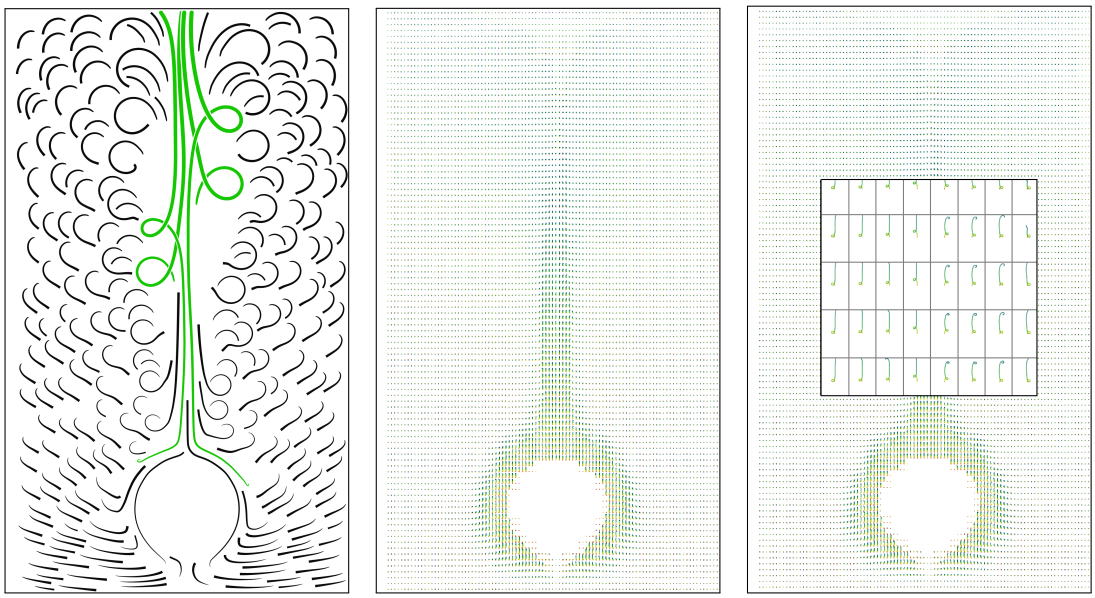


Figure 4.22 — Hotroom2 dataset visualized with pathline glyphs for backward time (seeding resolution 101×101). Spatial color coding is used (Figure 4.18(b)). Figures (a)–(f) show different starting times (0.500–0.625) for the same integration time (0.1). Figures (g)–(i) have the same starting time (0.5), but increasing integration time (0.1, 0.2, 0.3); the marked areas are shown in Figures (j)–(l).



(a) (b) (c)

Figure 4.23 — Comparison to pathline selection. Figure (a) was generated with the method by Weinkauf et al. [297], which tries to select and display only pathlines that do not intersect with others. (b) Pathline glyphs do not require any selection algorithm and can densely cover the domain (seeding resolution 101×101). (c) The zoom lens enables a detailed analysis of pathline behavior.

bottom, the upper part of the center area has very low pathline density. In contrast, pathline glyphs (Figure 4.23(b)) can cover the complete domain with equal density. Two major areas can be identified: the area around and above the cylinder contains long pathlines, while the pathlines in the remaining part of the domain are comparably short. However, zooming-in is required to see the individual shape of the pathlines. The proposed zoom lens (Section 4.4.2.2) enables this without hiding the context (Figure 4.23(c)). For example, the asymmetry of the center area is visible: the pathlines on left side of the zoomed area reach the domain boundary, while the ones on the right exhibit an additional whirl.

4.5 Hierarchical Line Integration

The previous section (Section 4.4) exemplified that the analysis of transport mechanisms and spatio-temporal relationships requires the computation of integral curves. While pathline glyphs already necessitate the computation of a larger number of curves, other techniques exist that compute integral curves for each point inside the domain respectively each pixel of the resulting image. This is the case, e.g., for techniques related to LIC or the FTLE (see Section 4.2). Furthermore, additional computations along the trajectories may be required. For example, in the case of LIC and its variants, an additional noise texture is filtered by convolution along each trajectory.

The computational complexity of all these methods with dense coverage is $O(N^m K)$ for an m-dimensional spatial domain of resolution N. A factor of N^m is due to the dense sampling of the domain. The factor K represents the number of traditional integration steps along each curve, which is often proportional to the image resolution, in particular for LIC. Especially for (time series of) 3D data and long integration times, common for FTLE computation, this complexity behavior leads to long computation times and sometimes even restricts the user to inappropriately low resolutions.

4.5.1 Hierarchical Scheme

In the following, a computation scheme reducing the complexity of computing densely seeded trajectories is described. The basic idea is to exploit the coherence of spatio-temporally proximate portions of these trajectories. In contrast to standard approaches, the computation of redundant parts of curves is avoided by reusing previously computed partial solutions in a hierarchical computation scheme, with each level of the hierarchy doubling the length of the computed curves. In this way, the overall complexity is reduced to $O(N^m \log K)$.

4.5.1.1 Hierarchical Advection

Advection is performed in a hierarchical manner for all points in parallel. Figure 4.24 illustrates the concept for some seeds contributing to the same solution. In the initial step, a short solution is computed for each seed and the corresponding end point is stored at the seed, possibly together with additional quantities evaluated along the trajectory (Section 4.5.1.2). This provides a mapping from initial points to their partial solutions, denoted *coordinate map* in the following. A similar concept was already introduced in Section 4.1.2 with the flow map $\boldsymbol{\xi}_{t_0}^{T}$, mapping start points of trajectories to their end points. However, the coordinate map is not restricted to storing positional

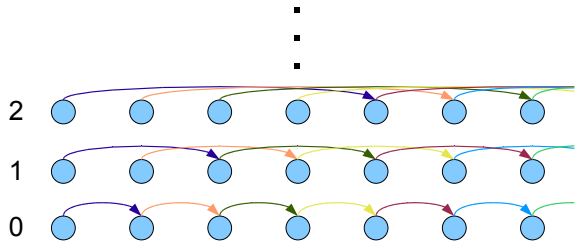


Figure 4.24 — Three levels of the hierarchical computation scheme (with $s = 2$) for points contributing to the same solution. In step 0, short solutions (arrows) are computed from every point. These solutions are stored at their respective starting point. The stored solutions are then used in the next level to construct longer solutions, which are then stored again. This procedure is repeated in the following levels until solutions of designated length are obtained.

information, as it is explained in Section 4.5.1.2 and demonstrated for LIC (see Section 4.5.3.2). Furthermore, the hierarchical advection scheme can also be used to compute flow maps for FTLE computations (see Section 4.5.3.1). In this case, the flow map is obtained by generating several coordinate maps during the hierarchical computation scheme. Therefore, to avoid ambiguities, both concepts are separated in this work by using different denotations.

The coordinate map can be seen as a function

$$\boldsymbol{\xi}^{T_i,i}(\mathbf{x},t) : D \times \mathbb{R} \rightarrow D \times \mathbb{R}, \tag{4.11}$$

mapping a point $\mathbf{x}$ with time t to its end position after advection for time T_i, with i defining the hierarchy level. Typically, $D \subseteq \mathbb{R}^n$ holds for the domain. The advection time T_i increases with every hierarchy level i and the mapping can include additional quantities (see Section 4.5.1.2). Next, a new solution is computed for each point by following this map $s \geq 2$ times, combining the quantities and storing them in the next level of the hierarchy. Subsequent levels are generated by repeating this procedure until the solutions match the prescribed integration range. The construction of the coordinate map for the i^{th} level of hierarchy can therefore be described as $\boldsymbol{\xi}^{T_i,i}(\mathbf{x},t) = (\boldsymbol{\xi}^{T_{i-1},i-1})^s(\mathbf{x},t) = (\boldsymbol{\xi}^{T_{i-1},i-1} \circ \boldsymbol{\xi}^{T_{i-1},i-1} \ldots \circ \boldsymbol{\xi}^{T_{i-1},i-1})(\mathbf{x},t)$, e.g., $\boldsymbol{\xi}^{T_i,i}(\mathbf{x},t) = \boldsymbol{\xi}^{T_{i-1},i-1}(\boldsymbol{\xi}^{T_{i-1},i-1}(\mathbf{x},t))$ for $s = 2$. From this follows that it is sufficient to store only the current level and to overwrite it with the next level to avoid increasing memory consumption. Of course, the zeroth level, $\boldsymbol{\xi}^{T_0,0}(\mathbf{x},t)$, has to be derived directly from the underlying problem. In the case of vector fields, $\boldsymbol{\xi}^{T_0,0}(\mathbf{x},t)$ is computed by integration, i.e., by solving the corresponding initial value problem.

The exponential growth of the integration range with the number of levels in the hierarchy results in logarithmic complexity (see Section 4.5.1.3), leading to a considerable acceleration of line integration. Additionally, it lends itself to parallel execution on multi-core and many-core architectures like GPUs, due to memory locality and independence of subtasks. In contrast to pyramid methods, which construct hierarchical structures with decreasing resolution, the proposed method uses hierarchical levels of constant resolution to increase the accuracy and to obtain results of high spatial resolution.

The coordinate map can provide only a discretized representation of vector fields. Therefore, each level of the hierarchy introduces error due to interpolation when following the coordinate map. A detailed analysis of this error is provided in Section 4.5.1.4. Furthermore, time-dependent mappings, e.g., by pathlines in unsteady vector fields, can be handled in general by adding time as a further dimension. However, a special treatment of the temporal dimension allows reducing the memory requirements of the approach, which is discussed in Section 4.5.2.

4.5.1.2 Hierarchical Evaluation of Quantities

This section addresses methods that require not only the computation of end points of trajectories but also the evaluation of quantities (performing operations) along them. An example of this is LIC, where a texture is convolved with a filter whose support spans the trajectory. The proposed acceleration scheme allows for simultaneous evaluation of such quantities during the hierarchical integration procedure and hence accelerates their evaluation. For this, the quantities are stored with each point in the hierarchical scheme, i.e., in the coordinate map. However, only quantities that can be evaluated in a hierarchical manner are suitable for acceleration by the proposed scheme. In other words, it must be possible to decompose the operation into subtasks and to

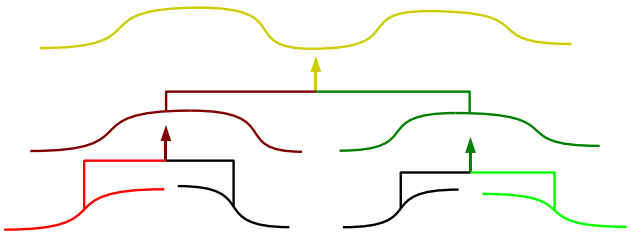


Figure 4.25 — Hierarchical evaluation of quantities—the quantity along a trajectory is computed from the quantities of shorter parts.

hierarchically merge the partial results for the final result (see Figure 4.25). This includes: integration, convolution with specific kernels, computation of average, minimum, or maximum values, and distance computation. In these cases, the same operation is applied to the subparts and as merge operation.

Convolution plays a special role. It relies on a signal along the trajectory, which has to be sampled. The proposed scheme increases the sampling distance and with that the size of the convolution kernel with every level of the hierarchy. Thus, high spatial frequencies must be suppressed by the kernel to avoid aliasing from undersampling. Direct application of the scheme would imply a sampling distance in the next level that equals to the size of the filter kernel in the current level. This might result in a too coarse sampling for the convolution. This issue is addressed by performing the integration of end points and the computation of quantities in a decoupled manner. The quantities are computed by applying the scheme with increased s, i.e., including more samples than actually necessary for end point computation. This allows for sufficiently large kernel sizes, assuring a proper sampling of the signal (see Section 4.5.3.2). Another requirement is that the convolution can be decomposed into multiple subsequent convolutions with appropriate filter kernels. These requirements are met by the Gaussian kernel.

Depending on the quantities and the underlying operations, interpolation has to be performed. Respective accuracy issues are addressed in Section 4.5.1.4.

4.5.1.3 Complexity Analysis

The computation of the end point of a trajectory of integration range r and constant integration step size l requires $i_{\text{direct}} = r/l$ integration steps in a standard approach. In this case, the computational cost depends linearly on r. The number of computation steps in the hierarchical scheme depends on the number of levels h in the hierarchy and the number of concatenation steps s in each level. The concatenation increases for each level the integration range by a factor of s, i.e., the first concatenation results in a range $r_1 = ls$, the second in $r_2 = (r_1)s = lss = ls^2$, and so on. Assuming constant s and l leads to

$$r = ls^h \Rightarrow h = \frac{\log \frac{r}{l}}{\log s}.$$

Then, the total number of concatenation steps i_{hier} is

$$i_{\text{hier}} = sh = \frac{s}{\log s} \log \frac{r}{l}.$$

Using the constant $c_h = s/\log s$ leads to

$$i_{\text{hier}} = c_h \log i_{\text{direct}}.$$

Therefore, the number of computation steps in the hierarchical scheme grows only logarithmically with increasing integration range. It also follows that s has to be chosen larger than one. Furthermore, i_{direct} and i_{hier} have to be rounded to the next integer value. Hence, the minimum number of computation steps with the hierarchical scheme is achieved for $s = 2$.

This analysis shows that the hierarchical method already performs only half of the computation steps compared to the standard approach for $i_{\text{direct}} = 16$. Furthermore, doubling the integration range, i.e., adding another level in the hierarchy, causes only small constant additional computational cost. Of course, a speed-up of the overall procedure of a factor of 2 is achieved in practice only for considerably larger i_{direct}, due to the setup time required for both methods, e.g., for memory allocation. Figure 4.26 documents the computation times for a typical example (see Section 4.5.4 for implementation details). This performance plot shows that, if neglecting the setup time, the standard and hierarchical approaches indeed exhibit the predicted performance characteristics. The plot uses an exponentially scaled x-axis to expose the logarithmic growth of the hierarchical method. Hence, the standard method shows exponential and the hierarchical method linear behavior in this plot. Because of a setup time of around 28 milliseconds in this example, a speed-up of two is reached at an integration range around 13, where 128 computation steps are performed with the standard and 15 steps with the hierarchical method.

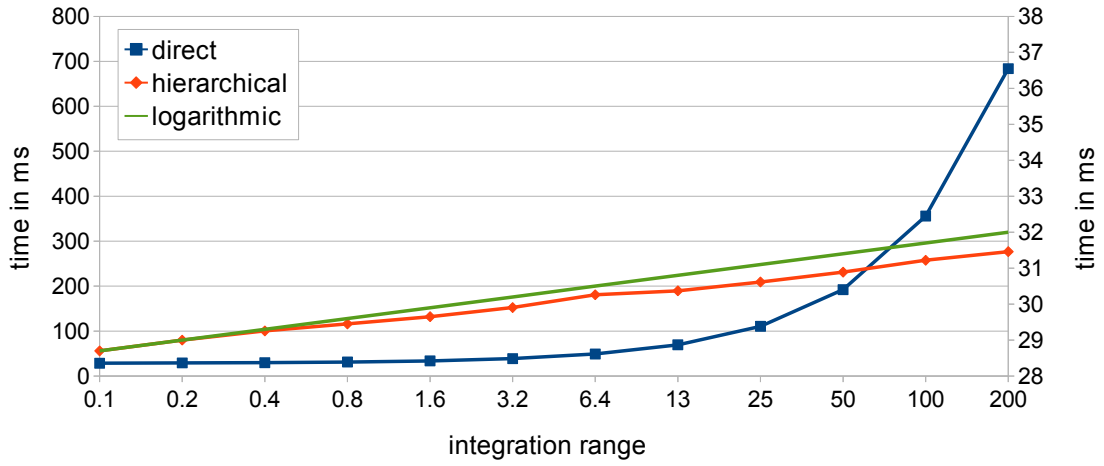


Figure 4.26 — Execution time for an exponentially growing integration range with the standard (blue, left axis) and the hierarchical (red, right axis) method (resolution 512^2 and $s = 2$). As a reference, $0.3 \log_2$ is plotted (green, right axis). The standard method exhibits exponential growth, whereas the time for the hierarchical method grows linearly.

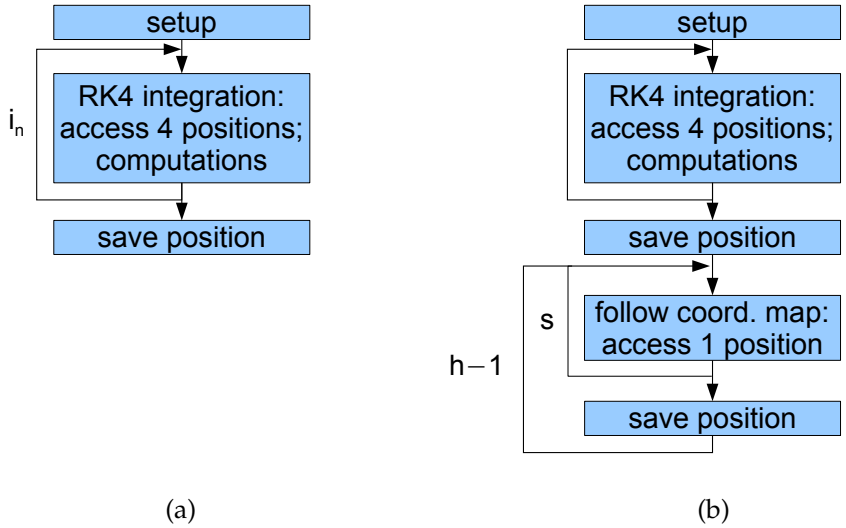


Figure 4.27 — Flow chart of the (a) standard and (b) hierarchical method with fourth-order Runge-Kutta integration. The standard method repeats the same step for the entire computation. Identical steps are executed by the hierarchical method only for the zeroth level of hierarchy. The remaining levels access only single positions in the coordinate map.

It has to be noted that the standard and hierarchical method only partially perform the same computation steps during their computations (Figure 4.27). For the zeroth level, the hierarchical method uses computation steps identical to the standard approach, a fourth-order Runge-Kutta integration scheme in this example. All additional levels need only access to single positions for following the coordinate map, necessitating one interpolation operation only. In contrast, fourth-order Runge-Kutta integration, like most other solvers, needs access to several locations within the vector field, involving respective interpolation operations. This means that the acceleration by the hierarchical scheme can benefit not only from a reduced number of computation steps, but also, depending on the hardware and interpolation scheme used, from reduced effort for the computations in higher levels of the hierarchy.

If the integration range does not align with powers of s, the integration range can be matched by adding an additional level of hierarchy together with a reduced step size in the lowest level. The additional level adds only a small overhead in computation time and the smaller step size increases the accuracy of the hierarchical integration.

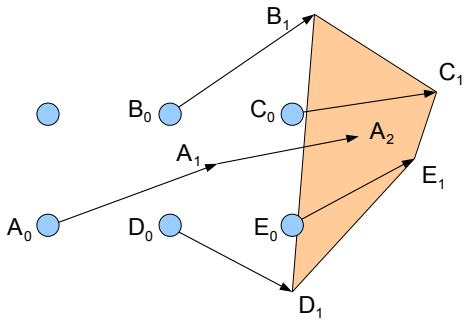


Figure 4.28 — Linear interpolation inside the coordinate map. The coordinate map is defined on nodes of the sampling grid (blue dots). Position A_0 is mapped to A_1 and likewise $B_0..E_0$ to $B_1..E_1$. Because A_1 does not match a grid point, A_2 is determined by bilinear interpolation of $B_1..E_1$.

4.5.1.4 Accuracy

In the general continuous setting, it is unlikely that partial solutions exactly hit nodes of the computational grid. This makes interpolation of the coordinate map necessary (Figure 4.28). Unfortunately, interpolation introduces deviations to the solutions computed by the hierarchical method. A formal derivation of the error order of the integration scheme is presented in the appendix of the original paper [6] and is just shortly summarized in the following. Using tensor-product linear interpolation [100], i.e., bi-linear in 2D and tri-linear in 3D, the error order is asymptotically bounded by

$$|g_n(x_l)| < \frac{c^2 M}{2L} e^{h2L}, \tag{4.12}$$

with $g_n(x_l)$ being the global error at node x_l for h levels of hierarchy, L a Lipschitz constant related to the continuity of the flow map, and M the maximum second derivative of the flow maps of all levels of hierarchy. The relevant result of Equation (4.12) is that the error is second order in the cell size c.

This asymptotic, analytical discussion is backed by a number of exemplary experimental results. Different 2D vector fields were used: three synthetic datasets and one from CFD. Motivated by the Helmholtz-Hodge decomposition, two of the synthetic fields contain only a single whirl or a single source, representing its extreme cases. The third synthetic dataset is the so-called quad-gyre [236]. Finally, the Hotroom dataset (Section 4.3.6.1) is a typical CFD example. To restrict the analysis to the error introduced by repeated interpolation of the coordinate maps, the reference end points for the first two synthetic fields are computed analytically as ground truth, and the initial integration in the zeroth level of hierarchy is also performed analytically for these cases.

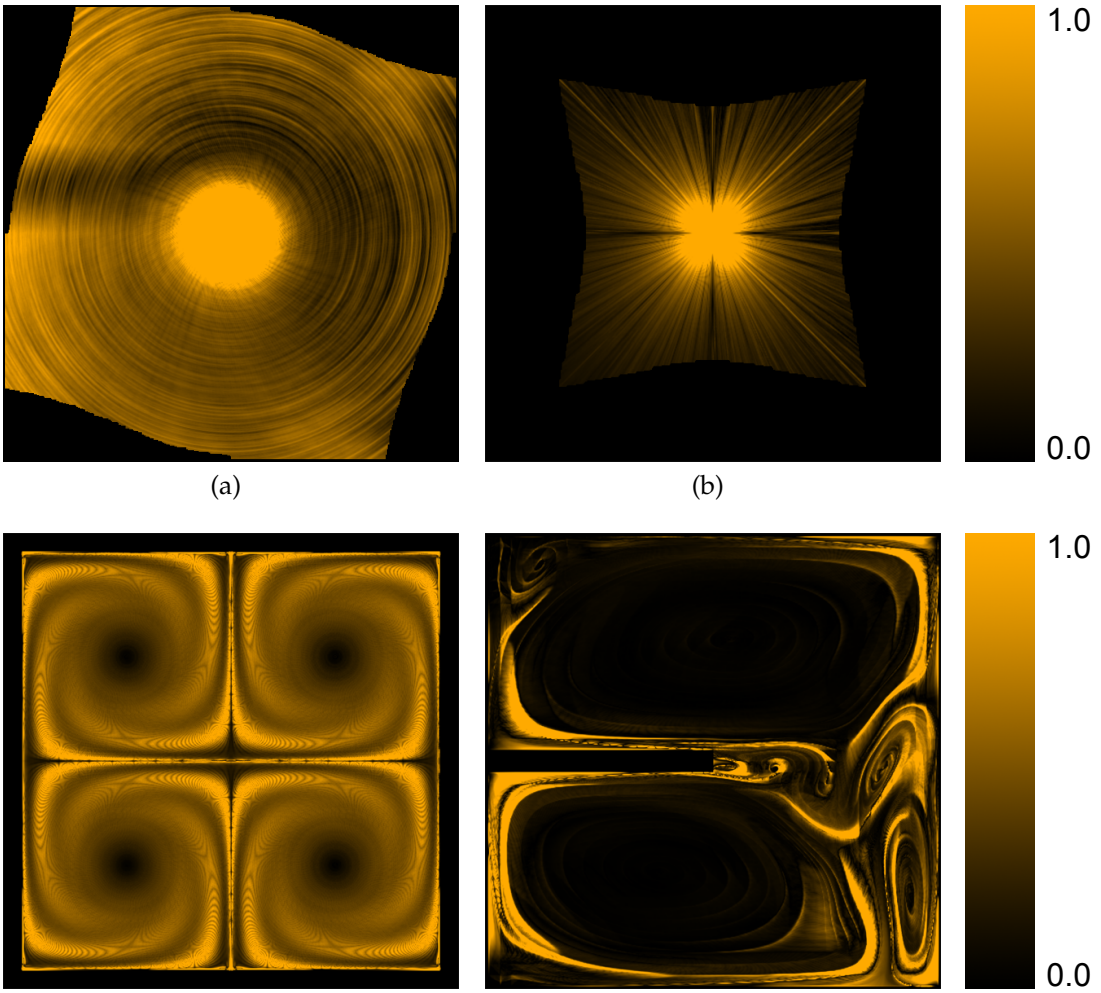


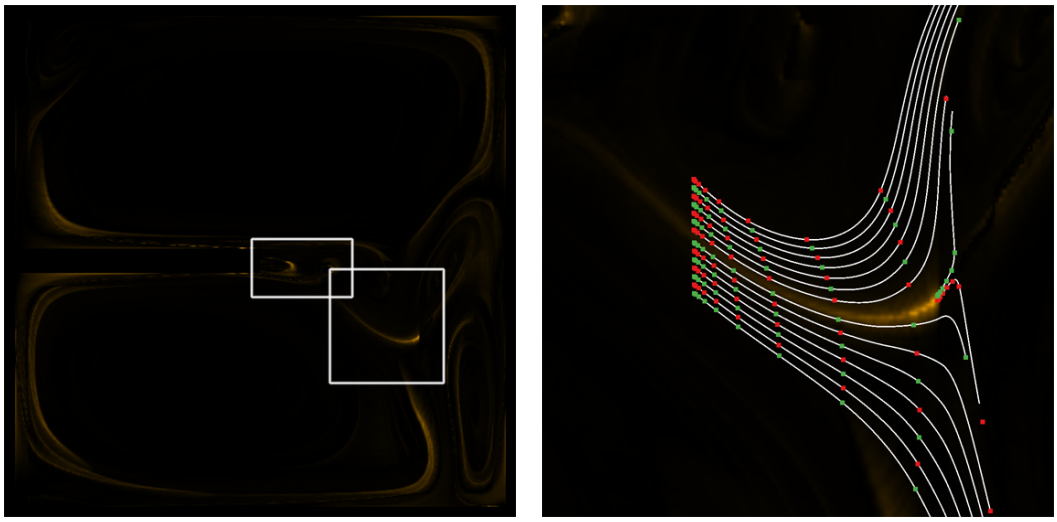
Figure 4.29 — Deviations in hierarchically computed end points (resolution 512^2, 1024 integration steps) for (a) whirl, (b) source, (c) quad-gyre, and (d) Hotroom dataset. The error is normalized with its 95^{th} percentile, which is 0.0023% for the whirl, 0.0012% for the source, 0.0149% for the quad-gyre, and 0.0983% for the Hotroom dataset. A black–orange color map is applied, with orange corresponding to the 95^{th} percentile of the error. End points lying outside the dataset domain were rejected from the analysis and colored black. The maximum error is 1.2386% for the whirl, 0.0490% for the source, 0.1594% for the quad-gyre, and 1.2320% for the Hotroom dataset. The average trajectory length (as percentage of the domain size) is 20.48% for the whirl (step size 0.0002), 20.48% for the source (step size 0.0002), 54.18% for the quad-gyre (step size 0.5), and 31.58% for the Hotroom dataset (step size 0.1).

Figure 4.29 shows the error distribution for these vector fields. The error is measured as the Euclidean distance between each reference end point and the corresponding hierarchically computed end point. The vector field domain has unit size so that the Euclidean distance can be directly interpreted as distance relative to the extents of the domain. Since the errors of hierarchical integration are small, the error values are scaled in each plot independently for appropriate visualization: they are normalized so that the color map covers error values from 0 to the 95^{th} percentile. The maximum error for the whirl example (Figure 4.29(a)) occurs at the center, which can be explained by the increasing curvature of streamlines toward the center. The error for the source vector field (Figure 4.29(b)) is much more angle-dependent, but also with highest error at the center. This results from issues with the sampling grid: streamlines are crossing more grid nodes for certain angles, reducing the error introduced by interpolation. In the quad-gyre example (Figure 4.29(c)), high errors originate from the borders around the four whirls. This is also the case for the Hotroom dataset (Figure 4.29(d)); high errors occur mainly near the borders.

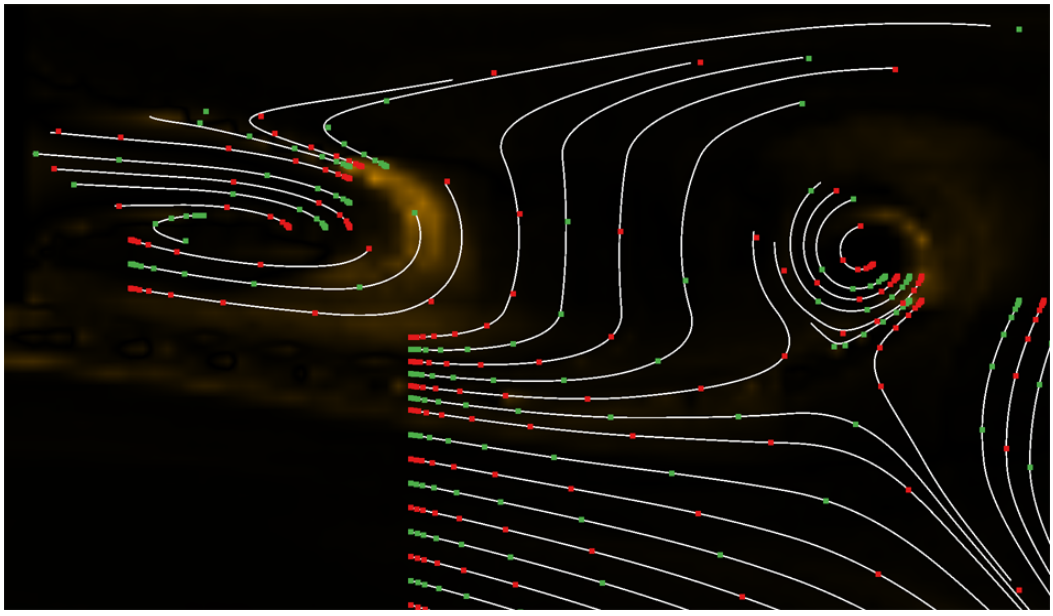
Figure 4.30 shows a visual comparison of the integration with the standard and the hierarchical method. To avoid visual clutter, only selected results for areas where high deviations occur are visualized. The images show that at most positions with high error, only the last hierarchically computed point deviates substantially from the end point of the reference curve. Still, these end points tend to lie on the reference curves or their continuations. The hierarchical method approximates the direction of the curves accurately in these cases; however, the velocities along the curves tend to be strongly non-linear and are hence approximated with increased error.

In addition to the error images, a quantitative analysis of the overall error by means of the root-mean-square error (RMSE) and the 95^{th} percentile of the Euclidean distance error was performed. Figure 4.31 shows the error with respect to the resolution of the computational grid. According to this plot, doubling the resolution in each dimension reduces the error approximately by a factor of four. This means that hierarchical advection indeed converges quadratically to the true solution with increasing grid resolution, as predicted by the analytical error bound from Equation (4.12) (page 116). Therefore, numerical accuracy can be balanced with computational and memory costs by adjusting the resolution of the computational grid.

Figure 4.32 additionally shows the error with respect to the integration range. The error depends almost linearly on the integration range and a higher resolution of the computation grid results in a lower slope of the error curves, i.e., the error growth with respect to the integration range is lower.



(a) (b)



(c)

Figure 4.30 — Visual comparison of integration in the Hotroom dataset. The zoomed areas ((b) and (c)) are marked in the overview image (a). The white curves are generated with direct integration. The hierarchical method cannot generate a full curve, but only end points; hence, only the end points of every level in the hierarchy are visualized as dots alternately colored red and green for better differentiation of neighboring curves. The lines and dots are seeded in areas with high deviation to demonstrate how the high end point deviation is related to the complete curve geometry. The differential images in the background are normalized with respect to the maximum error, which is 1.2320% in this dataset.

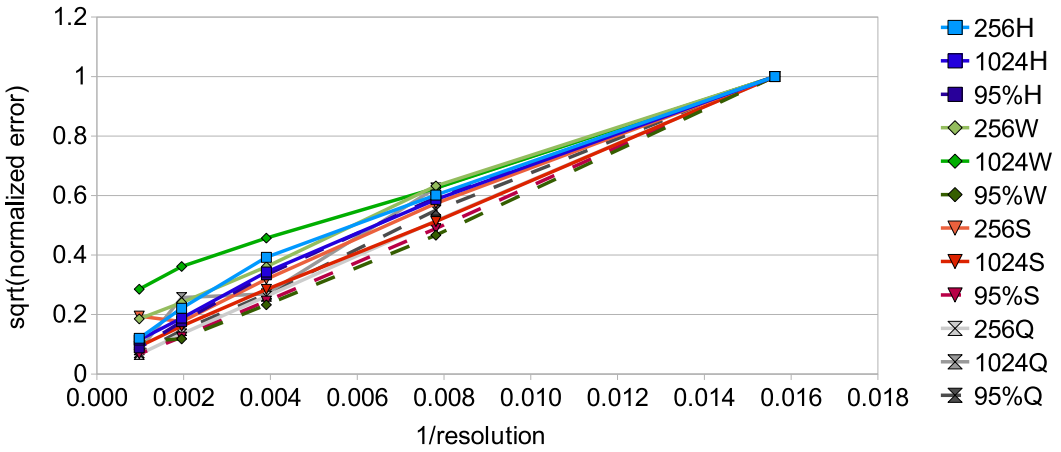


Figure 4.31 — Square root of the normalized error in dependency on the reciprocal of the resolution in one dimension, for the Hotroom (H), whirl (W), source (S), and quad-gyre (Q) dataset. RMSE for an integration range of 256 and 1024 steps and the 95th percentile of the error for the 1024 case were measured. The error was normalized to the error at reciprocal resolution 1/64.

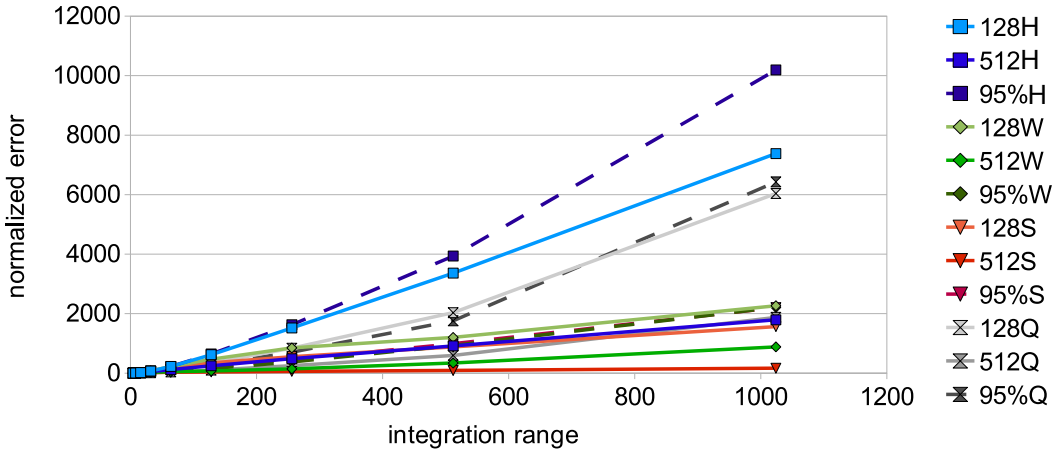
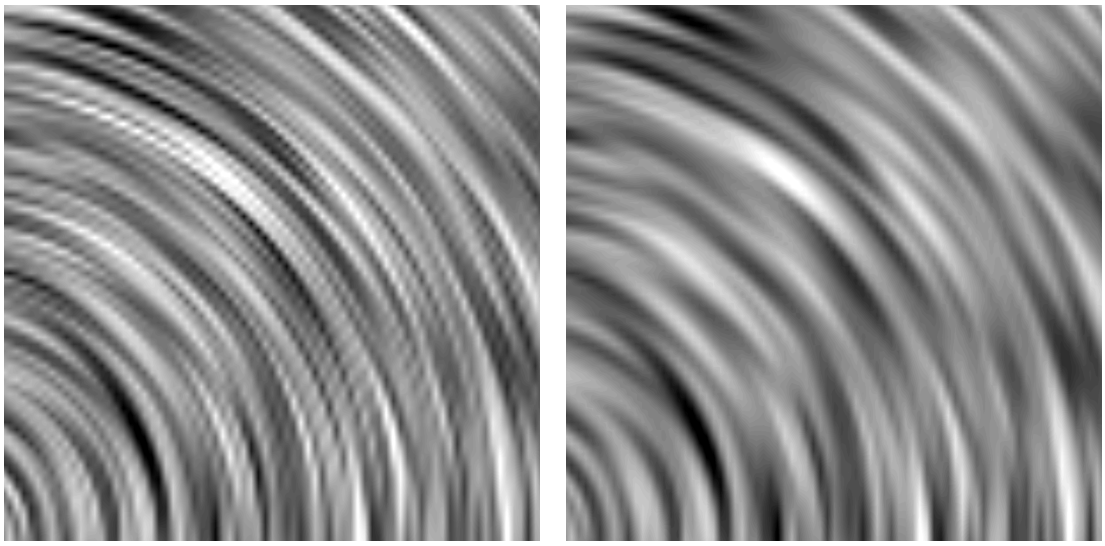


Figure 4.32 — Normalized error in dependency of integration range. The error was normalized to the error of the shortest range. RMSE at resolutions of 128^2 and 512^2 were measured for the Hotroom (H), whirl (W), source (S), and quad-gyre (Q) dataset. Additionally, the 95th percentile of the error for a resolution of 128^2 was measured.

The computation of quantities along trajectories requires interpolation as well, and thus, accuracy issues also occur in this context. An LIC example is shown in Figure 4.33. Here, the normal LIC implementation produces aliasing artifacts caused by (for this purpose intended) undersampling of the noise texture. These artifacts are suppressed by the hierarchical LIC method due to multiple resampling of the noise. However, high frequencies are removed not only along the trajectories, but also perpendicular to them, which reduces the level of detail of the LIC image. In general, LIC performed with the hierarchical method is less sensitive to undersampling artifacts of the noise image yet at the cost of increased blurring.

As detailed in this section, tensor-product linear interpolation acts as a low-pass filter that removes high frequencies in the quantities and coordinates and thus decreases the accuracy of the results. More accurate reconstruction schemes may be employed to reduce this problem. For example, efficient higher-order B-spline techniques [271, 272] could be used, like the fast third-order interpolation proposed by Sigg and Hadwiger [252], or related pre-filtering methods [79] could be applied.



(a) (b)

Figure 4.33 — Two contrast-enhanced LIC images. Image (a) was computed with the standard method. (b) The hierarchical method exhibits less aliasing due to blurring caused by repeated resampling via interpolation.

4.5.2 Algorithmic Aspects

Some aspects of the proposed method require a thorough treatment when implementing the respective algorithms. This includes the application to time-dependent data, interpolation issues, and the handling of domain boundaries. A detailed discussion of these aspects is provided in the original paper [6] and is summarized in the following.

In contrast to the direct integration method, the hierarchical method requires the parallel computation of solutions densely covering the domain because solutions at different positions are required to construct longer solutions (see Section 4.5.1.1). This holds also for the temporal dimension in the case of time-dependent data, i.e., solutions for different starting times are computed in parallel. They are then concatenated to obtain solutions covering a longer time span. Therefore, the hierarchical method is only beneficial if solutions for different starting times are required, e.g., to create an animated visualization covering a larger time span. In this context, it has to be noted that coordinate maps for points with identical time t (see Equation (4.11), page 111) are stored together. It is therefore not required to explicitly store t for every point, which reduces the memory requirements of the method. The memory requirements can be further reduced with a streaming approach. It can be shown that computations for time-dependent data can be performed partially in a sequential way with respect to the temporal dimension. Only solutions lying in the current integration range have to be computed in parallel. Therefore, a streaming approach can be employed which keeps only the data covering this time span in memory. The complete time span of the data is not required in memory.

The error introduced by the hierarchical method is caused by the interpolation of the coordinate map; hence, the accuracy of the method depends directly on the accuracy of interpolation. Using relative positions (between end and starting positions) instead of absolute positions in the coordinate map improves the accuracy because then the number representation is not wasted for storing the absolute part of the position, and hence cancellation issues [118] are reduced. Furthermore, the hardware interpolation provided by GPUs is often of reduced accuracy. Implementing the interpolation in software can increase accuracy in these cases.

Different spatial domain types can be distinguished with respect to domain boundaries. In this work, only results for closed and periodic domains are presented. In periodic domains, trajectories outside the domain behave identically to trajectories inside the domain at corresponding locations. This case can be handled by the hierarchical scheme through the usage of relative positions in the coordinate map and their repetition outside the domain. The relative positions

include relative periodicity and therefore preserve the correct periodicity of the positions when they are followed. When implementing the method on GPUs, this can be easily handled by using texture wrapping for the coordinate maps. The 3D time-dependent ABC dataset in Section 4.5.4 is processed with this method. In closed domains, trajectories always stay inside the domain as in the Hotroom dataset used here. This case does not require any dedicated treatment by the hierarchical scheme.

4.5.3 Application

There are several methods requiring dense evaluation of integral curves in vector fields that fulfill the requirements described in Section 4.5.1.2, and thus can benefit from the acceleration scheme. Two common methods, the computation of the FTLE and LIC, will be discussed in the following. Another example is the method by Van Wijk [278] that implicitly constructs stream surfaces: particles are traced backward for every point in the domain and hence the method lends itself to acceleration by the scheme. Similar to this, Westermann et al. [300] describe a method for extracting time surfaces in vector fields for visualizing steady flow. Additionally to the backward tracing, their method determines the distances traveled by the particles, which is a quantity suitable for acceleration by the hierarchical scheme. The Mz-criterion [130] for objectively detecting vortices in incompressible flow is another candidate for acceleration.

4.5.3.1 Finite-Time Lyapunov Exponent

The computation of FTLE fields (see Section 4.1.2) usually requires sampling grids of high resolution because the topologically relevant features are immanent as ridges, which are often thin and massively folded. Especially in the case of 3D domains, the integration of a huge number of trajectories is necessary. Furthermore, the trajectories are often required to be comparably long to capture the spatio-temporal structure of the vector field. These requirements make FTLE computation very suitable for acceleration by hierarchical integration.

As the scheme introduces interpolation error (see Section 4.5.1.4), accuracy can become an issue when the FTLE is used for predictability analysis. However, the usage of the FTLE in the context of a structural analysis of vector fields is here addressed, where the goal is to extract prominent ridges in the FTLE, namely the identification of coherent regions, and therefore small perturbations or offsets can typically be neglected. The reader is referred to the error analysis in Section 4.5.1.4 and also to the work by Garth et al. [112] for a discussion of accuracy-related issues. In Section 4.5.4, the proposed method is applied to the computation of the FTLE and results as well as error analysis are provided.

4.5.3.2 Line Integral Convolution

LIC provides a dense vector field representation by convolving a noise texture along streamlines of a vector field. Usually, box or Gaussian filter kernels (for smoother results) are used. To generate the resulting image, the convolution has to be carried out for every pixel, leading to a dense evaluation of streamlines and therefore qualifying it for acceleration by hierarchical computation. However, as described in Section 4.5.1.2, the convolution has to be split up into multiple convolutions applied subsequently with the scheme. This is possible if Gaussian filter kernels are used because subsequent convolution with Gaussian filters with widths σ_1 and σ_2 corresponds to a single convolution with a Gaussian filter with width $\sigma = (\sigma_1^2 + \sigma_2^2)^{1/2}$. Hence, the application of the hierarchical scheme results in multiple convolution with increasing width of the filter kernel.

In LIC, the convolution of the input noise poses sampling issues. To avoid aliasing artifacts, the noise has to be sampled at an appropriate sampling rate. The sampling rate for the zeroth level of hierarchy depends directly on the input noise. The sampling rate is lowered in subsequent levels by the concatenation during integration with the hierarchical scheme, which requires each convolution to accordingly suppress high frequencies to avoid aliasing. This requirement is met by the Gaussian kernel. By considering the Fourier transform of the Gaussian filter, $\sigma\sqrt{2\pi}e^{-2\pi^2 f^2 \sigma^2}$, with frequency f, an appropriate sampling can be determined.

Using the Nyquist sampling theorem, a sampling distance of σ leads to a maximum sampled frequency of $f = 1/(2\sigma)$, assuming a band-limited signal. For the non-ideal low-pass characteristics of Gaussian filtering, the sampling distance σ allows capturing more than 99.99% of the energy in the signal. This is sufficient to avoid sampling artifacts in typical applications. The end points stored in the coordinate map of the previous level of the hierarchy prescribe the minimum sampling distance along a trajectory in the current level. Since it was determined that σ is an appropriate sampling distance for convolution, the size of the filter kernel is adapted such that σ of the current level matches this minimum distance from the coordinate map. To get a good approximation of the convolution integral, several samples with this distance along the trajectory are necessary. Throughout the evaluation of this work, $s = 2$ was used for the integration of the end points. However, the convolution required integration until 3σ, i.e., using $s = 6$, to obtain good results.

4.5.4 Results

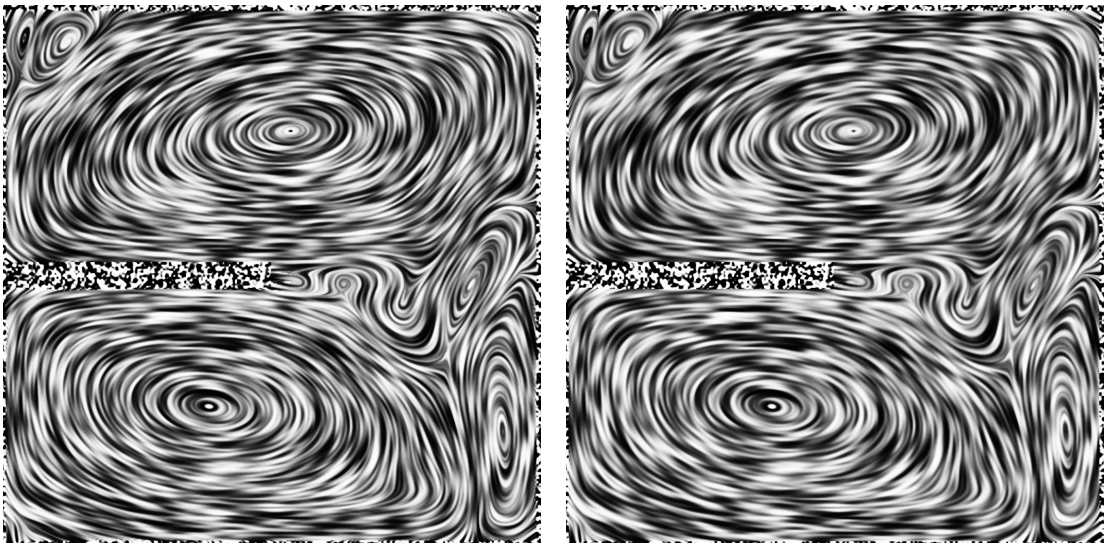
The implementation uses CUDA to perform the computations on the GPU. The standard implementation and the zeroth level of the hierarchical implementation employ fourth-order Runge-Kutta integration with fixed step size. OpenGL was used for graphical output. The hardware and operating system used is described in Section 1.4.

The hierarchical computation of LIC is demonstrated with the Hotroom dataset resulting from a CFD simulation with a heated boundary at the bottom and a cooled boundary at the top; a detailed description can be found in Section 4.3.6.1. Results for FTLE computations are obtained with the 3D time-dependent variant of the Arnold-Beltrami-Childress (ABC) [92] analytic vector field from the field of dynamical systems theory. The ABC vector field follows the common parametrization of $A = \sqrt{3}$, $B = \sqrt{2}$, and $C = 1$.

Computation times for LIC are shown in Table 4.1. Due to the uniform step sizes, computation times only dependent on the resolution of the vector field but not the contained data. Therefore, the performance numbers are valid for any dataset. A grid resolution of 512^2 was used as a medium-sized example. Computation times are mostly linear in the number of computed texels; thus, timings for other resolutions can be inferred from the measurements. The presented number of steps applies to the standard method; the hierarchical method achieves same integration ranges with fewer steps (Section 4.5.1.3). For both methods, forward and backward advection has to be performed along streamlines, doubling the number of computation steps (factor $2\times$ in the column "steps" of the tables). The ratio of direct versus hierarchical method is included as a measure of speed-up. To solely compare the computation times, computation without setup time was also measured. The performance of LIC is

Table 4.1 — Computation times (in milliseconds) of the standard (direct) and hierarchical method for LIC.

	with setup time			without setup time		
steps	direct	hier.	ratio	direct	hier.	ratio
2×13	42.48	39.68	1.07	8.08	4.88	1.66
2×27	51.96	40.64	1.28	17.56	5.84	3.01
2×55	70.72	41.88	1.69	36.32	6.88	5.28
2×110	107.48	42.92	2.50	72.88	8.12	8.98
2×221	181.96	44.56	4.08	147.36	9.96	14.80
2×443	330.96	45.88	7.21	296.36	10.88	27.24
2×886	628.93	47.04	13.37	594.52	12.24	48.57



(a) (b)

Figure 4.34 — Comparison of (a) standard and (b) hierarchical LIC for a single time step of the Hotroom dataset. Histogram equalization was applied to both images.

more than doubled at 110 computation steps for each direction. When excluding the setup time, hierarchical LIC is already more than twice as fast (compared to traditional LIC) with only 27 computation steps for each direction.

In the hierarchically computed LIC image (Figure 4.34(b)), the same structure of the vector field can be identified as in the image obtained with standard LIC (Figure 4.34(a), resolution 512^2, 221 integration steps, step size 0.01); there are almost no visible differences between the two images.

Table 4.2 shows the performance numbers for 3D time-dependent FTLE computation. The FTLE field has a spatial resolution of 128^3, and a sequence of 64 time steps were computed in all cases. The setup time (around 4 seconds maximum) can be neglected considering the total computation time. The hierarchical method has approximately logarithmic complexity with increasing integration range. The direct method theoretically exhibits linear complexity with increasing integration range. However, the results even show a slightly faster growth of the computation time for this method. A conjectured explanation for this behavior is that the memory locality of nearby starting curves is degrading with longer integration ranges and memory access becomes more expensive due to increased cache misses. The computation with the hierarchical method is already more than twice as fast as with the standard method for a rather short integration range of 4 time steps. The memory consumption measured

Table 4.2 — Computation times (in seconds) of the standard (direct) and hierarchical method for FTLE computation in 3D time-dependent vector fields.

integration range	direct	hier.	ratio
2	117	63	1.86
4	230	69	3.33
8	457	78	5.86
16	923	93	9.92
32	1 926	117	16.46
64	4 292	165	26.01

for the longest integration range corresponding to 64 discrete time steps was around 2600 MB for the hierarchical computation and 490 MB for the direct method, which includes around 400 MB for the dataset. Less than 100 MB of GPU memory is required in both cases.

Figure 4.35 shows the resulting FTLE field for the ABC dataset ((a) standard, (b) hierarchical, resolution 128^3, integration range of 64 dataset time steps, step size 0.05). Spatial periodicity was exploited according to Section 4.5.2. No differences can be observed by visual inspection and the numerical comparison of the FTLE values reveals that the results are very similar, with a maximum error of 0.86% and a 95^{th} percentile error of 0.18%. Figure 4.35(c) shows the FTLE difference mapped to the ridge surface of the standard result, whereas Figure 4.35(d) shows the corresponding distance between the ridge surfaces. The aliasing artifacts are due to sharp and high-valued FTLE ridges with respect to the used spatial resolution and appear in both the standard and hierarchical computations. The ridges are highly consistent regarding the distance measure.

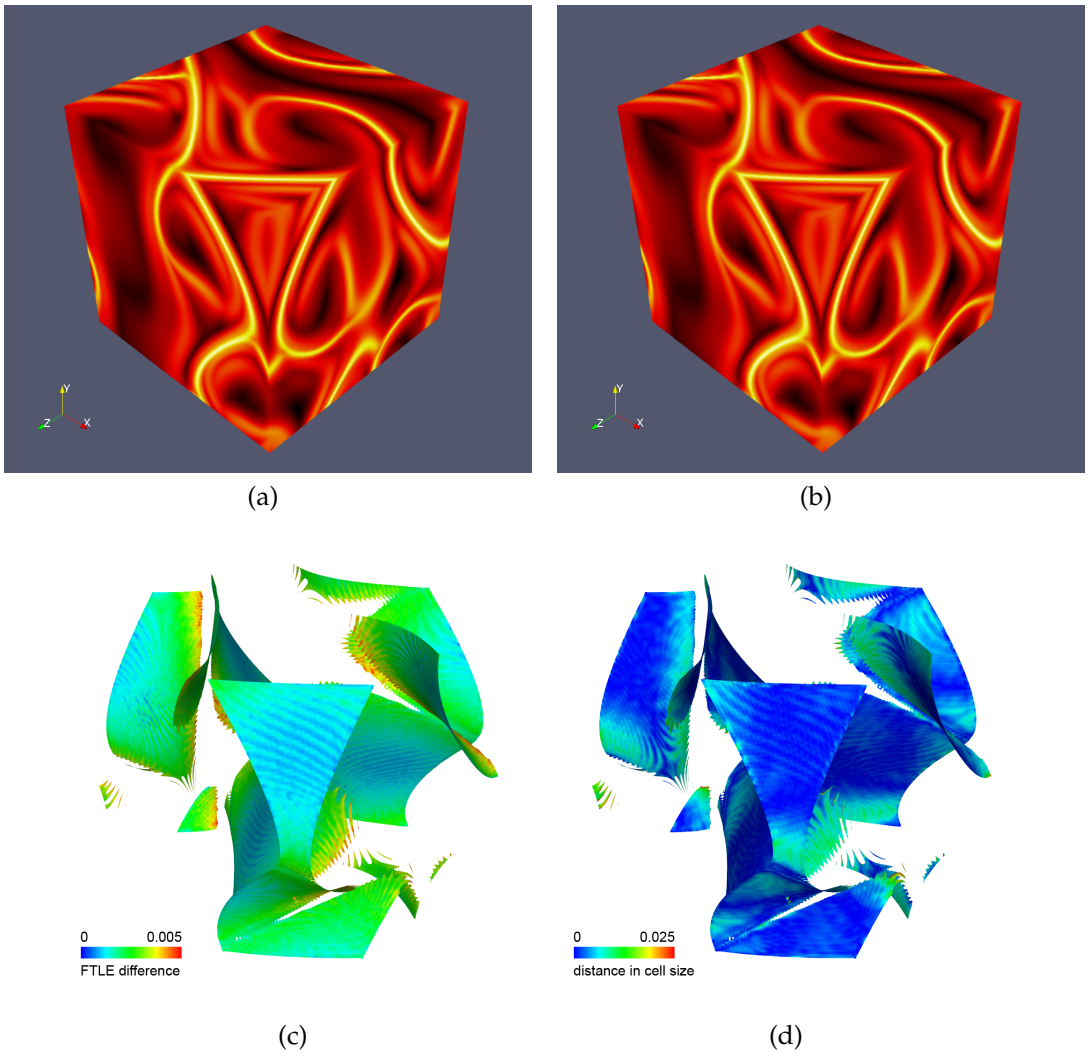


Figure 4.35 — Results of the 3D time-dependent FTLE computation. The forward FTLE for the ABC dataset is shown. The FTLE from (a) standard and (b) hierarchical computation is shown using direct volume rendering together with a black-body radiation color map. Image (c) shows the FTLE difference on the ridge surfaces of the standard version and image (d) depicts the spatial deviation of the ridge surfaces.

CHAPTER 5

Symmetric Second Order Tensors

Tensors generalize the concept of scalars and vectors. Although scalars are tensors of zeroth order and vectors are tensors of first order, the term tensor is typically used in the context of second or higher order tensors. Tensor data is employed when scalars or vectors are not sufficient to describe the respective problem or phenomenon. Examples are the velocity gradient tensor in fluid dynamics or tensors describing stress, strain, or deformation in material or life sciences. Dick et al. [86] [87], e.g., demonstrate real-time simulation and visualization of stress tensors in the context of implant planning in orthopedics.

Another source for tensor data is *diffusion tensor magnetic resonance imaging* (DT-MRI) [149], a medical imaging technique facilitating in vivo measurements of the rate and orientation of water molecule diffusion within tissue. Ehlers and Wagner [96], e.g., incorporate DT-MRI data for the simulation of drug-delivery inside the human brain. These examples and many other applications employ symmetric second order tensors. Therefore, the visualization of this class of tensors is of great interest. This is typically a non-trivial task due to the multivariate nature of these tensors.

This chapter[1] presents an extension of the FTLE and the concept of Lagrangian coherent structures (Section 4.1.2) to symmetric second order tensor fields. This approach allows the extraction of regions of coherent behavior from such tensor fields, which is demonstrated for stress and diffusion tensor data.

[1] **Parts of this chapter have been published in:**
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):781–794, 2011 [7], © 2011 IEEE.

5.1 Related Work

In the past, different methods were developed that aim at providing meaningful visualizations of multivariate tensor data. An overview of newer developments in this field can be found in the books by Laidlaw and Vilanova [170] and Westin et al. [301]. This chapter focuses on the visualization of second order tensor fields describing diffusion or mechanical stress.

A diffusion tensor can be visualized geometrically with a diffusion ellipsoid [171, 216], where the axes and radii correspond directly to the eigenvectors and eigenvalues of the tensor. Similarly, composite shapes [303] can be used, while Kindlmann [154] proposes the use of superquadrics to combine the benefits of both the ellipsoid and box glyphs.

Another common practice is to visualize properties derived from diffusion tensor fields on cross sections of the dataset. Widely used are local scalar measures such as mean diffusivity, fractional anisotropy [37], as well as the coefficients for linear anisotropy, planar anisotropy, and spherical isotropy [302]. They can be visualized, e.g., with direct volume rendering [155]. For the visualization of fiber structure, methods from classical vector field visualization have been adapted. Characteristic lines of a tensor field can be traced by following its eigenvector fields and strategies to reduce resulting visual clutter are available [282]. Hyperstreamlines [83] additionally include the other eigenvectors as thickness transversal to the chosen eigenvector. Tensorlines [298] incorporate a notion of artificial inertia to the line tracing process in order to apply some control on the direction of tracing when areas of insufficient linear anisotropy are crossed. The HyperLIC approach [314], where a noise field is filtered using primitives deformed by the tensor field, provides a dense tensor field visualization in the style of line integral convolution [63]. Further references to DT-MRI visualization techniques are given in the survey article by Vilanova et al. [283].

Tractography in diffusion tensor fields deals with the methodology of (in vivo) reconstruction of fiber tracts in organized tissue. A technical review is given by Mori et al. [193]. Concerning cerebral tissue, the greater goal of tractographic methods is to uncover the organization of the brain, e.g., in order to understand its functional architecture [177] or to construct atlases of the brain [287]. Concerning muscle tissue, the discussion on its organization is in parts still ongoing [115], and tractographic methods based on DT-MRI [317, 211, 230] can aid its uncovering. An effective way to gain such insight in the case of cerebral tissue is to group extracted fibers according to similarity and to connectivity with different functional areas. Brun et al. [54] propose a mapping of each fiber to an Euclidean feature space and interpret the pairwise distance

of fibers in this space as a similarity measure that they represent as a weighted undirected graph. Subsequent normalized cuts in this similarity graph allow an unsupervised segmentation of fiber bundles. Ding et al. [91] propose a different fiber similarity measure based on the ratio of fiber lengths and the average Euclidean distance of overlapping fiber segments. A *k-most-similar-fibers* algorithm allows them to identify bundles of fibers and a classification of them.

In visualization, topological analysis of tensor fields was first performed by Delmarcelle and Hesselink [84]. Such methods extract the essential structure and properties of the multivariate tensor field, thus leading to a significant reduction of information to be assessed. The topology of 3D tensor fields is analyzed by Zheng and Pang [315] and Zheng et al. [316]. Tricoche et al. [263] use invariants for the extraction of topological features from 3D tensor fields.

Other approaches aim at extracting essential information from tensor fields by means of general feature extraction, such as interfaces [200], creases [156, 158], edges [311], and changes in tensor shape and orientation [157].

The focus of the method presented in this chapter is the visualization of the structural organization within symmetric second order tensor fields. In this respect, it can be stated that certain issues are not solved satisfactorily by previous approaches: in classical tensor field visualization (i.e., direct visualizations of scalar measures derived from tensors, or characteristic line visualization), the structure of a tensor field can only be implicitly deduced. Methods from the area of feature extraction are mostly based on local criteria (i.e., based on some kind of local gradient measure), which only capture local and distinct variation but cannot extract further structural information from regions of coherent behavior. Approaches from tensor field topology are particularly sensitive to noise, which makes them less robust. Tractographic methods successfully extract structural information from diffusion tensor fields, but require performing fiber similarity comparisons on a large number of fiber trajectories.

The method described in the following is based on the concepts of the FTLE and LCS introduced in Section 4.1.2. Related work for extracting coherent structures in vector fields is discussed in Section 4.2.3

5.2 Generalized Lagrangian Coherent Structures

In this section, the concept of map-based extraction and visualization of Lagrangian coherent structures (Section 4.1.2) is generalized. Due to the fact that the computation of the FTLE is solely based on the flow map, the following generalization is proposed: instead of restricting the derivation of the map to a vector field or dynamical system, it should be possible to extract and visualize coherent structures using the right Cauchy-Green deformation tensor from any map, irrespective of its origin. As in Section 4.5, the term *coordinate map* is used in the following to denote this extension of the flow map. Regarding the gradient computation on the coordinate map, the only requirement is that the map is spatially C^1 continuous, except for isolated points or null sets that may be excluded from the analysis. In the case of discrete coordinate maps, it is beneficial to require a regular sampling to account for the sampling dependency of the FTLE and to prevent artifacts from gradient estimation.

The generalization also addresses mechanisms where a starting point corresponds to more than one end point. This is the case for undirected correspondences and for eigenvector fields, where two possible end points exist. This generalization is even more important when applying it to data from medical imaging methods with high angular resolution [265], [262], [106], where characteristic lines could be extracted, e.g., with the methods by Schultz and Seidel [250] or by Hlawitschka et al. [141]. In contrast to diffusion tensors, crossing fibers can be represented with multiple discriminable directions in this case, which may lead to more than two end points for a given starting point.

To simplify the computation of LCS, gradient computations of such 1-to-m correspondences are avoided. Instead, these situations are modeled by a set of m maps. The generalized map that defines the mapping of a point $\mathbf{x}$ at time t_0 to one of its m possible positions at time $t_0 + T$ shall be defined as $\boldsymbol{\xi}^T_{t_0,k}(\mathbf{x})$, with $k \in \{k_1, \ldots, k_m\}$ selecting the different maps. If m equals to 1, k is omitted for clarity. Care has to be taken if local operators are applied to $\boldsymbol{\xi}^T_{t_0,k}(\mathbf{x})$, since the choice of equal k at different positions within the support of the operator may not ensure consistent selection of corresponding mappings. The consistency of mappings must be established through an appropriate choice of k.

Since the gradient of the coordinate map is the basis for the extraction of LCS, it shall be derived here. Other local operators can be derived similarly. The adapted gradient tensor operator $\tilde{\nabla}$ of the coordinate map is formulated as the

following limit process

$$\begin{aligned}\left(\tilde{\nabla}\boldsymbol{\xi}^T_{t_0,k}(\mathbf{x})\right)_{ij} &= \frac{\tilde{\partial}\left(\boldsymbol{\xi}^T_{t_0,k}(\mathbf{x})\right)_i}{\partial x_j} \\ &= \lim_{h\to 0}\frac{\Delta_{j,h}\left(\boldsymbol{\xi}^T_{t_0,k}(\mathbf{x})\right)_i}{h}\end{aligned} \tag{5.1}$$

using index notation, with

$$\Delta_{j,h}\left(\boldsymbol{\xi}^T_{t_0,k}(\mathbf{x})\right)_i = \left(\boldsymbol{\xi}^T_{t_0,\eta}(\mathbf{x}+h\mathbf{e}_j)\right)_i - \left(\boldsymbol{\xi}^T_{t_0,k}(\mathbf{x})\right)_i \tag{5.2}$$

where $\mathbf{e}_j$ is the j-th basis vector and η ensures that the previously formulated continuity requirement of the coordinate map is satisfied by a selection of the mapped position at location $\mathbf{x}+h\mathbf{e}_j$ such that the limit in Equation (5.1) exists. An explicit choice of η for the case of coordinate maps extracted from eigenvector fields of symmetric tensors is given in Section 5.3.3.

The parameters representing time require further consideration. For datasets that consist of a single time step (stationary data), t_0 is constant and is therefore omitted in the notation. In addition, there are cases where the underlying mechanism does not represent a temporal evolution. If the map is the product of a tracing procedure, T can be interpreted as "tracing time". Otherwise, T is also omitted in the notation. The normalization by tracing time T in the definition of the FTLE (Equation (4.9), Section 4.1.2, page 73), was originally motivated by the growth rate of a perturbation. If T is uniform over the domain, normalization only results in a scaling and hence does not change resulting features (ridges). However, if trajectories are stopped earlier, e.g., because they leave the domain, this normalization can change features. Furthermore, many maps may not reflect a temporal process. Therefore, it is proposed to generally leave this step out and to decide on its application depending on the respective use case.

The logarithm was introduced in Equation (4.9) because perturbations grow exponentially in linear vector fields or dynamical systems and the resulting measure captures the underlying linear behavior. However, since the proposed generalization allows for the analysis of arbitrary maps, a logarithmic measure should generally not be enforced. Furthermore, since the logarithm is a monotonic function, it cannot change existing features (i.e., ridges), it only changes the level at which they are expressed. Therefore, this step is left optionally and should only be applied if the goal of the analysis is the measurement of exponential growth. If the resulting data is visualized directly, it can still be

beneficial to compute the logarithm in terms of a transfer function for better exploitation of the available color map range.

Based on the previous definitions, the right Cauchy-Green deformation tensor of the generic coordinate map is

$$\mathbf{C}^T_{t_0,k}(\mathbf{x}) = (\tilde{\nabla}\boldsymbol{\xi}^T_{t_0,k}(\mathbf{x}))^\top \tilde{\nabla}\boldsymbol{\xi}^T_{t_0,k}(\mathbf{x}). \tag{5.3}$$

From this the generalized counterpart to the FTLE is defined as

$$\tilde{\sigma}^T_{t_0}(\mathbf{x}) = \max_{k\in\{k_1,\ldots,k_m\}} \left(\sqrt{\lambda_{\max}(\mathbf{C}^T_{t_0,k}(\mathbf{x}))} \right), \tag{5.4}$$

called the maximum *finite separation ratio* (FSR), for which the following equality holds: $\sigma^T_{t_0} = 1/|T| \cdot \ln \tilde{\sigma}^T_{t_0}$, for the special case $m = 1$. The formulation of Equation (5.4) contains a selection of the maximum separation over all k, in analogy to the previous definition of the Lyapunov exponent as a measure of maximum separation. However, there also may exist application-specific cases where a different measure such as a minimum or average FSR may be desired. The equation can be accordingly adapted in this case. Furthermore, logarithm is omitted, which has benefits in many applications where exponential growth has no significance. It is a direct measure of the maximum change of an initial distance under the considered mechanism. For the example of cerebral anatomy, this allows for a direct measurement of the maximum distance of two axonal fibers after tracing from proximate starting points in relation to their initial distance.

Also conceivable, but not applied here, is an FSR maximum in analogy to the FTLE maximum according to Sadlo and Peikert [234], which measures the maximum separation along an entire trajectory instead of measuring it from the final flow map for a fixed advection time T. This is equivalent to the maximum FSR over a range of advection times T, and can easily be modeled with the FSR by including the maps for different advection times in the set of k from which the definition of Equation (5.4) already selects the maximum.

The generalized counterpart to LCS are the ridges in the FSR, which are proposed to be called *separatrices of coherent regions* (SCR). In this new nomenclature, the term "Lagrangian" is discarded because the considered maps do not necessarily describe transport phenomena. Furthermore, the term "separatrix" is introduced in order to disambiguate the term LCS with respect to coherent regions and the structures that separate them from each other.

One reason why LCS became widely accepted in flow visualization is that they give an appropriate view to transient vector fields: as opposed to vector field topology, they allow an intuitive interpretation with respect to transient

advection and come in the form of time series (of varying t_0) that enable this interpretation at each instant of time. One question is to what extent these properties of LCS transfer to the concept of SCR. If the time steps of the data represent a process that is temporally consistent with the progression of the seeding moment t_0 and advection time T, the coordinate map can be extracted in a time-dependent way, e.g., by tracing pathlines in the case of vector fields. In this case, SCR analyze transient advection and hence lead to a transient view of the problem in question, just as LCS. However, if the mechanism represented by the coordinate maps is instantaneous, if T is not consistent with t_0 of different coordinate maps, or if the scope of the analysis is purely structural, the coordinate map should be extracted in an instantaneous manner, e.g., by tracing streamlines in the case of vector fields. The SCR for each t_0 have then to be primarily examined on their own.

5.3 Coherent Structures in Symmetric Tensor Fields

The FSR is now applied to symmetric second order tensor fields, in particular to diffusion tensor fields from DT-MRI and to stress tensor fields. Beside the definition of integral curves, which are required to build the coordinate map, the orientation ambiguity of eigenvectors in tensor fields has to be considered.

5.3.1 Diffusion Tensor Fields

One of the most prominent sources of symmetric second order tensor fields are medical diffusion tensor images. Since these describe the dominant direction of diffusion of water molecules in tissue, it is reasonable to base a structural analysis of such data on sets of characteristic lines, in analogy to stream- and pathlines in vector fields. In medical applications, such as tractography, it is common to use integral curves of the tensor field's major eigenvector. They are denoted as eigenvector lines in the following. Because the major eigenvector is only significant in anisotropic regions, these lines are commonly stopped when leaving such areas. In the presented examples for diffusion tensor fields (Section 5.4), these lines are used for the computation of the FSR and fractional anisotropy [37] is used as stopping criterion. However, the concept of the FSR is independent of these choices; other definitions for eigenvector-based lines are possible just as well, e.g., tensorlines introduced by Weinstein et al. [298].

In medical data, the geometric structure of tissue is likely to be independent of the actual rate of diffusivity, which is why the major eigenvalue is not incorporated in the propagation of an eigenvector line. For cases where the absolute rate of diffusivity contributes to the understanding of a diffusion tensor field, and for applications from general physics, a step size control is optionally included that scales the obtained direction vector with a factor $\delta = \lambda_1 / \lambda_{\max}$, where λ_1 is the local major eigenvalue and $\lambda_{\max}$ is the maximum major eigenvalue throughout the entire dataset. Regarding neighboring eigenvector lines, this variation can create shearing (differing propagation velocities along eigenvector lines), provoking separation of their end points, which becomes visible in the FSR. Section 5.4.1 includes an example demonstrating the utility of this variation.

5.3.2 Stress Tensor Fields

Tensors describing mechanical stress [187] are another class of symmetric second order tensors. In contrast to diffusion tensors, stress tensors can exhibit negative eigenvalues. Their eigenvalues correspond to the principal stress magnitudes and their eigenvectors to the principal stress directions. The sign of the

eigenvalue classifies the stress as tension (positive eigenvalue) or compression (negative eigenvalue). In contrast to diffusion tensors, a classification into different types of anisotropy (linear, planar, spherical) is often not useful and there is typically no case of linear anisotropy, where only a single direction along the major eigenvector would be sufficient. The analysis of stress tensor fields can be based on lines following the principal stress directions as described by Dick et al. [87]. In Section 5.5, coordinate maps are extracted from these lines and used for the calculation of the FSR for all principal stress directions, resulting in three different FSR fields.

5.3.3 Orientation Ambiguity in Tensor Fields

Because of the orientation ambiguity of eigenvectors, there are for each starting position two end points of eigenvector-based lines. Therefore, the resulting mapping can be represented by two coordinate maps $\boldsymbol{\xi}^T_{t_0,k}(\mathbf{x})$ ($k \in \{-1,+1\}$) (Section 5.2), where k selects the final position according to the initial propagation direction $\mathbf{v}_0(\mathbf{x}) = k\boldsymbol{\epsilon}(\mathbf{x})$ depending on the eigenvector $\boldsymbol{\epsilon}(\mathbf{x})$. Directly applying Haller's approach of FTLE computation [128] to these coordinate maps would be inappropriate, since misaligned (opposing) starting directions can lead to overestimated separation of trajectories as well as discontinuities. Therefore, local orientation (formally motivated in Figure 5.1) is decided using

$$\eta = \mathrm{sgn}_1(\mathbf{v}_0(\mathbf{x}) \cdot \boldsymbol{\epsilon}(\mathbf{x} + h\mathbf{e}_j)),$$

with $\mathrm{sgn}_1(x \neq 0) = \mathrm{sgn}(x)$ and $\mathrm{sgn}_1(0) = 1$. This definition of η makes sure that trajectories consistent with $\mathbf{v}_0(\mathbf{x})$ are chosen for the evaluation of $\tilde{\nabla}\boldsymbol{\xi}^T_{t_0,k}(\mathbf{x})$ (Equations (5.1)–(5.3)). Similarly, local alignment of eigenvectors is also performed during line tracing. Following Equation (5.4), the FSR for symmetric tensor fields is defined as

$$\tilde{\sigma}^T = \max\left(\sqrt{\lambda_{\max}(\mathbf{C}^T_{t_0,+1})}, \sqrt{\lambda_{\max}(\mathbf{C}^T_{t_0,-1})}\right).$$

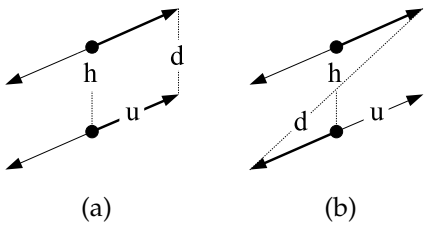


Figure 5.1 — Accounting for the ambiguity of eigenvector orientation for differential operators. In (a) and its symmetric counterpart, $\lim_{h\to 0} d/h = 0$ exists, whereas in (b) this is not the case, requiring local orientation of the eigenvectors.

5.3.4 FSR Visualization

As noted by Haller [128], LCS correspond to ridges in the FTLE field. The same relation holds for SCR in the FSR field and such ridges can be obtained with appropriate ridge extraction algorithms [95], [110]. However, due to the noise sensitivity of these approaches, the geometry of ridges in 3D FSR scalar fields is preferably not extracted but direct cross-sectional visualizations are performed with a linear gray scale color map, where low FSR values are mapped to black and high values mapped to white (see, e.g., Figure 5.6). Positions, for which no FSR was computed, because the stopping criterion (Section 5.3.1) was already met at the starting position, are marked turquoise. Additionally, if FSR values cover a high dynamic range, one can resort to gray scale color mapping of FSR values on a logarithmic scale. For the results of the stress tensor field, high values are mapped to red or green, to allow a combined visualization of two FSR fields (see Figure 5.9). As previously shown by Garth et al. [113], a direct LCS visualization without explicit ridge extraction is often sufficient to faithfully identify coherent regions and their separating structures.

5.3.5 Implementation

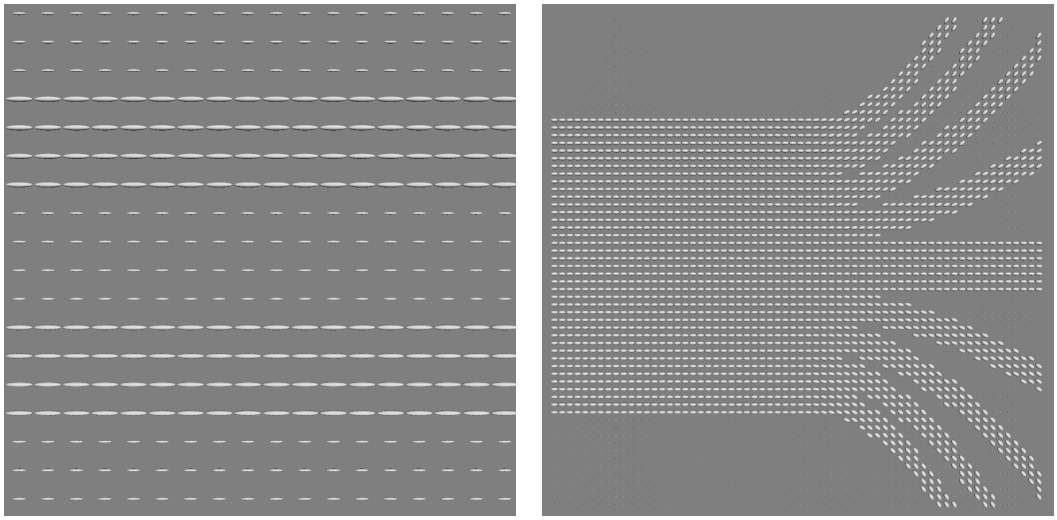
In practice, LCS in vector fields are visualized by defining a grid of seeding positions for the computation of the flow map. Since a full 3D flow map may easily require the costly computations of hundred thousands or even millions of stream- or pathlines, several acceleration methods have been proposed (see Section 4.2.3 and Section 4.5). With tensor fields, the issue of computational complexity is even aggravated. First, the coordinate map for diffusion tensor fields requires the computation of two integral curves at each of its nodes (one parallel to the local major eigenvector and another antiparallel to it). In the case of stress tensor fields, the number of computed curves per node even increases to six if the FSR is computed for all three principal stresses. Second, for each of its integration steps, the propagation of these lines requires the local solution of the eigensystem of the tensor field. To make the analysis of tensor fields feasible at all, the computation of the FSR was accelerated using the GPU inspired by the approach of Kondratieva et al. [161]. The method is implemented in C++ using OpenGL and its shading language. With this implementation, the computation of the FSR field in Figure 5.5(a) (1024×1024 resolution) requires around one second. More details of the implementation can be found in the original paper [7].

5.4 Results for Diffusion Tensor Fields

In the following, the properties of the proposed method for the visualization of SCR in diffusion tensor fields are demonstrated. The FSR is visualized and compared to other visualization techniques for synthetic and measured diffusion tensor fields. The datasets are:

1. Alternating diffusivity: a synthetic $64 \times 64 \times 64$ tensor field consisting of unidirectional linear anisotropy with total diffusivity alternating by a factor of two every four slices. All tensors are thus linearly anisotropic and differ only in scale (Figures 5.2(a) and 5.3).
2. Synthetic fiber bundle: a synthetic $64 \times 64 \times 64$ tensor field that models a simplified bundle of fibers that separates at one side (Figures 5.2(b) and 5.4).
3. Canine heart: a measured $256 \times 256 \times 134$ DT-MRI image of an excised canine heart (Figure 5.5).

All FSR fields were computed in fourfold native dataset resolution, normalized with the 95^{th} percentile, and visualized with a linear gray scale color map. Further results for a diffusion tensor field of a human brain can be found in the original paper [7].



(a) (b)

Figure 5.2 — Synthetic tensor datasets. Tensor glyph (diffusion ellipsoid) visualization of (a) the alternating diffusivities dataset and (b) the synthetic fiber bundle dataset.

5.4.1 Comparison to Other DT-MRI Visualization Methods

Due to its dependence on the major eigenvector and thus on linear anisotropy, the approach is compared to established methods for analysis of linear anisotropy in diffusion tensor fields. In detail, these methods are:

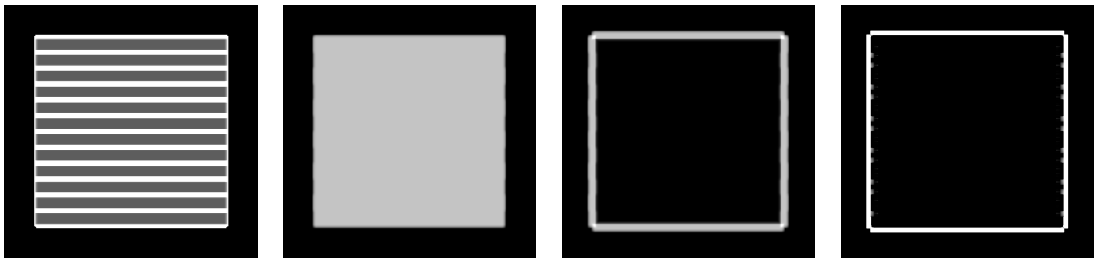
1. Fractional anisotropy (FA) as introduced by Basser and Pierpaoli [37].
2. The gradient magnitude of FA, $\|\nabla \mathrm{FA}\|$.
3. A ridge criterion based proposed by Kindlmann et al. [156], $|\nabla \mathrm{FA} \cdot \epsilon_3|^{1/3}$, where ϵ_3 is the minor eigenvector of the Hessian of FA.

All methods used the same resolution as the FSR and gradients were computed with central differences. Drawing analogies from previous results in the visualization of LCS in vector fields, the FSR can be expected to be capable of highlighting or discriminating the following features in tensor fields:

- Regions with locally deviating major eigenvector directions.
- Regions of locally varying diffusivity (if step size control based on the local eigenvalue is used).
- Structuring of locally coherent regions according to semi-global divergence of trajectories, i.e., separation in the course of a fiber bundle.

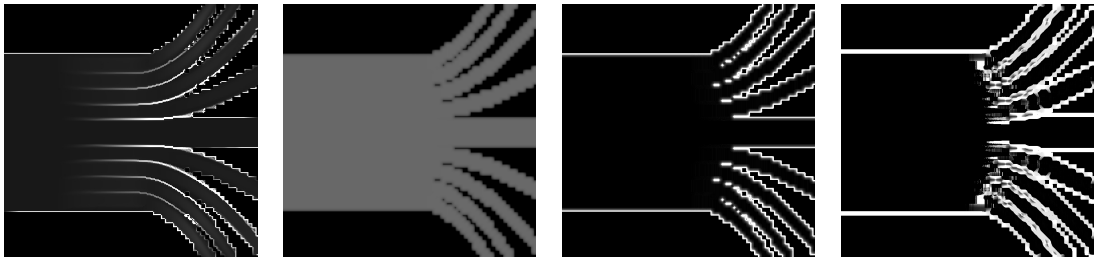
Section 5.3.1 introduced a variation of the eigenvector line tracing model that adds sensitivity to tensor diffusivity by performing propagation steps proportional to the major eigenvalue for the extraction of eigenvector lines. Figure 5.3 demonstrates results obtained by applying this variation to the alternating diffusivities dataset. It is apparent that this variation makes regions of changing diffusivity visible in the FSR field. Since the change in total diffusivity does not influence the respective ratios of eigenvalues, this feature cannot be detected with any visualization method solely derived from fractional anisotropy.

Figure 5.4 presents results obtained with the synthetic fiber bundle dataset. All methods delimit the fiber bundle against the background. Separating line trajectories form ridge features in the FSR, as expected. Additionally, features visible in the FSR appear to not only coincide with structures visible in the other methods, but to form a superset of features that grows with increasing integration length. The possibility to perform such a semi-global analysis is one of the major advantages of the FSR, since it allows one to structure coherent regions in accordance to distant separation in the progression of trajectories (see also Section 5.4.2).



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

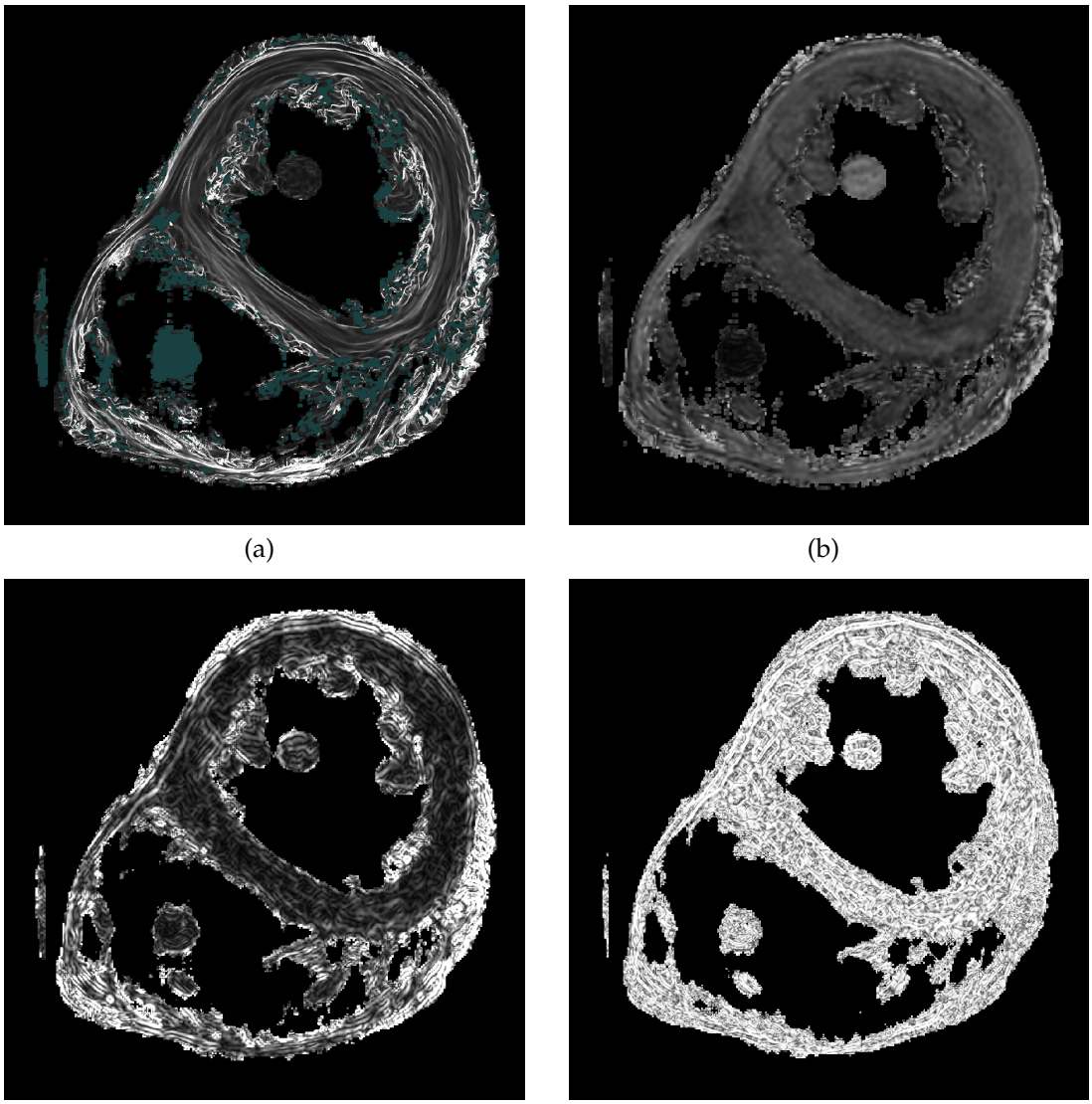
Figure 5.3 — A comparison of (a) the FSR, (b) FA, (c) the gradient magnitude of FA, and (d) the FA ridge criterion applied to the synthetic alternating diffusivity dataset. The FSR computed with eigenvalue-based step size control discriminates regions with differing total diffusivity, which cannot be extracted with the other methods.



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

Figure 5.4 — A comparison of (a) the FSR, (b) FA, (c) the gradient magnitude of FA, and (d) the FA ridge criterion applied to the synthetic fiber bundle dataset. The features visible in the FSR field form a superset of those visible in the gradient magnitude of FA and the FA ridge criterion.

Figures 5.5 and 5.6 show the canine heart dataset. The FSR distinguishes coherent regions of uniform fiber orientation (separated by ridges of high gray scale value). The myocardium of the mammalian heart consists of several layers of helical muscle fiber tracts. The angles of these fibers are changing from the epicardium to the endocardium (e.g., described and visualized in [255]). This layer structure of the myocardium is apparent in the FSR field of the heart dataset; thin ridges appear where the angle of neighboring fiber layers is changing. The different angles of the layers are also visible in the eigenvector lines shown in Figure 5.6. To some extent, these structures are likewise visible with the FA-based approaches (Figure 5.5), but altogether, the FSR field tends to represent these structures in a more coherent and robust manner.



(a) (b)

(c) (d)

Figure 5.5 — A comparison of (a) the FSR, (b) FA, (c) the gradient magnitude of FA, and (d) the FA ridge criterion applied to the measured canine heart dataset. Although computed at the same resolution as the other approaches, the FSR reveals extensive coherent structures that are far less sensitive to noise.

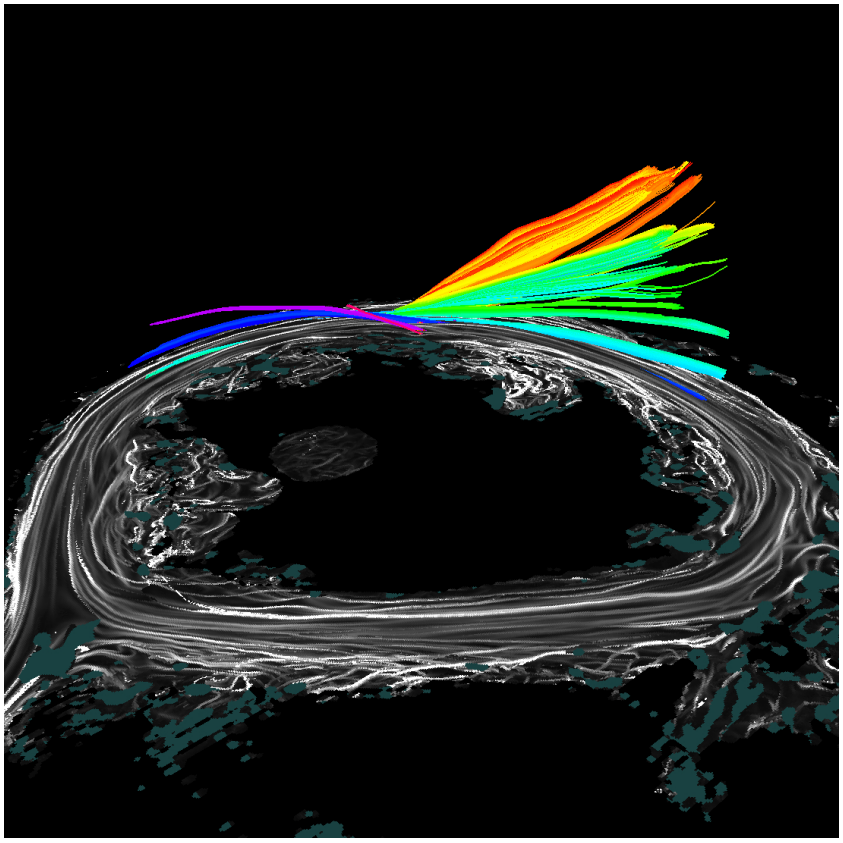
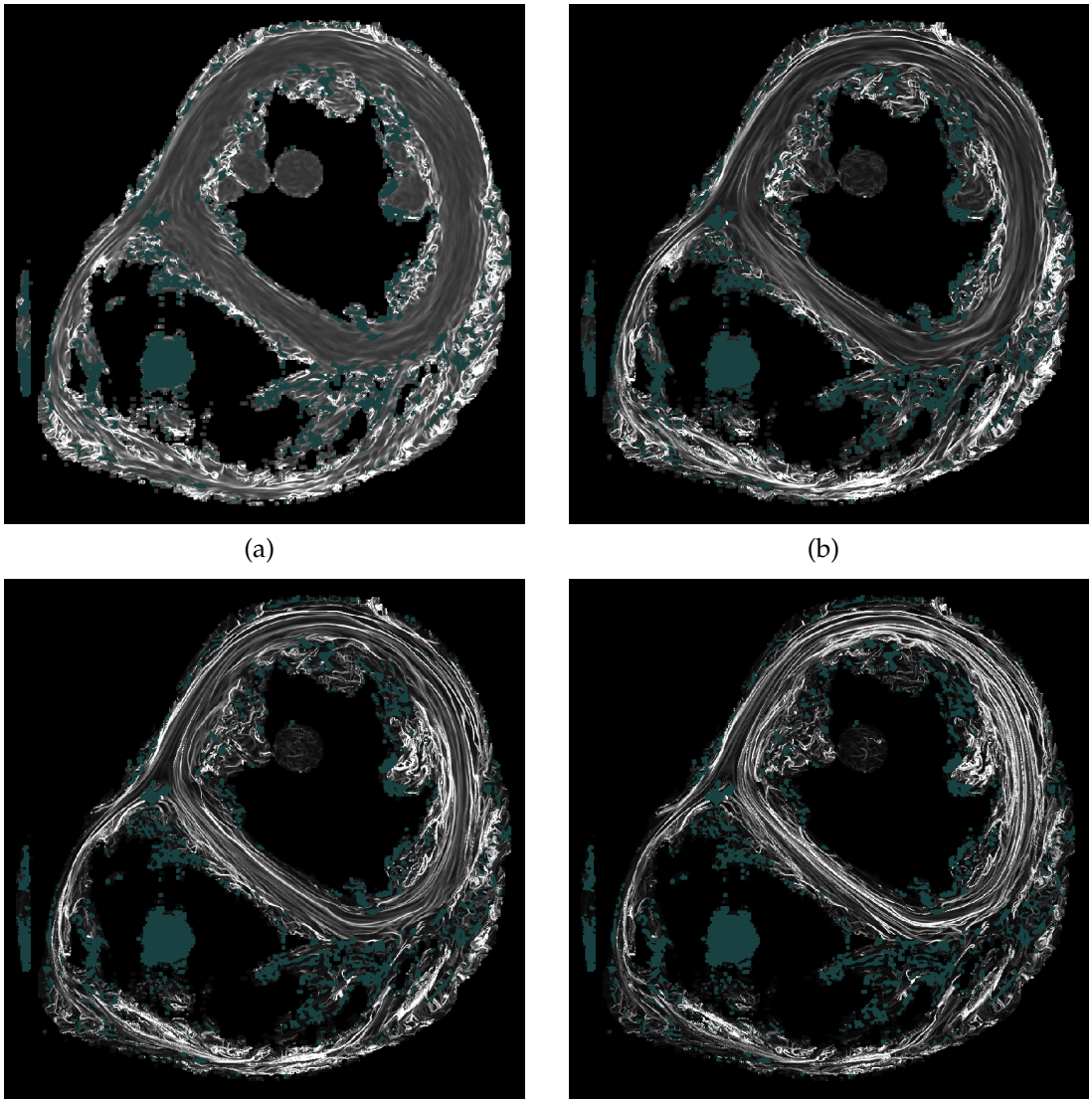


Figure 5.6 — FSR-guided seeding of eigenvector lines in a cross-section of the canine heart dataset. Lines are seeded in regions with low FSR values within a rectangular region of interest. The angle between the XY-plane and the eigenvector at the respective seeding position is mapped to hue in the HSV color model, a common approach in this context. It shows that seeding at positions of low FSR in combination with this color mapping allows the identification of coherent bundles of fibers.

5.4.2 Varying Integration Length

A major benefit of the FSR is that it allows for a gradual transition from local to global analysis by increasing the length of the integral curves. In doing so, propagation length effectively represents a scale space parameter that determines the scale at which separation of trajectories is analyzed. The effect is illustrated in Figure 5.7 with the canine heart dataset, where structures in the FSR grow and noise is suppressed with increasing line length. This effect of growing structures in the FSR also becomes apparent when comparing the respective first images of Figures 5.4 and 5.8, which were obtained with different propagation lengths.



(a) (b)

(c) (d)

Figure 5.7 — The effect of increasing propagation length (measured in multiples of the voxel edge length—(a): 5 voxels, (b): 25 voxels, (c): 75 voxels, (d): 150 voxels) on the FSR demonstrated for the heart dataset. With increasing propagation length, structures become more coherent and noise artifacts are averaged.

5.4.3 Susceptibility to Noise

As indicated in Figure 5.7, a semi-global analysis by means of increasing the propagation length for the FSR helps reduce the impact of noise, as discussed by Haller in the context of vector fields [129]. In Figure 5.8, the propagation length is fixed and the effect of different levels of noise to the FSR field is demonstrated with the synthetic fiber bundle dataset. The FSR is also compared to the gradient magnitude of FA. As a semi-global measure, one would expect the FSR to degrade strongly due to accumulation of error along trajectories. However, while the quality of both FSR and $||\nabla \mathrm{FA}||$ does degrade, the overall structure within the FSR remains comparatively intact. Obviously, these results can only give an initial qualitative impression. A detailed quantitative analysis is subject of future work.

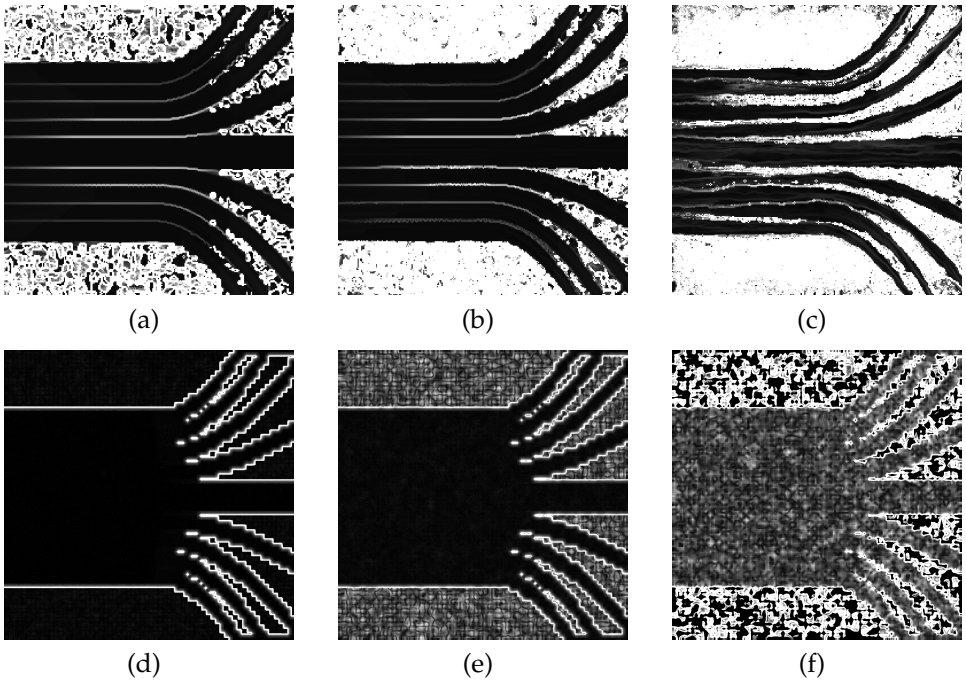


Figure 5.8 — Comparing the FSR (upper row) and the gradient magnitude of FA (lower row) under the effect of noise. White noise with uniform distribution has been added to all entries of the tensors. The maximum amplitude of the noise (relative to the maximum tensor component throughout the entire dataset) is increased from left to right ((a) and (d): 0.2%, (b) and (e): 2%, (c) and (f): 20%). The quality of structures visible with both approaches degrades with an increasing level of noise. However, even though the FSR is a semi-global measure that might potentially accumulate errors, the overall structure persists even at high noise levels.

5.5 Results for Stress Tensor Fields

The FSR was applied to medical stress tensor fields by using integral curves following the principal stress directions (see Section 5.3.2). The presented results base on a stress tensor field at a resolution of $86 \times 81 \times 226$ that resulted from a simulation of a human femur under load; the tensor field is only defined inside the bone. Figure 5.9 shows results for this dataset. The FSR field for lines that follow the principal stress with highest positive eigenvalue, which corresponds to tension, is shown in green. Red color is applied to the FSR field for lines that follow the principal stress with highest negative eigenvalue, which corresponds to compression. While both FSR fields exhibit a lot of noise in the center area of the bone, clearer structures are visible at the bottom and top part of the bone. Figures 5.9(b)–(d) show close-ups of the top part of the bone (marked in Figure 5.9(a)), where distinct ridge lines are visible in the FSR fields. These ridges separate regions with coherent directions of the principal stresses.

The close-up (Figure 5.9(b)) shows that ridges in the tension and compression FSR fields typically do not occur at the same position and have different orientations. The tension FSR field (green) exhibits a distinct ridge at the left side of the bone (marked as (1)), whereas the compression FSR field (red) has a distinct ridge at the right side (marked as (2)). A direct visualization of principal stress directions can be obtained by computing lines following the principal stress directions (see Section 5.3.2). The lines were computed in both possible eigenvector directions for the tension (Figure 5.9(c)) and compression FSR field (Figure (d)). They can still visually cross the ridges in the FSR field or leave the dataset domain because a 2D slice of the FSR field was combined with 3D lines in this visualization. The seeds (marked blue) for the line visualizations were placed along lines that cross the prominent ridges in the marked areas to illustrate the relationship between these ridges and the directions of principal stress. The images confirm that ridges in the FSR field show the separation of the lines following the principal stress direction. A similar dataset is used in the work of Dick et al. [87] and their results confirm the resulting FSR structures. The typical distribution of principal stress directions in a human femur can be recognized in the FSR field. For example, a ridge in the tension FSR field can be found where the stress lines separate on the upper left side. These results show that the FSR can help interpret stress tensor fields. Depending on the length of the underlying lines, the FSR can be seen as a semi-global measure for coherent principal stress regions, with low FSR values indicating areas of coherent stress directions and high FSR values separating different regions of coherent stress.

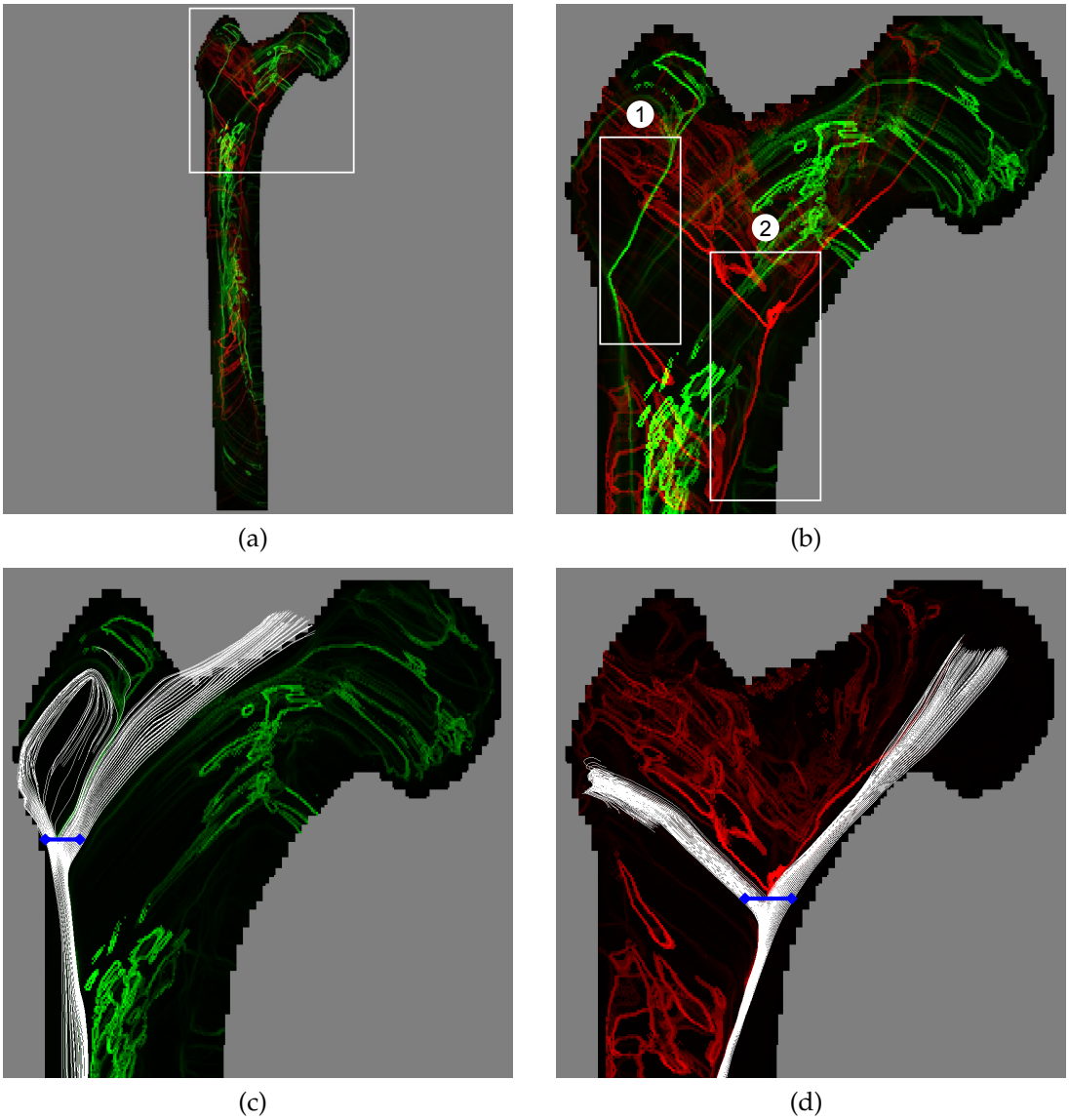


Figure 5.9 — Visualization of the FSR fields for the stress tensors of the femur dataset. (a) Single slice of the dataset with a combined visualization of the tension (green) and compression (red) FSR fields. The marked area is shown in the other images. (b) Ridges in the FSR field indicate separating lines of principal stress direction: (1) distinct ridge in the tension FSR filed, (2) distinct ridge in the compression FSR field. (c) Tension FSR field combined with 3D line visualization of the principal stress direction. (d) Compression FSR field combined with 3D line visualization of the principal stress direction. In both cases, the seeds for the line visualization were placed along the line between the blue markers.

Part III

Uncertainty

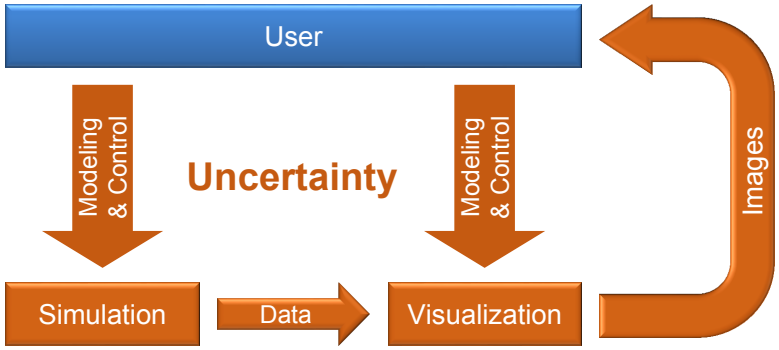


Figure III — Simulation and analysis process. Uncertainty can occur in all stages of the process (shown in orange).

In the context of simulations, uncertainty can occur in all phases of the simulation and analysis process (Figure III). In many cases, simulation models exhibit simplifications of real processes, or input parameters cannot be determined accurately. Furthermore, the computations required for the simulation can introduce numerical inaccuracies. These examples consider only the simulation phase itself but the additional steps required for analyzing the resulting data can also introduce uncertainties. For instance, visualizing the results may require interpolating the data or the resulting images are of reduced resolution to allow interactive frame rates for exploration.

In all these cases, uncertainties have to be considered. While not all types of uncertainties require a treatment, e.g., the computations are accurate enough for certain use cases, the user should be at least aware of them. For a correct assessment of the simulation and its results, knowledge about inaccuracies and uncertainties in the simulation process are crucial. Furthermore, uncertainty, especially in the analysis phase, can also make the work with the simulation and the resulting data less effective, e.g., an inaccurate interaction could increase the time for exploring the data or the risk for missing important features in the data.

In the following, uncertainty in the principal stages of the simulation and analysis process (Figure III) will be discussed from the perspective of visualization. It will be shown how visualization methods can be designed or adapted to address these uncertainties.

CHAPTER 6

Uncertainty and Visualization

Handling uncertainty is an important topic in visualization [148]. The focus of research in this context often lies on uncertainty in the data to be visualized. However, as mentioned in the introduction to this part of the thesis, uncertainty cannot only occur in the data resulting from the simulation. This chapter[1] will therefore not only discuss the visualization of uncertain data but also present two other scenarios in which uncertainty should be considered when developing the respective visualization.

Uncertainty in the parameters of the simulation model is treated in Section 6.2. By the example of simulating underground CO_2 storage, a visualization environment is presented that allows interactive exploration of the parameter space. This is enabled through the combination of model reduction and fast data processing on the GPU. Section 6.3 contains a typical uncertainty visual-

[1] **Parts of this chapter have been presented at or published in:**
M. Hlawatsch, S. Oladyshkin, and D. Weiskopf. Employing model reduction for uncertainty visualization in the context of CO2 storage simulation. Presentation at the workshop on visualization for decision making under uncertainty (part of IEEE VIS 2015): `http://vda.univie.ac.at/uncertainty2015/submissions/hlawatsch_vdmu.pdf`, 2015, last access: 2016-01-22 [8].
M. Hlawatsch, P. Leube, W. Nowak, and D. Weiskopf. Flow radar glyphs—static visualization of unsteady flow with uncertainty. *IEEE Transactions on Visualization and Computer Graphics*, 17 (12):1949–1958, 2011 [4], © 2011 IEEE.
M. Hlawatsch, F. Sadlo, and D. Weiskopf. Predictability-based adaptive mouse interaction and zooming for visual flow exploration. *International Journal for Uncertainty Quantification*, 3 (3):225–240, 2013 [9], © 2013 Begell House, Inc.

ization scenario: a novel visualization technique for uncertain time-dependent vector fields is introduced. The technique allows also a temporal analysis of uncertainty. Uncertainty in user interaction is the focus of Section 6.4, which discusses a method for handling this type of uncertainty in a flow visualization setting. The method adapts the speed of mouse interaction in dependence of the visualized data allowing a more effective exploration of the data.

6.1 Related Work

Since this chapter presents methods for three different types of uncertainty, the related work is organized accordingly.

6.1.1 Computational Steering

Unknown or uncertain input parameters of a simulation can be explored with a computational steering approach [274]. In a computational steering scenario, parameters of the simulation can be changed interactively during the runtime of the simulation, e.g., to guide the simulation into a certain direction. Computational steering is often realized by a strong coupling of simulation and visualization, with the latter providing the interface for changing the simulation parameters. An overview and discussion of computational steering can be found, e.g., in the work by Wright et al. [308]. To mention a few examples, Waser et al. [292] demonstrate the steering of a simulation for a flooding of a city. They later extended their approach for a semi-automatic exploration of the parameter space [293]. Ament et al. [24] integrate real-time FTLE computation in their steering environment. The exploration of parameter spaces is supported by the uncertainty-aware approach of Berger et al. [44]. In all these cases, the simulation run and its outcome are typically treated to be certain. This means that the uncertainty is usually not directly visualized but the user has to explore the parameter space to get an impression of the uncertainty. Therefore, special visualization techniques for uncertain data are typically not required in connection with computational steering.

6.1.2 Visualization of Uncertain Data

However, it is also possible to build simulations that incorporate uncertainty directly in their model. In this case, uncertainty is also present in the generated result. To provide an accurate representation, it is important in this scenario to include this uncertainty in the visualization of the results. Overviews of uncertainty visualization are given by Pang et al. [210], and more recently

by Brodlie et al. [53]. Potter et al. [223] present a taxonomy for uncertainty visualization, and Zuk and Carpendale [318] present a theoretical analysis of different uncertainty visualizations. A user study related to uncertainty visualization was performed by Sanyal et al. [242].

Techniques to visualize the uncertainty arising from numerical integration methods are discussed by Lodha et al. [183]. Isosurfaces in uncertain scalar fields are treated by Pfaffelmoser et al. [214] and Pöthkow et al. [222]. Wittenbrink et al. [305] describe glyph-based visualization of flow uncertainty. For the specific case of bidirectional flow fields, Zuk et al. [319] present a glyph-based visualization of uncertainty. Texture-based methods are used by Botchen et al. [49] for uncertain unsteady flow; a similar approach is applied by Allendes Osorio and Brodlie [21]. Uncertainty in vector fields can also be incorporated in vector field topology as shown by Otto et al. [207, 208]. Feature extraction methods for uncertain vector fields can be found in the work by Petz et al. [213], Otto et al. [209], and Otto and Theisel [206].

Uncertainty can be modeled with ensemble data and their visualization is therefore strongly related to uncertainty visualization. Lagrangian analysis of ensembles is investigated by Guo et al. [123] and Hummel et al. [145]. Sanyal et al. [243] visualize ensembles from weather simulations. Finally, the interpolation of uncertain data is the focus of the work by Schlegel et al. [247] and by Athawale and Entezari [31]. Pöthkow and Hege [225] discuss nonparametric models for uncertainty visualization.

6.1.3 Uncertainty in User Interaction

The handling of uncertainty in user interaction (Section 6.4) is demonstrated on the example of interactive seeding of integral curves. A comprehensive overview of streamline seeding can be found in the survey by McLoughlin et al. [190]. Laramee describes various seeding techniques for flow visualization [173, 174]. Interactive seeding in a virtual environment is part of the work by Bryson and Levit [56]. Bürger et al. [61] present a seeding approach for integral curves based on the FTLE. Interactive seeding in blood flow data is presented by Van Pelt et al. [275]. However, all these techniques do not address seeding in the context of uncertain user input.

In general, there is plenty of previous research on data-driven user input adaptation. Many examples are concerned with 1D input [18, 26, 188], e.g., for browsing through lists. Previous examples of adaptive zooming include speed-dependent zooming for browsing large documents [146], smooth zooming and panning for 2D maps [277], and automatic zooming in scrolling interfaces [76]. However, to the best of the author's knowledge, there is no specific previous

work on the uncertainty of user input in the context of Lagrangian flow visualization. The only work in this direction is by Brecheisen et al. [51], who perform sensitivity analysis for DTI fiber tracking. In contrast to the work presented in this chapter, they focus on tensor fields instead of vector fields and on stopping criteria for tracking instead of seeding and flow field analysis.

The concepts for adapting the mouse interaction in Section 6.4 are related to flow maps and the FTLE. A definition of the FTLE (Section 4.1.2) and related work (Section 4.2.3) can be found in Chapter 4.

6.2 Uncertainty in Simulation Model Parameters

In cooperation with scientists from stochastic hydrogeology within SimTech (Chapter 1), an interactive tool for visualizing data from a simulation of underground CO_2 storage [30] was developed. In this case, the uncertainty occurs in several unknown input parameters of the simulation, e.g., the percentage of barriers in the underground storage site. Instead of directly using Monte-Carlo approaches, which require a large number of simulation runs, the simulation uses a model reduction technique called polynomial chaos expansion [205].

The output of the simulation is a field of polynomials that specifies the simulated quantities for every cell of the spatial grid. Evaluating these polynomials with a specific set of input parameters provides the corresponding result. By exploiting the computational power of GPUs, it is possible to evaluate all polynomials of the full dataset fast enough to achieve interactive frame rates. Thus, it is possible to interactively change the input parameters, e.g., the injection rate of the CO_2, and directly see their effect on the result, e.g., the underground CO_2 concentration. This enables the user to explore the uncertainty of the data and the dependence of the result on the input parameters.

This approach exhibits strong similarities to computational steering approaches (Section 6.1.1). However, traditional computational steering scenarios have to rerun at least parts of the simulation every time a parameter is changed. Therefore, this is only feasible in scenarios where the simulation can be executed fast enough. In the scenario presented in this chapter, the simulation is only executed to compute the result of the model reduction—the field of polynomials. This allows a closer coupling of the simulation model and the result visualization. Furthermore, the simulation itself does not have to be fast enough for interactive steering.

6.2.1 Implementation

6.2.1.1 Data

The simulation data is defined on an almost regular grid with only small deformations of individual cells. To enable a fast processing of the data on the GPU, the simulation data was approximated with a regular grid (Figure 6.1). To reconstruct the physical appearance of the simulated storage site, a height map is applied during rendering, which distorts the regular grid accordingly.

The visualized data is defined on a regular grid of resolution $39 \times 120 \times 20$ with 100 time steps. Only 78720 of the 93600 cells are active; the polynomials have 15 coefficients in these cells. This results in approximately 9 MB of data for every time step, and around 900 MB for the full dataset.

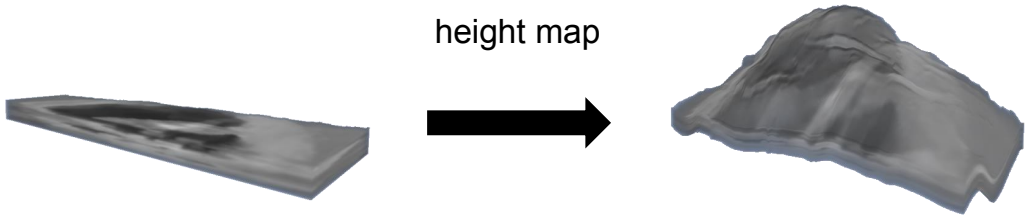


Figure 6.1 — The simulation data is internally processed on a regular grid (left). A height map is applied to imitate the physical appearance of the simulated area (right). Additionally, rock porosity is mapped to a gray scale color map.

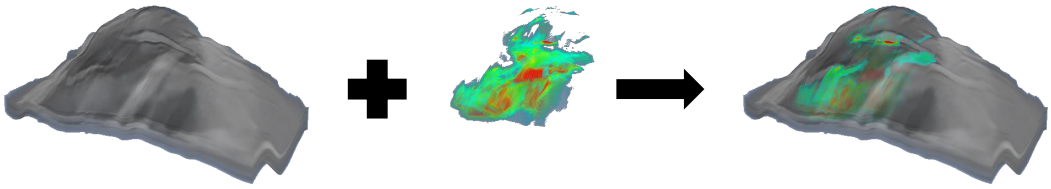


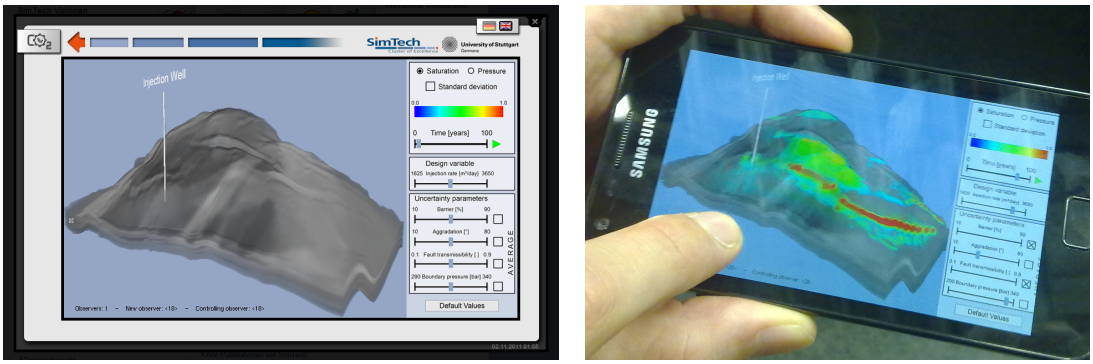
Figure 6.2 — The geological information and the computed CO_2 saturation are visually combined to provide the context of the geological setting. The visualization is generated with a standard ray casting approach.

6.2.1.2 Visualization

The implementation uses CUDA to evaluate the field of polynomials in parallel on the GPU and compute the respective scalar field. Besides the fast computation on the GPU, a further advantage is that repeated data transfers between CPU and GPU are avoided. To display the resulting scalar field, a standard GPU-based volume ray casting approach [97] without any specific optimization is used (Figure 6.2).

To provide the geological context, the resulting CO_2 saturation or pressure is combined with data describing rock porosity during visualization (Figure 6.2). For this, both datasets are sampled during ray casting and blended together with the rock porosity information mapped to transparency.

One goal of the implementation was to make it publicly available as a demonstrator for the research of SimTech. Therefore, the possibility was integrated to capture the generated images and transfer them via remote framebuffer protocol (RFB). The protocol also transfers the user interaction. In this way, the visualization tool is accessible with web browsers and mobile devices (Figure 6.3). It has to be noted that this is a server-side rendering approach, i.e., the client only displays the images generated on the server.



(a) (b)

Figure 6.3 — The developed software allows remote access to the visualization. It is possible to interact with it via (a) web browser or (b) mobile device.

6.2.1.3 Performance

Performance measurements with the implementation yielded in the following results (see Section 1.4 for hardware specification). Evaluating the polynomials on the GPU required less than 2 ms, while copying the data (GPU internally) required almost 20 ms. Hence, the bottleneck of the implementation is the memory transfer. The overall frame rate including evaluating the polynomials and rendering the data was typically around 40 fps for the view shown in Figure 6.3 with a viewport of 818×466.

6.2.2 Examples

In the following, the capabilities of the developed visualization tool are demonstrated. First of all, the different time steps of the simulation can be examined (Figure 6.4). The displayed time step can be interactively selected with a slider or the different time steps can be displayed in an animation. Loading and processing the data is performed fast enough to allow an interactive selection of time steps. The images indicate that changes in CO_2 saturation decrease with increasing time: there are larger differences in the visible structures between the first two shown time steps (Figures 6.4(a) and (b)) than between the last two (Figures 6.4(c) and (d)).

The developed tool can display two different simulated quantities: CO_2 saturation (Figure 6.5(a)) and the pressure in the storage site (Figure 6.5(b)). Furthermore, the standard deviation of the current result can be displayed for both quantities (Figure 6.5(c)). Switching between the different visualizations is possible without any notable latency.

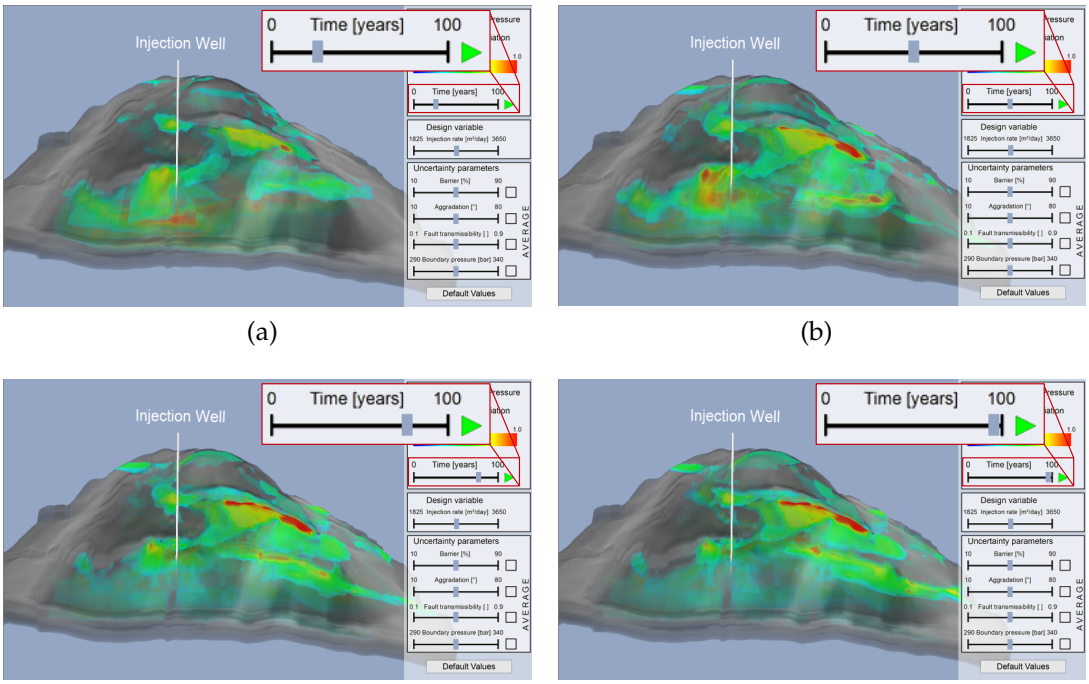
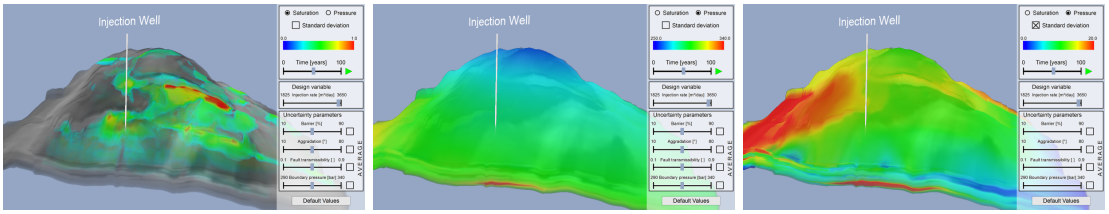


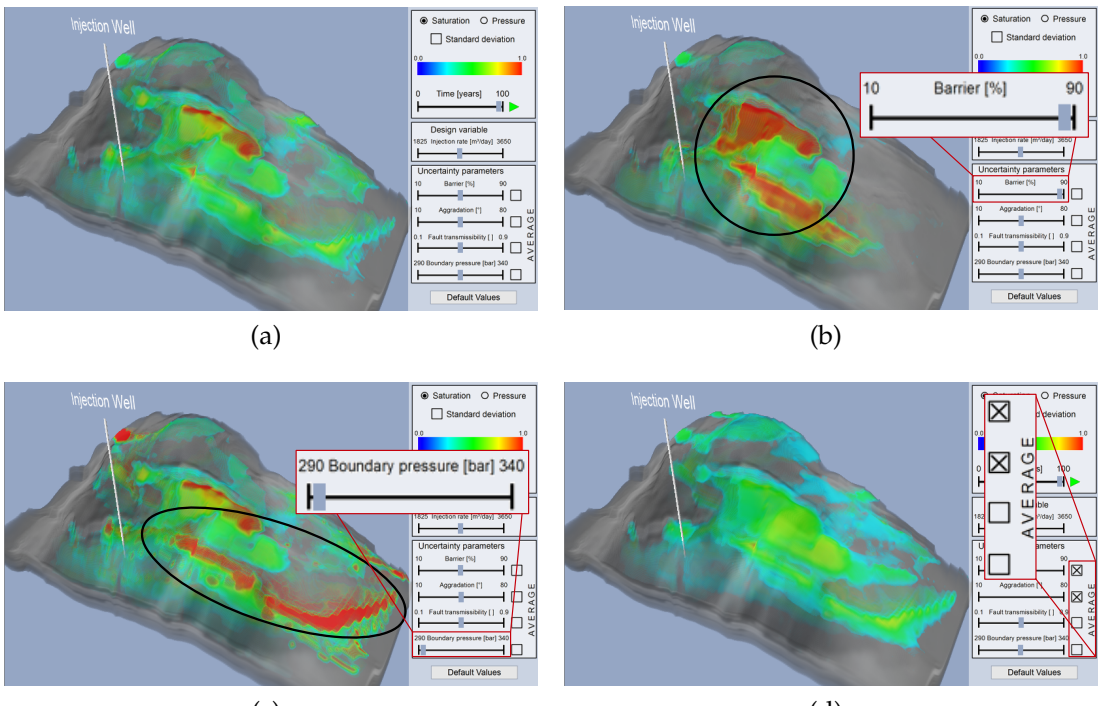
Figure 6.4 — Visualization of different time steps of the simulation result. Time steps can be manually selected with a slider (close-up) or automatically displayed as an animation sequence. CO_2 saturation is shown.

The features of the visualization described so far are not related to the uncertainty in the simulation model (except for the visualization of the standard deviation). To explore the uncertainty in the simulation model, the visualization tool has sliders for changing the simulation parameters (Figure 6.6). As previously explained, due to the model reduction approach, results for different parameters can be obtained without a repeated execution of the simulation. The visualization tool evaluates the polynomial data with the adjusted parameters on-the-fly and enables an interactive exploration of the parameter space. Figures 6.6(b) and (c) show respective examples. The images show how changing the parameters influences the CO_2 saturation in comparison to default parameters (Figure 6.6(a)). Increasing the percentage of barriers in the underground storage site results in a large area with high CO_2 saturation near the injection well (Figure 6.6(b)). Reducing the boundary pressure leads to increased CO_2 saturation in the elongate and narrow area marked in the image (Figure 6.6(c)). It is also possible to average over one or multiple parameters instead of setting them to a specific value (Figure 6.6(d)). In this case, only low to medium CO_2 saturation occurs.



(a) (b) (c)

Figure 6.5 — Visualization of different simulation properties. (a) CO_2 saturation and (b) pressure in the storage site can be displayed. (c) Additionally, the standard deviation of the result can be shown for both quantities (here for pressure). The same rainbow color map is used for all these cases. The visualization of the CO_2 saturation applies transparency to values near the minimum.



(a) (b)

(c) (d)

Figure 6.6 — Exploring the parameter space. (a) The resulting CO_2 saturation at the end of the time range with default parameters. Parameters of the simulation like (b) the percentage of barriers or (c) the boundary pressure can be interactively changed and their influence observed, e.g., areas with increased CO_2 saturation (marked black). (d) It is also possible to average parameters instead of setting them to specific values.

6.3 Uncertainty in Data

The simulation of groundwater flow is a good example of an application in which uncertainty cannot be neglected. Obtaining on-site conditions is usually expensive. Therefore, an accurate simulation model cannot be obtained but only a range of plausible parameters. In this case, it is common practice to perform Monte-Carlo simulations [166]. Monte-Carlo simulations randomly sample the parameter space and allow for stochastic conclusions about the simulated system. For instance, the average flow direction can be computed.

The average flow direction could be used to generate a vector field, which can be visualized with common flow visualization techniques. However, the average flow direction is in many cases not meaningful, e.g., if the variance is high or a normal distribution of values is not indicated. Hence, a visualization of averaged values alone is not accurate and can lead to misinterpretations.

In the following, an approach based on the previously described flow radar glyphs (Section 4.3) for visualizing uncertain vector fields is presented. It can handle uncertain direction and magnitude of flow velocity. Its utility is shown by applying it to simulation data for a groundwater remediation setup.

6.3.1 Extension of Flow Radar Glyphs

The extension of flow radar glyphs to uncertain flow fields or other data for which a range of directions is defined (instead of a unique vector per point in space and time) is quite straightforward. To cover these cases, the concept of the glyph was applied to the two limiting angles (or statistical percentiles) of the angular range of directions. The two resulting curves represent the contour of the glyph in the uncertain case. The area in between can be filled to visually represent the angular ranges (Figure 6.7(a)). Instead of single points representing single directions at given positions in space and time, an arc displays the possible range of directions. An issue of this approach is that

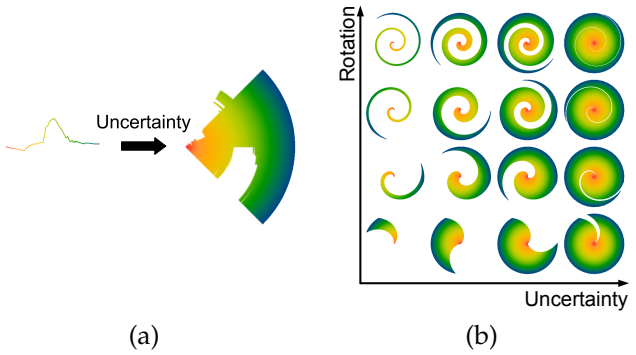
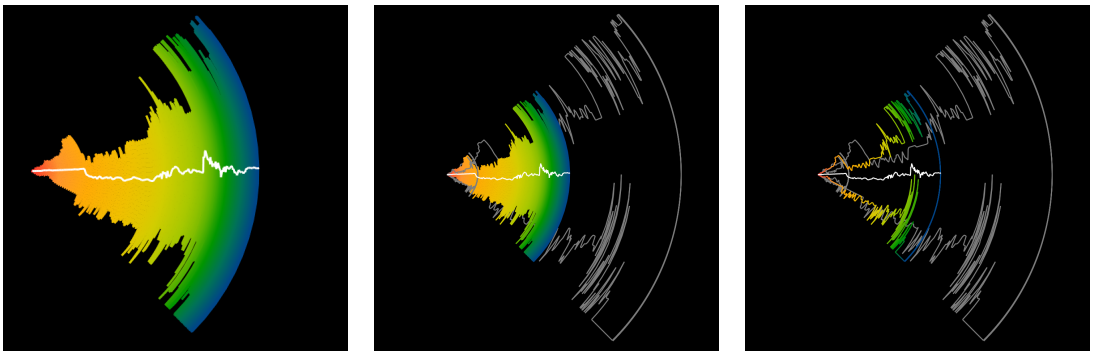


Figure 6.7 — (a) Flow radar glyphs for uncertain directions. Instead of a single curve, a filled area represents now the temporal evolution of the angular range in the case of uncertain directions. (b) Problems of the glyph when visualizing strong rotation and high uncertainty.



(a) (b) (c)

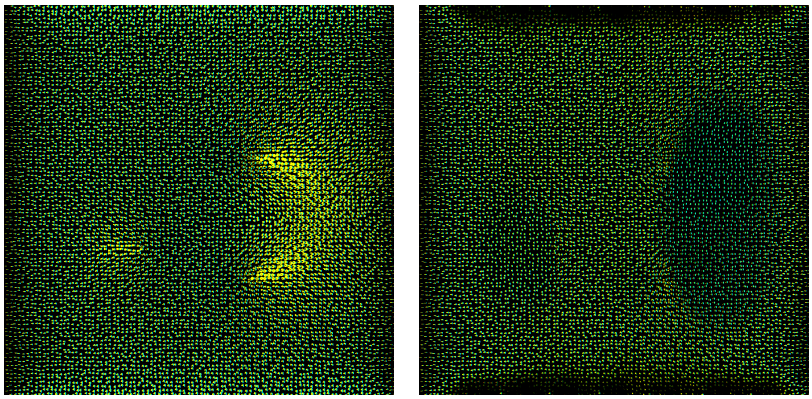
Figure 6.8 — Different types of flow radar glyphs for uncertainty. (a) This variant visualizes the range of directions as filled area. The average direction is displayed as white curve. (b) Overlaid on the glyph from (a), additional contours show the range for the minimum and maximum magnitude (both gray). (c) Variant of (b) using contours only.

the glyph is difficult to read in cases of strong rotation and high uncertainty (Figure 6.7(b)). Nevertheless, a qualitative comparison with other positions is still possible, while a quantitative analysis is more challenging.

The glyph can be further extended to visualize more aspects of uncertain vector fields (Figure 6.8). For example, the statistically averaged direction can be visualized with an additional curve (Figure 6.8(a)). If vector magnitude is uncertain and of interest, the glyph can be extended to additionally show the contours of the glyphs with lowest and highest magnitudes (Figure 6.8(b)). The contours provide an impression of the possible ranges of directions and magnitudes. To reduce visual clutter, this variant can be displayed without the filled area of the glyph that represents the average magnitude (Figure 6.8(c)).

Filling the glyph to visualize the directional range without further modifications has the effect that positions with high uncertainty appear brighter in large-scale views because they cover a larger amount of pixels of the image (Figure 6.9(a)). This is beneficial when areas with high uncertainty are of primary interest. Often, however, areas of low uncertainty should be emphasized. To achieve this, the transparency of the glyphs can be modulated according to the degree of uncertainty (Figure 6.9(b)): positions with high uncertainty are assigned high transparency and fade out.

It is suggested to use the filled type without contours (Figure 6.7), possibly with the averaged direction overlaid (Figure 6.8(a)), for large-scale analysis. The glyphs extended with the minimum and maximum magnitude should only be used on smaller scales to avoid visual clutter. The use of uncertainty-modulated



(a) (b)

Figure 6.9 — (a) Directly drawing the glyphs emphasizes positions with high uncertainty due to the larger extent of the glyphs there. (b) Modulating transparency with uncertainty attenuates these positions and emphasizes regions with low uncertainty. Both images are contrast enhanced.

transparency strongly depends on the application. Hence, it is best to let the user interactively switch between both options to provide an impression of the uncertainty in the data.

6.3.2 Groundwater Simulation with Uncertainty

6.3.2.1 Setting

In-situ chemical oxidation (ISCO) techniques with injected clean-up reactants have proven to be powerful tools in order to clean up contaminated soils [144]. Their attraction lies in conceptual simplicity and low technological effort. However, successful application requires a profound understanding and fine-tuned control of the underlying processes, most of all concerning a closed-loop flow field between injection and extraction wells for the clean-up reactants. The generic lack of knowledge about on-site conditions requires predictive simulation and evaluation prior to the field implementation. This is a non-trivial task since the system is space- and time-dependent and subject to small-scale uncertainty of material parameters that define the permeability for groundwater flow [232].

Figure 6.10 shows a typical remediation setup for in-situ oxidation. Well (A) injects the reactant into an ambient groundwater flow (from left to right). The injected reactant passes through the contaminated zone and well (B) extracts the residual reactant. The success depends, among other things, on the percentage of residual reactant that can be extracted at well (B). If the wells (A) and (B)

are not perfectly aligned with the regional ambient groundwater flow, not all the reactant injected in well (A) will be captured by the extraction well (B). To avoid this, a second extraction well (C) is drilled in order to capture the reactant missed by well (B).

In this example, the problem of an imperfect well setup is considered (e.g., occurring under seasonal variations of ambient flow). In order to reveal the system response during an injection at (A) with extraction at (B) and additional pumping at (C), the simulation is evaluated through four different periods of pumping regimes (Figure 6.10, right). Regions of interest are the so-called stagnation points (left of (A) and right of (B) and (C)), where the ambient flow and the individual flow components from the wells cancel out, leading to regions with very low velocities, very large residual times, and occurrence of all flow directions around each stagnation point.

The remediation problem is discretized on a 128×128 cell grid, and evaluated at 250 time steps covering all four periods (50 time steps for the first three periods, and 100 time steps for the last period). Since the investigated system is very flat compared to its other dimensions, and because the system force is typically uniform over the entire depth, depth-integrated (2D) simulations of flow are fully sufficient in this and many other cases. A Poisson-type partial differential equation based on Darcy's law is used for creeping (potential) flow

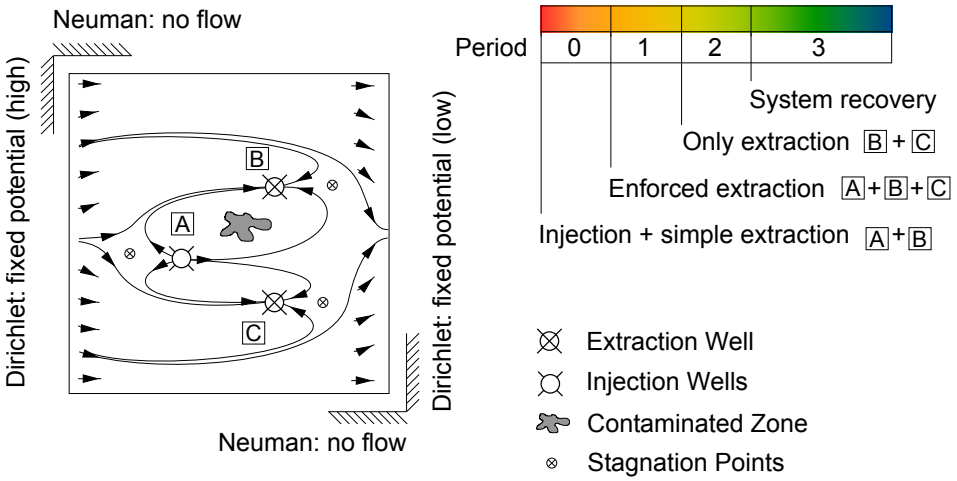


Figure 6.10 — Remediation setup (left) for in-situ oxidation with one injection (A) and two extraction (B, C) wells operating according to the pumping configuration (top right). The contaminated zone is located between (A) and (B). Boundaries are implemented as no-flow (northern and southern) and fixed potential (eastern and western) boundaries with regional ambient groundwater flow from left to right.

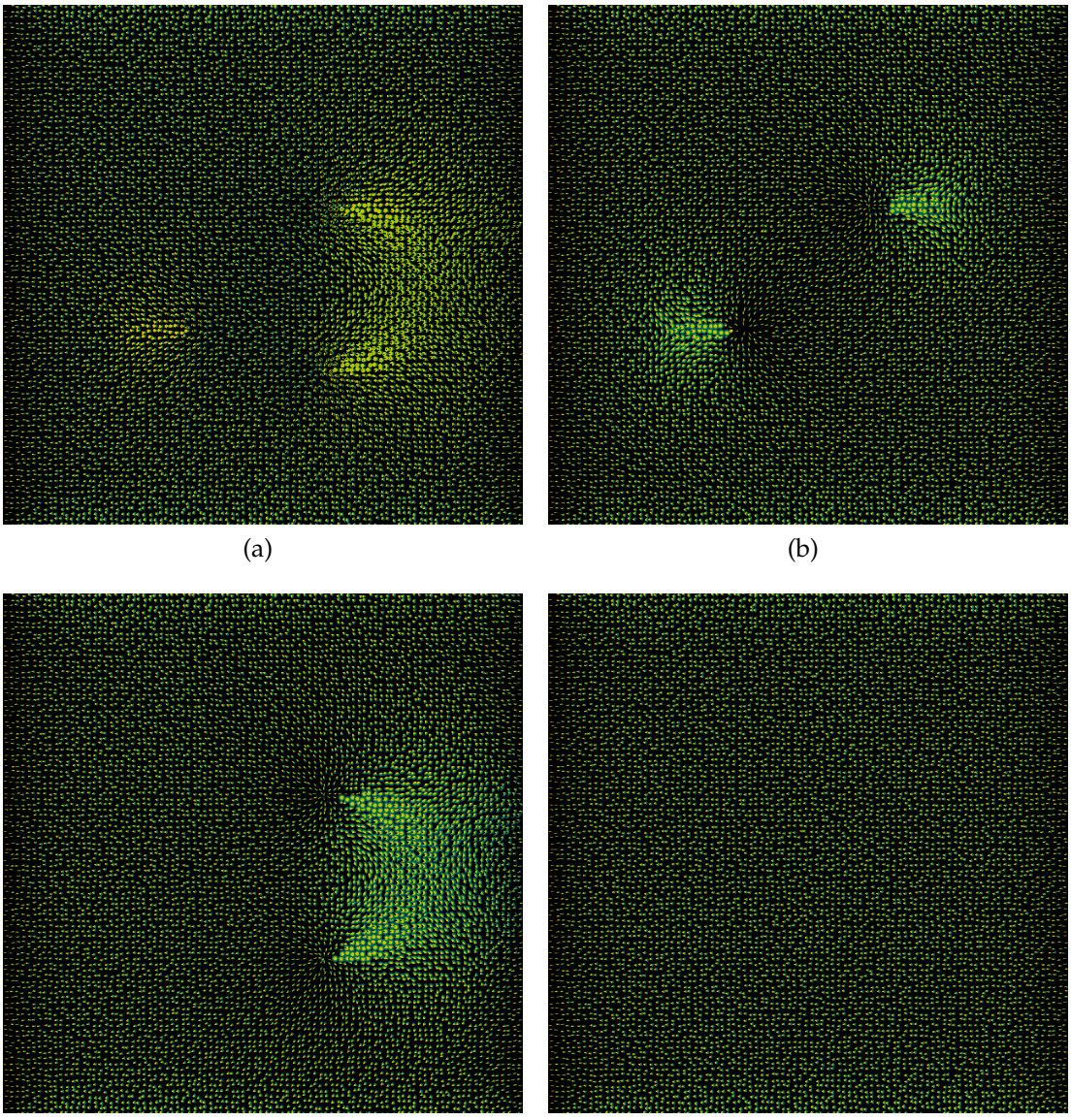
in porous media [40]. The lack of prior knowledge about on-site conditions is accounted for by 500 Monte-Carlo runs of the flow equation with randomized geostatistical permeability fields [232].

6.3.2.2 Visualization

The result of the Monte-Carlo simulation is now visualized with flow radar glyphs. Uncertainty associated with direction and velocity of flow is quantified based on the 10th and 90th percentiles of all Monte-Carlo simulation runs. Figure 6.11 shows the spatiotemporal distribution of uncertainty resolved for the periods that cover the different pumping regimes (0), (2), and (3). Figures 6.11(b) and 6.11(c) indicate that the highest uncertainty occurs close to all wells, i.e., in the regions around the possible locations of the three stagnation points (filled flow radar glyphs). Figure 6.11(d) shows the component of uncertainty due to the underlying geostatistical randomness of permeability (without any well activity). Apparently, uncertainty has a homogeneous character throughout the domain, except at the left and right borders, where the imposed boundary conditions dictate a lower uncertainty. This is obvious since the Monte-Carlo ensemble is designed to represent the manifold of possible spatial patterns and, as a matter of the input statistics, the frequency of possible outcomes is uniform throughout the domain. Figure 6.12 provides a more detailed analysis of the simulation result. It reveals that uncertainty decreases whenever and wherever the flow directions induced by any of the wells coincide with the direction (from left to right, see Figure 6.10) of the ambient background flow (Figure 6.12(d) transition to period (1)). If these individual flow components are opposed to each other, e.g., close to the stagnation points, the uncertainty in flow angle is increased (Figure 6.12(c) transition to period (1)). The reason for this is the physics of flow: all possible flow directions occur around the stagnation point, and the positions of the stagnation points change in every simulation run. A similar effect can be observed between wells (B) and (C) (Figure 6.12(b)).

6.3.2.3 Lessons Learned

From joint sessions with domain and visualization experts within SimTech, the following experiences are reported. The domain experts indicated that the combined visualization of spatiotemporal flow behavior and uncertainty by flow radar glyphs provided insight that would be much more time-consuming to obtain with their prior visualization techniques. They usually employ multiple plots with color mapping for different features of the data, e.g., uncertainty and average direction, but not a combined visualization. The domain experts learned how to work with the glyph visualization just within minutes, i.e., there



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

Figure 6.11 — Uncertainty of flow direction based on the 10th and 90th percentiles of all Monte-Carlo runs. Normalized glyphs are seeded on a regular grid. To highlight uncertainty, no transparency modulation (see Section 6.3.1) is used. Different time ranges are shown with adapted color mapping (see Section 4.3.1.2)—(a) full time range, (b) period (0), (c) period (2), (d) period (3).

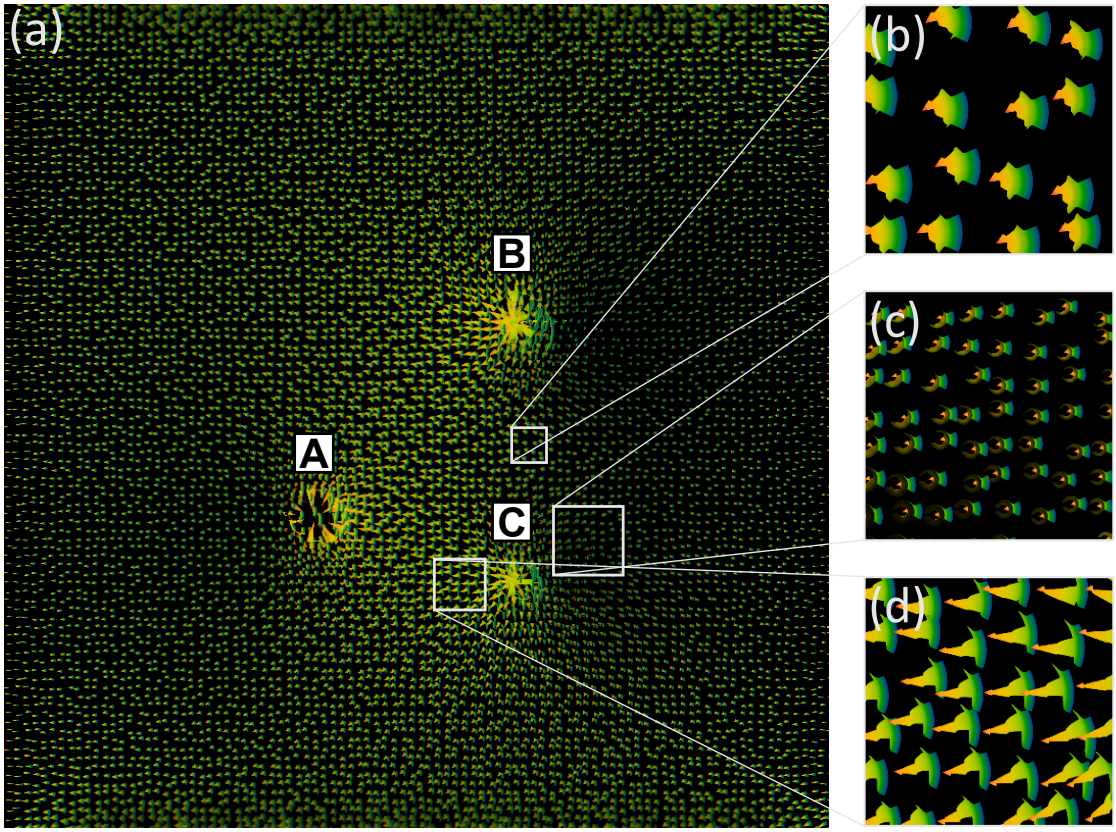


Figure 6.12 — Detailed analysis of flow uncertainty. Magnitude-scaled glyphs with transparency modulation (see Section 6.3.1) are seeded on a jittered regular grid. (a) Overview with the following close-up areas marked: (b) below well (B), (c) right of well (C), and (d) left of well (C). See Figure 6.10 for well denotations.

is no steep learning curve. They saw the main application in understanding boundary conditions (even for arbitrary complex geometries), force terms that vary in space and time, and arbitrary complex input statistics. They reported that the technique could improve the analysis of complex, uncertain, dynamic, and distributed systems. Beside scientific purposes, they suggested the glyph visualization as a powerful didactical tool in education. Their main criticism was that details of the temporal evolution get lost on the overview level.

6.4 Uncertainty in User Interaction

Uncertainty in user interaction can occur on two sides. On the input side, uncertainty can emerge from insufficient control or insufficient (e.g., only pixel-accurate) precision of the input device with respect to the visualization task. Furthermore, uncertainty is also present on the output side, i.e., in the display. There, the fixed resolution of the display can lead to uncertainty due to discretization errors. These two aspects are addressed in the following in the context of interactive flow visualization and exploration.

Focusing on interactive visualization with a feedback loop (Figure 6.13), means are provided that adapt the user's mouse input to its uncertainty. The effectiveness of the interaction loop is further improved by reducing the uncertainty on the output side with a data-driven zoom lens. Both methods help the users reach their aimed result faster, i.e., to gain relevant information for their task in a more efficient way. Considering the exploration of flow fields, a prominent example of an aimed result is the determination where a quantity, such as heat, is transported from.

Since the main application context is the visualization of vector fields, the presented technique is based on interactive visualization using integral curves such as streamlines and pathlines. The uncertainty with respect to user input poses a predictability problem in that context. User input includes the seeding location, seeding time, and integration length of the curves. In the Lagrangian framework, often the end points of integral curves are of special interest since they provide information about transport and interrelation. In terms of the predictability problem, the perturbations are associated with the user input, and

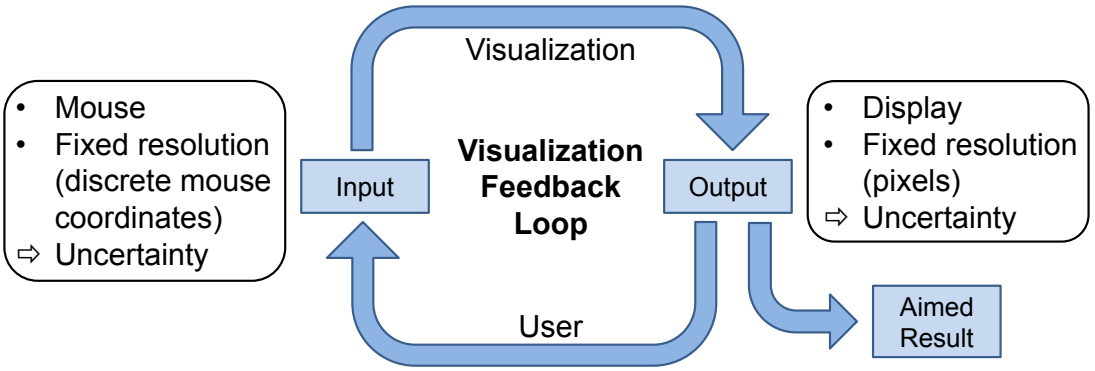


Figure 6.13 — Convergence to aimed result within the feedback loop of interactive visualization. Convergence speed is affected by uncertainty on the input and output side.

it is of interest how these perturbations influence the resulting integral curves, in particular their end points. Since both seeding time and advection time are typically entered numerically, these two sources of error are not addressed but it is focused on the seed placement using the mouse.

A prominent and widely used concept for quantifying predictability is the Lyapunov exponent (in problems with infinite time domain) and its variant for finite-time domains called finite-time Lyapunov exponent (FTLE) (see Section 4.1.2). Since the Lyapunov exponent is identical to the FTLE in its special case of infinite advection time, the presented approach is built upon the FTLE for generality. The FTLE represents a conservative measure for the growth of perturbations over finite advection time intervals (Figure 6.14). Hence, it provides a quantitative measure of the deviation of the end points of streamlines or pathlines with respect to the uncertainty of their seeds—constituting the basis of the proposed predictability-based interactive flow visualization technique.

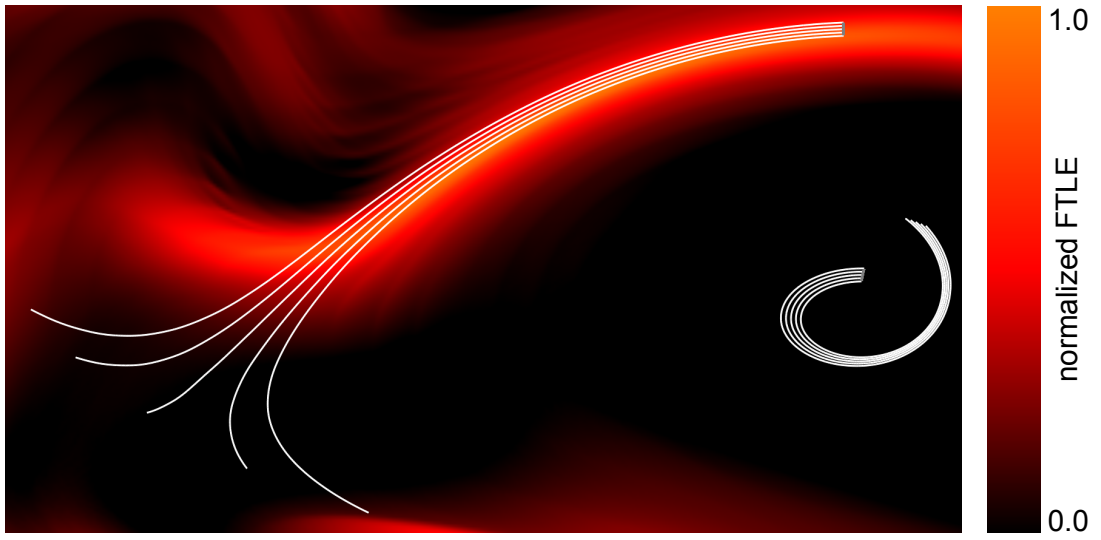


Figure 6.14 — The FTLE and predictability. (a) Nearby seeded pathlines started in regions with low FTLE values (center right) exhibit coherent behavior, whereas identically seeded pathlines started at locations with high FTLE (upper right) exhibit divergent behavior. Hence, the predictability of pathline behavior is low when interactively seeding in areas of high FTLE. (b) This color map is used for FTLE visualization throughout this chapter. Negative FTLE values are clamped to zero prior to normalization.

6.4.1 Uncertainty-Aware Interaction

The method presented in the following addresses two topics: predictability-based interaction (Section 6.4.1.1) and zooming (Section 6.4.1.3). These building blocks can be used separately, but provide synergies when combined.

6.4.1.1 Predictability-Based Mouse Interaction

The adaptive interaction is based on predictability by introducing a data-driven mapping from the mouse input to the mouse cursor. Since the FTLE represents unpredictability, the basic idea is to decelerate the mouse according to the FTLE field. This improves interaction by providing better control of seed placement in regions with low predictability, i.e., high uncertainty of the result with respect to the user input. Furthermore, it allows for more coherent and convergent interactive visualization sessions. Compared to the traditionally interleaved zooming phases for seed placement and visualization phases for investigating integral curves (Figure 6.13), the proposed adaptive method avoids loss of context that would be introduced by switching between different views and scales. It provides a coherent yet accurate exploration. Nevertheless, the adaptive method does not replace traditional mouse interaction. It turned out in the conducted user study (Section 6.4.1.5) that standard interaction is often preferred for navigation and coarse exploration, whereas the adaptive approach serves well for detailed analysis. Therefore, it is proposed that the user is allowed to freely switch between standard interaction and the adaptive technique to combine the best of the two.

Coordinate Systems The adaptation approach involves three different coordinate systems: (mouse) input coordinates $\mathbf{m} \in \mathbb{Z}^p$ from the operating system, sub-pixel accurate internal cursor coordinates $\tilde{\mathbf{m}} \in \mathbb{R}^p$, and the physical coordinates $\mathbf{x} \in \mathbb{R}^n$ of the respective n-dimensional dataset. Common desktop mouse interaction results in $p = 2$, whereas interaction with a 3D mouse involves $p = 3$. The data-driven mapping $\mathbf{m} \mapsto \tilde{\mathbf{m}}$ is provided by the proposed technique, whereas the mapping $\tilde{\mathbf{m}} \mapsto \mathbf{x}$ is accomplished by a simple transformation $\mathbf{x} = \phi(\tilde{\mathbf{m}})$, e.g., by back projection. Due to the adaptive nature of the approach, an explicit mapping $\mathbf{m} \mapsto \tilde{\mathbf{m}}$ is not maintained but changes are mapped: After obtaining the input vector $\Delta\mathbf{m}_i = \mathbf{m}_{i+1} - \mathbf{m}_i$, the mapping $\Delta\mathbf{m}_i \mapsto \Delta\tilde{\mathbf{m}}_i$ is applied and $\tilde{\mathbf{m}}_{i+1} = \tilde{\mathbf{m}}_i + \Delta\tilde{\mathbf{m}}_i$ is computed. The corresponding physical coordinates are evaluated by $\mathbf{x}_{i+1} = \phi(\tilde{\mathbf{m}}_{i+1})$.

Adaptation Assuming small changes $\Delta\mathbf{m}_i$, $\Delta\tilde{\mathbf{m}}_i$, and $\Delta\mathbf{x}_i$, a first-order approximation by linearization with infinitesimal vectors $\mathrm{d}\mathbf{m}_i$, $\mathrm{d}\tilde{\mathbf{m}}_i$, and $\mathrm{d}\mathbf{x}_i$ can be used:

$$\mathrm{d}\mathbf{x}_i = (\nabla\phi(\tilde{\mathbf{m}}_i))\mathrm{d}\tilde{\mathbf{m}}_i. \tag{6.1}$$

From this follows with the definition $s_\phi(\tilde{\mathbf{m}}_i) := \|(\nabla\phi(\tilde{\mathbf{m}}_i))\mathrm{d}\tilde{\mathbf{m}}_i\| \,/\, \|\mathrm{d}\tilde{\mathbf{m}}_i\|$:

$$\|\mathrm{d}\mathbf{x}_i\| = s_\phi(\tilde{\mathbf{m}}_i)\,\|\mathrm{d}\tilde{\mathbf{m}}_i\|\,. \tag{6.2}$$

It is now assumed that $\phi(\tilde{\mathbf{m}})$ is restricted to translation, rotation, and uniform scaling, valid and constant for the whole scene. In the case $p = n - 1$, e.g., 2D mouse controlling a position in 3D space, $\mathbf{x}$ is assumed to stay on a user-defined hyperplane. With these assumptions, $s_\phi(\tilde{\mathbf{m}}_i)$ is a constant scalar factor, i.e.,

$$\|\mathrm{d}\mathbf{x}_i\| = s_\phi\,\|\mathrm{d}\tilde{\mathbf{m}}_i\| = \|(\nabla\phi(\tilde{\mathbf{m}}_i))\|\,\|\mathrm{d}\tilde{\mathbf{m}}_i\|\,. \tag{6.3}$$

For the data-driven adaptation, the growth factor $\gamma(\mathbf{x}_i)$ of a perturbation $\mathrm{d}\mathbf{x}_i$ is required, seeded at position $\mathbf{x}_i$ and resulting in the perturbation $\mathrm{d}\boldsymbol{\xi}_i$ after advection. The perturbation corresponds to the motion of the trajectory's seed from position $\mathbf{x}_i = \phi(\tilde{\mathbf{m}}_i)$ due to cursor displacement $\mathrm{d}\tilde{\mathbf{m}}_i$:

$$\|\mathrm{d}\boldsymbol{\xi}_i\| = \gamma(\mathbf{x}_i)\,\|\mathrm{d}\mathbf{x}_i\| = \gamma(\phi(\tilde{\mathbf{m}}_i))s_\phi\,\|\mathrm{d}\tilde{\mathbf{m}}_i\|\,. \tag{6.4}$$

For the adaptation, it is now required that the size of the perturbation $\mathrm{d}\boldsymbol{\xi}_i$ is proportional to the size of the input vector $\mathrm{d}\mathbf{m}_i$, with user defined ratio s_u controlling the speed of the end point of the trajectory:

$$\|\mathrm{d}\boldsymbol{\xi}_i\| = \gamma(\phi(\tilde{\mathbf{m}}_i))s_\phi\,\|\mathrm{d}\tilde{\mathbf{m}}_i\| \stackrel{!}{=} s_u\,\|\mathrm{d}\mathbf{m}_i\|\,, \tag{6.5}$$

leading to

$$\|\mathrm{d}\tilde{\mathbf{m}}_i\| = (s_u/s_\phi)\,\|\mathrm{d}\mathbf{m}_i\| \,/\gamma(\phi(\tilde{\mathbf{m}}_i)). \tag{6.6}$$

Hence, the growth factor $\gamma(\phi(\tilde{\mathbf{m}}_i))$ needs to be determined for the adaptation.

A straightforward approach would be to derive $\gamma(\mathbf{x})$ from the FTLE. In this case, the maximum growth factor from the definition of the FTLE (Section 4.1.2) would be used:

$$\gamma_F(\mathbf{x}) = \sqrt{\lambda_{\max}(\mathbf{C}^T_{t_0}(\mathbf{x}))} = \exp(\sigma^T_{t_0}(\mathbf{x}) \cdot T) \tag{6.7}$$

However, as noted in Section 4.1.2, the FTLE represents only an upper bound of a typically anisotropic growth property: differently oriented perturbations usually undergo different growth rates. Thus, the above adaptation would not depend on the orientation of $\mathrm{d}\mathbf{m}$.

Instead, the aim is an adaptation of the mouse input speed such that, given constant mouse motion speed, the end point of the seeded trajectory also moves at constant speed. The aforementioned straightforward approach would accomplish this only for mouse motion along the maximizing perturbation. Thus, the adaption has to be based on the effective growth factor $\gamma_G(\mathbf{x})$ of a perturbation corresponding to $\mathrm{d}\mathbf{m}$ at $\mathbf{x} = \phi(\tilde{\mathbf{m}})$. This can be achieved using the flow map gradient $\nabla \boldsymbol{\xi}_{t_0}^T(\mathbf{x})$:

$$\gamma_G(\phi(\tilde{\mathbf{m}})) = \left\| \nabla \boldsymbol{\xi}_{t_0}^T(\phi(\tilde{\mathbf{m}})) \frac{\nabla \phi(\tilde{\mathbf{m}}) \mathrm{d}\mathbf{m}}{\|\nabla \phi(\tilde{\mathbf{m}}) \mathrm{d}\mathbf{m}\|} \right\|. \tag{6.8}$$

Here, it is assumed that the adaptation does not change the direction of the mouse input, i.e., $\mathrm{d}\tilde{\mathbf{m}} \propto \mathrm{d}\mathbf{m}$.

Similarly, for the discretized computation, the adapted motion vector $\Delta\tilde{\mathbf{m}}_i$ has the same direction as the input vector $\Delta\mathbf{m}_i$ and is only scaled according to Equation (6.6):

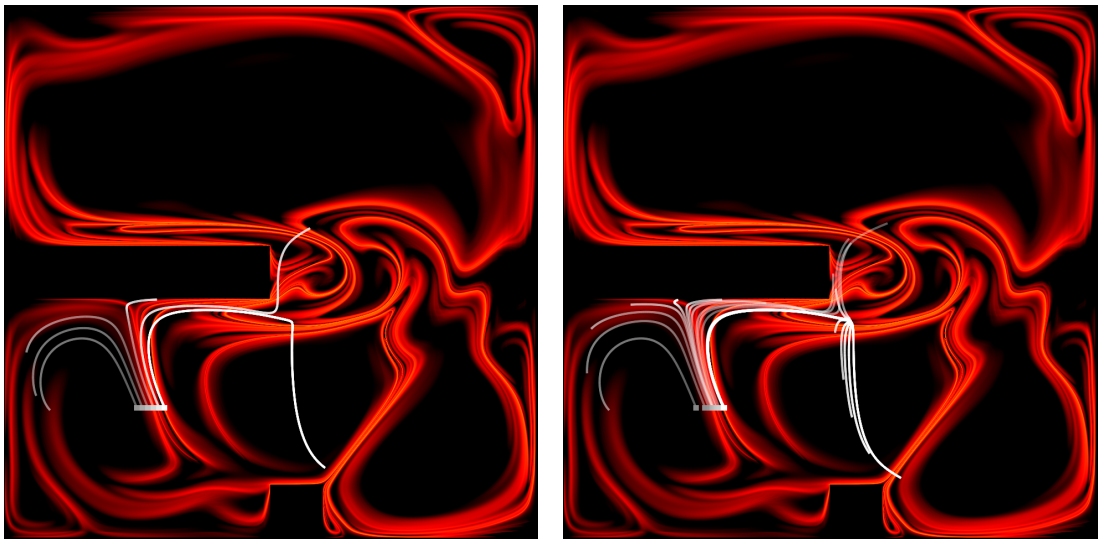
$$\Delta\tilde{\mathbf{m}}_i = (s_u / s_\phi) \Delta\mathbf{m}_i / \gamma(\phi(\tilde{\mathbf{m}}_i)). \tag{6.9}$$

If $\Delta\mathbf{m}_i$ is too large for approximation by linearization, the adaptation is split into k parts by subdividing $\Delta\mathbf{m}_i$ and applying Equation (6.9) for each of its sub steps $\Delta\mathbf{m}_i' = \Delta\mathbf{m}_i / k$. Large k increase the accuracy, but also the computational effort, hence, k has to be limited to maintain interactive rates. In the experiments for this work, $k = 5$ led to good results.

Figure 6.15 illustrates how the adaptation method works in an interactive exploration application based on pathlines. The results were generated with simulated mouse input to assure comparability. Figure 6.15(b) shows how cursor motion is decelerated when moving into areas of low predictability. This helps analyze the flow structure, incoherent motion of the end point of the pathline is avoided. In regions of high predictability, the cursor can move faster, which allows a fast exploration without the risk of missing important details. Note that the growth factor $\gamma(\mathbf{x})$ is clamped to 1.0 if $\gamma(\mathbf{x}) < 1.0$ for better usability. This limits the cursor speed $\Delta\tilde{\mathbf{m}}$ to the input speed $\Delta\mathbf{m}$ to avoid that very small mouse motions result in uncontrollable large cursor motions.

6.4.1.2 2D and 3D Applications

The description so far holds for adapting mouse motion in datasets of arbitrary dimension n. The only part that does not generalize in a straightforward manner is the mapping $\phi(\cdot)$ from input coordinates to physical coordinates. In 2D, $\phi(\cdot)$ typically represents translation and uniform scaling only. In the case of 3D data and if a 3D input device is used, $\mathbf{m} \in \mathbb{R}^3$ and $\tilde{\mathbf{m}} \in \mathbb{R}^3$. In this case, $\phi(\cdot)$ may reduce to the identity map $\phi(\tilde{\mathbf{m}}) \equiv \tilde{\mathbf{m}}$ in the formulation. However, if 2D



(a) (b)

Figure 6.15 — Comparison of (a) direct and (b) adaptive mouse input for interactive exploration with pathlines. For both images, the same simulated mouse input with constant speed and direction was used. The FTLE field is shown in the background. The direct method exhibits uniform motion of the pathline seed, whereas the adaptive method exhibits uniform motion of its end point. The temporal sequence of pathline motion is shown in a single image with older steps having higher transparency.

mouse input is used for 3D datasets, $\phi(\cdot)$ maps from $\mathbb{R}^2$ to $\mathbb{R}^3$. This mapping requires the definition of depth. For this, a common approach is proposed: a user-defined hyperplane to which the cursor is aligned, called "exploration plane" in the following (Figure 6.20).

6.4.1.3 Predictability-Based Zoom Lens

There are also situations in which uncertainty on the *output* side hinders convergence of the interactive visualization feedback loop (Figure 6.13), e.g., due to the discretization of the display into pixels. For this case, a zoom lens with data-driven magnification is introduced. In accordance with the overall approach, the lens is kept centered at the (sub-pixel accurate) mouse cursor position. At the same time, this improves convergence of interactive visualization by automatically adjusting views and scales. In contrast to the adaptation of the mouse motion, in which anisotropic adaptation (Equation (6.8)) based on $\nabla \boldsymbol{\xi}_{t_0}^T(\mathbf{x})$ provides adequate behavior, *isotropic* magnification adaptation is employed for the lens, based on the FTLE. In the prototype implementation,

the radius of the lens is kept fixed. However, it would be straightforward to implement a lens with a user-controlled radius.

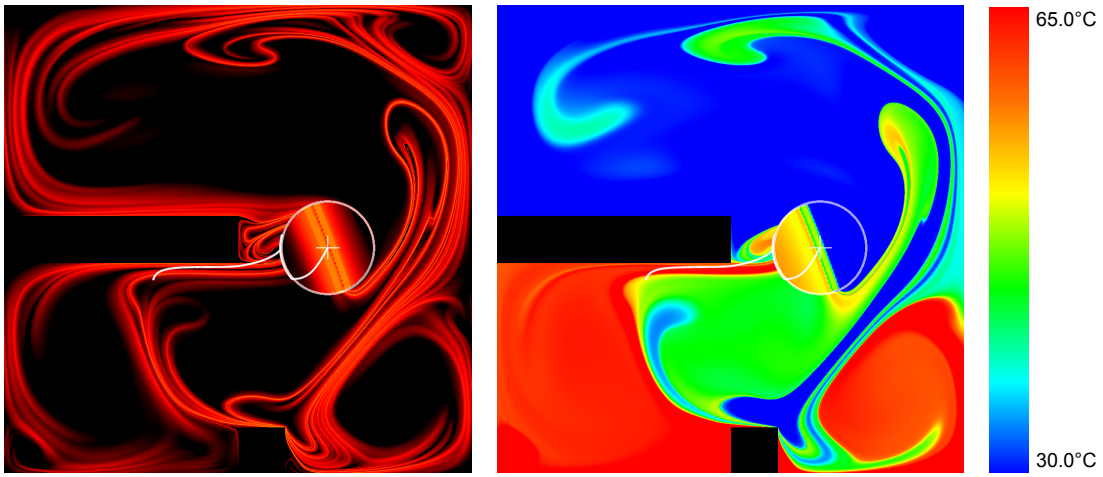
The zoom lens can be freely turned on and off in the implementation. When active, the lens is kept centered with $\phi(\tilde{\mathbf{m}})$, the position of the sub-pixel accurate mouse cursor. The FTLE $\sigma^T_{t_0}(\phi(\tilde{\mathbf{m}}))$ is also evaluated there and the magnification factor of the lens is adjusted to

$$f_z \cdot \sigma^T_{t_0}(\phi(\tilde{\mathbf{m}})) \tag{6.10}$$

during interaction. The parameter f_z defaults to 1 and allows the user to adapt the range of magnification to their needs.

From a symmetry point of view (Figure 6.13), the straightforward choice for controlling the magnification level of the lens would use the growth factor γ_F (Equation (6.7)), instead of $\sigma^T_{t_0}(\phi(\tilde{\mathbf{m}}))$. Although this would result in a motion of the content within the lens conforming to the speed of the end point of the respective trajectory (the speed factor between the seed and the end point would be compensated by the lens magnification), it would exhibit a major drawback: since separation factors in the flow map typically vary by several orders of magnitude, the same highly dynamic variation would appear in the magnification—leading to incoherent visualization. The logarithmic mapping inherent in the FTLE (Equation (4.9), Section 4.1.2, page 73) and the independence from the orientation of mouse motion calm down the temporal magnification change and motivate Equation (6.10). In Section 6.4.2.2, results for the zoom lens with direct and with adaptive mouse input are presented. In the experiments with 3D datasets, the lens is kept aligned to the exploration plane (see Figure 6.20(c)). Since the magnification operates on this plane, the FTLE $\sigma^T_{t_0}(\phi(\tilde{\mathbf{m}}))$ in Equation (6.10) is computed by projecting the flow map gradient $\nabla\boldsymbol{\xi}^T_{t_0}(\mathbf{x})$ to the plane.

A prominent scenario for the zoom lens, also present in the conducted user study (Section 6.4.1.5), is the inspection of finely folded FTLE ridges (Figure 6.16(a)). Trajectories started in these regions typically undergo massive separation, and it is often unclear where the sub-pixel accurate mouse cursor is located, i.e., at which of the finely folded FTLE ridges the trajectory was started. Moving with the sub-pixel accurate mouse cursor and adapting the magnification to the FTLE, the zoom lens provides appropriate view and scale for interpreting the underlying dynamics. Another application for the lens is the visualization of delocalized criteria [108] or the advection of quantities such as vorticity [237], which also exhibit fine-scale structures related to pathline behavior (Figure 6.16(b)).



(a) (b) (c)

Figure 6.16 — Applications for the adaptive zoom lens. (a) Analysis and interpretation of ridge lines in the FTLE field. (b) Visualization of delocalized temperature [108], i.e., temperature averaged along pathlines. (c) This color map is used for all images with temperature distribution.

6.4.1.4 Implementation

The implemented prototype uses OpenGL for rendering, GLUT for platform-independent management of the graphical user interface, and CUDA for fast computation of flow maps and integral curves on the GPU. Fast computations are important for performing the adaptation process at interactive frame rates and enable smooth interactions. Furthermore, the mouse cursor provided by the window manager was disabled and replaced with an application specific mouse cursor to gain control over the cursor behavior and speed. A more detailed description of these implementation aspects can be found in the original paper [9].

6.4.1.5 User Study

A user study was conducted with nine domain experts from the field of CFD simulation and flow visualization to assess the usefulness of the adaptation method. Adaptive mouse input was compared to direct mouse input and the utility of the adaptive zoom lens was tested. A detailed description of the study procedure and results are presented in [9]. The results are summarized in the following.

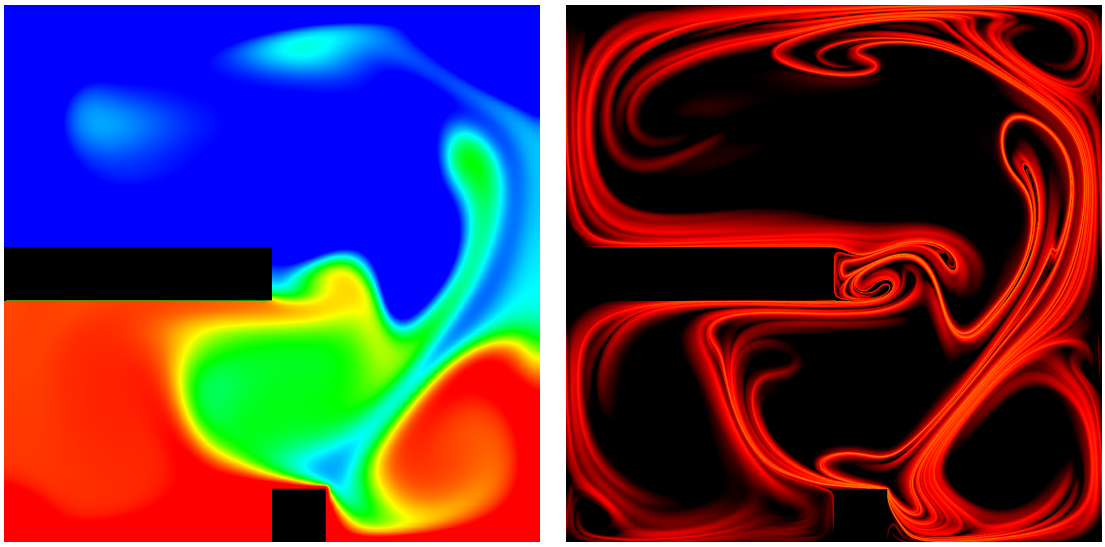
There is no clear winner for explorative tasks. Even participants that preferred the adaptive method liked to initially explore the data with the direct method.

The experts who preferred the direct method said that the adaptive mode was too slow and that they did not need its precision for such tasks. When it comes to detailed analysis of flow behavior at specific positions, e.g., the analysis of transport or FTLE ridges, most participants preferred adaptive mouse input. They liked its higher precision and the sensitivity to the underlying data. For the analysis of FTLE fields, most participants saw a benefit in the adaptive zoom lens. Observations during the study showed that the reasons for slowing down the mouse are not always clear to the user if the FTLE is not visible.

The participants were also asked to rate the interactive placement of pathlines. They all rated it very helpful. Even when an algorithm can generate a meaningful seeding, which is difficult for pathlines, they would like to further explore the data with pathlines and place additional lines.

6.4.2 Examples

The method is demonstrated for one 2D time-dependent dataset and one 3D stationary dataset. More results can be found in the original paper [9]. The Hotroom2 dataset from Section 4.4.4.1 is used as a 2D example. It resulted from a CFD simulation of buoyant air flow in a closed container with barriers, where the bottom is heated and the top cooled (Figure 6.17). Its spatial resolution is 101 $\times$ 101 and it spans 1001 time steps. The second dataset is a 3D CFD simulation



(a) (b)

Figure 6.17 — Overview of the Hotroom2 dataset. (a) Temperature distribution and (b) the corresponding reverse ($T < 0$) FTLE field are shown.

of a static mixer. It features two tangential inlets, one for hot air and one for cold, and an outlet for the mixed flow. For the GPU-based implementation, the unstructured grid was resampled at a spatial resolution of $201 \times 301 \times 201$.

The seeds of the pathlines are marked by small squares in all images. The temporal sequence of a moving pathline is merged into a single image and transparency is used to emphasize temporal progression with older pathlines exhibiting higher transparency.

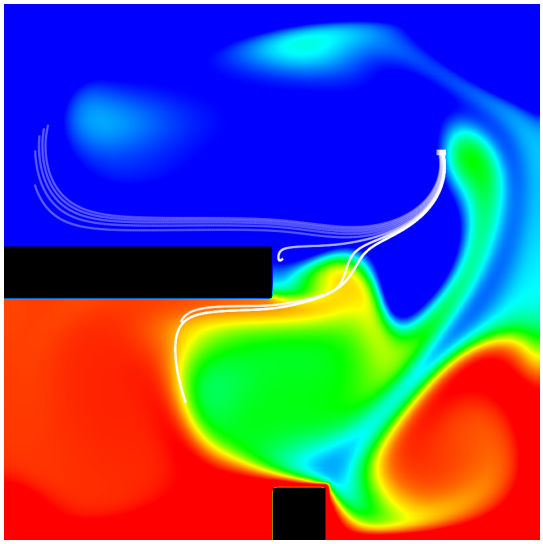
6.4.2.1 Two-Dimensional Visualization

In the case of the Hotroom2 dataset, the temperature distribution was examined (Figure 6.17). Warm air is advected from the bottom wall, whereas cold air is transported from the top wall. Concurrently, thermal conductivity leads to diffusion, equaling temperature distribution. The hot and cold air is predominantly mixing in the lower right quarter. The correlation between temperature and the reverse ($T < 0$) FTLE is apparent.

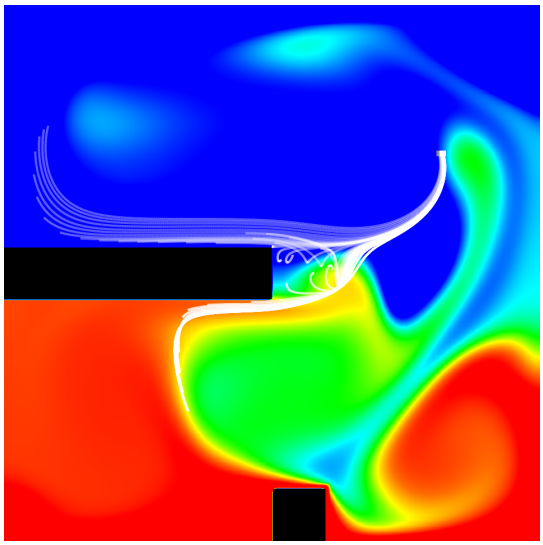
The results from interactive exploration with a pathline are shown in Figure 6.18. Reverse pathlines show where a quantity is advected from. Here, direct mouse input was again compared with the proposed adaptive method. To obtain comparable results, the same generated mouse input was used. Using direct mouse input, the accuracy is bounded by the screen resolution (Figure 6.18(a)). In this case, it is difficult to analyze the mixing behavior of the flow. With the adaptive method, the same user input results in a much higher accuracy of the seeding (Figure 6.18(b)). The pathline behavior can be clearly identified and tracked. As visible in the image, the adaptation provides a very dense sampling of the flow behavior. Therefore, the user is able to use faster mouse input without missing important details (Figure 6.18(c)).

6.4.2.2 Zoom Lens

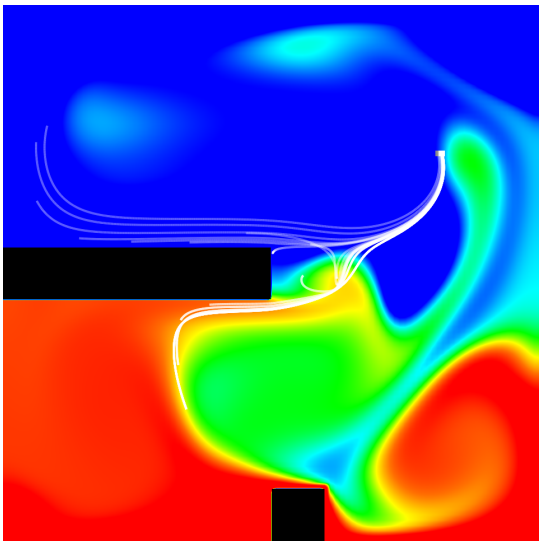
The predictability-driven zoom lens is particularly useful for inspecting transport phenomena in quantities that exhibit fine structures. One such example is the investigation of delocalized temperature [108], shown in Figure 6.16 for the Hotroom2 dataset. Another example is the analysis of FTLE fields. Figure 6.19 demonstrates the usage of the adaptive zoom lens for the analysis of the FTLE field in the same dataset. The adaptive zoom magnifies regions with high FTLE values and low predictability. It is apparent in Figure 6.16 that regions with different delocalized temperature values correspond to regions of qualitatively different behavior and hence their interfaces correspond to FTLE ridges. The FTLE-dependent magnification is also highly useful at positions with neighboring ridge lines (Figure 6.16). In combination with the visualization of the



(a)



(b)



(c)

Figure 6.18 — Results for the Hotroom2 dataset with reverse pathlines. Simulated pathline seed motion with direct mouse input at the highest accuracy (single pixel steps) is visualized in (a). Image (b) shows the result with the adaptive method applied to the same mouse input. Because of the deceleration by the adaptation, typical user input would consist of faster motion. A possible result may then look like the result in (c). It can be clearly seen that cold air was advected from the upper half, whereas hot air was entrained from the left lower quarter.

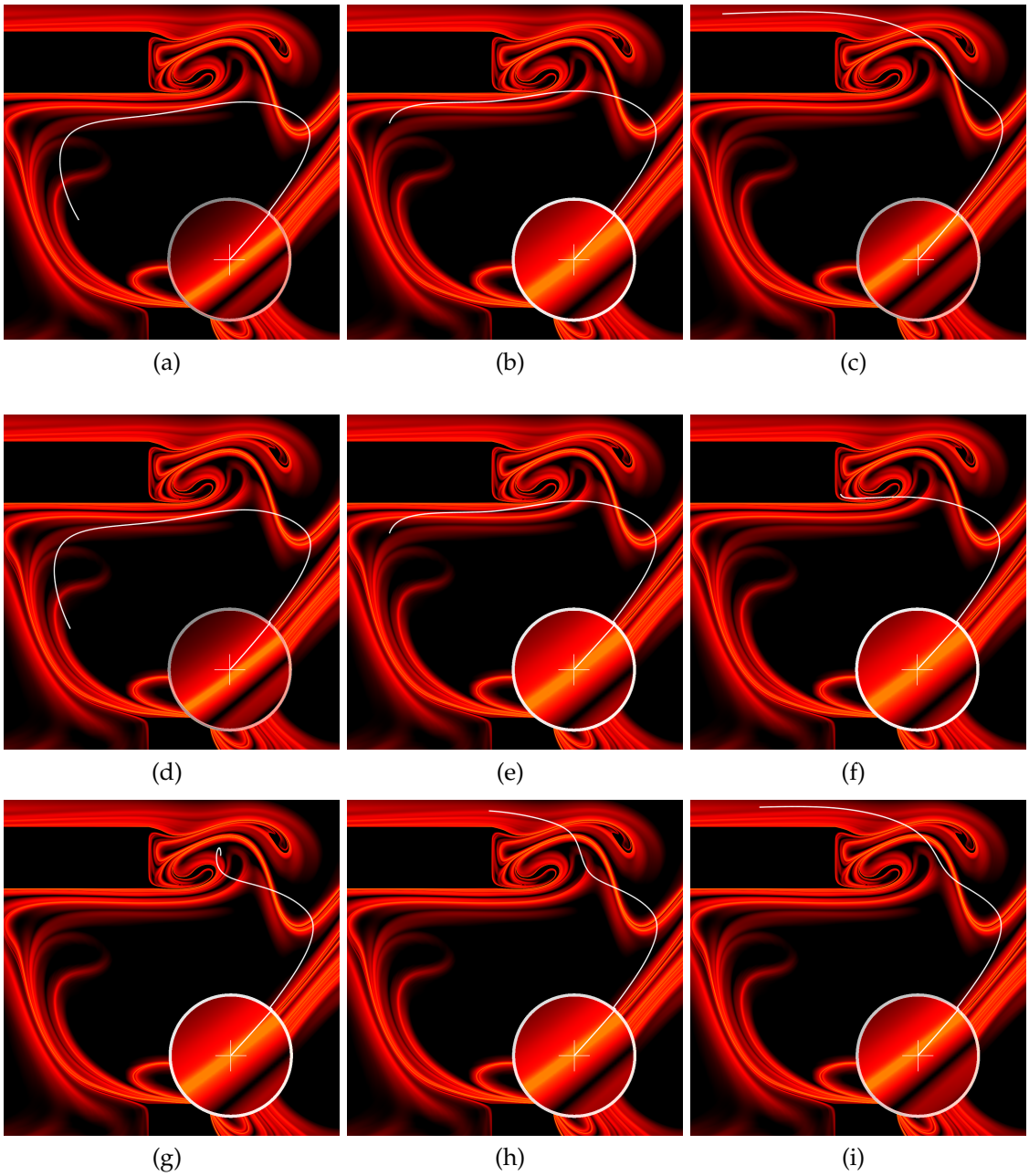
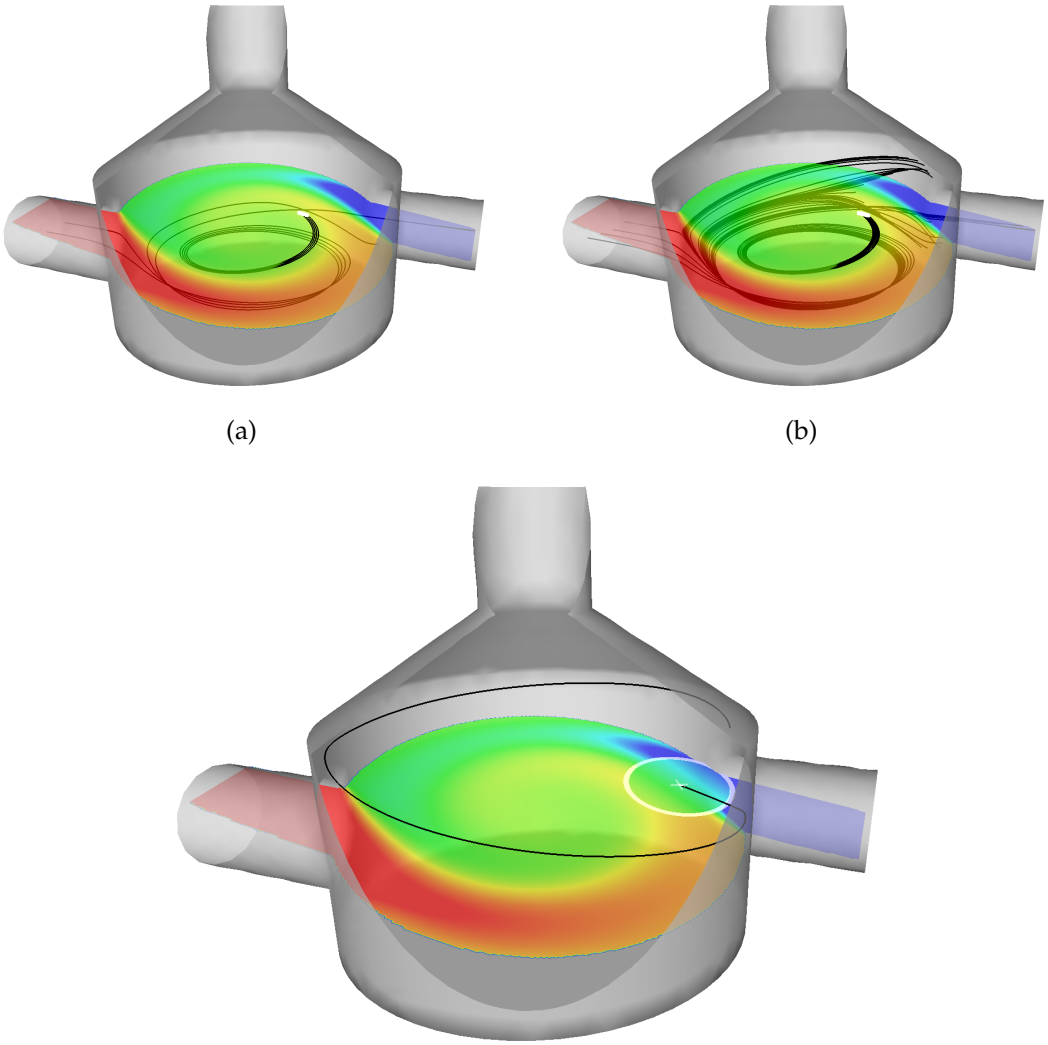


Figure 6.19 — Application of the adaptive zoom lens for the analysis of FTLE ridges in the Hotroom2 dataset. Images (a)–(c) show a sequence of motion steps with direct input. Images (d)–(i) show a sequence of motion steps with adaptive input. The same simulated mouse input and starting position was used for both results. The adaptive method allows a more precise movement of the zoom lens and the pathline. This allows one to navigate between the finely folded ridges with a smoother change of the adaptive magnification of the lens.

pathline seeded at the analyzed position, the interpretation of ridge lines in the FTLE field is substantially eased. As a comparison of Figures 6.19(a)–(c) and Figures 6.19(d)–(i) shows, it is advantageous to combine the adaptive zoom lens with adaptive mouse input.

6.4.2.3 Three-Dimensional Visualization

The result for the 3D flow in the mixer dataset is shown in Figure 6.20. The same behavior as in the 2D case can be observed. With direct mouse input (Figure 6.20(a)), flow features can be missed because the motion of the pathline end point is not uniform. With the predictability-based input adaptation (Figure 6.20(b)), uniform movement of the end point is achieved, which can reveal additional structures in the flow. Due to the 3D data, the adaptive zoom lens operates here on the exploration plane (Figure 6.20(c)).



(a) (b)

(c)

Figure 6.20 — Example of 3D flow in the mixer dataset. Images (a) and (b) show the temporal sequence of a moving pathline (black for better contrast, seeds marked with white squares) controlled with the same simulated mouse input and starting from the same initial position. Temperature is mapped onto the exploration plane (see Section 6.4.1.2). The pathline in (a) was moved with direct mouse input, the pathline in (b) with adaptive mouse input. Direct input leads to uniform movement of the pathline seed, which has the effect that the pathline does not capture all features of the 3D flow. With the adaptive method, the motion of the pathline end point is uniform leading to a better exploration of the flow. Image (c) shows the application of the adaptive zoom lens to 3D flow.

CHAPTER 7

Conclusion

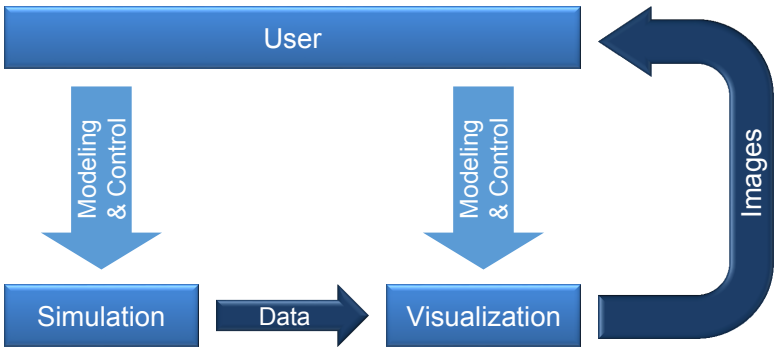
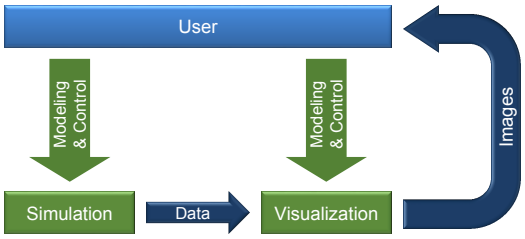


Figure 7.1 — Simulation and analysis process.

This thesis presented different visualization approaches and techniques for integrated simulation systems. They were classified according to a basic model describing the work with simulations (Figure 7.1). The different visualization approaches were developed for different stages or aspects of the simulation and analysis process: modeling and control of simulation and visualization components, visualization of different types of data resulting from simulations, and uncertainty occurring at different stages in the simulation and analysis process. A summary of the concepts, an assessment of their utility, and directions for future work building on them are given in the following.

7.1 Summary of Chapters



Modeling and Control A central aspect of the work with simulation is the modeling and control phase (Part I, marked green). Complex simulations are often designed in a modular way and require us to combine and setup different software modules and components. The same holds for the visualization of the simulation results. In many cases, different techniques must be combined to generate a comprehensive visualization of the data. Module workflows are a common concept for handling such modular software framework (Chapter 2).

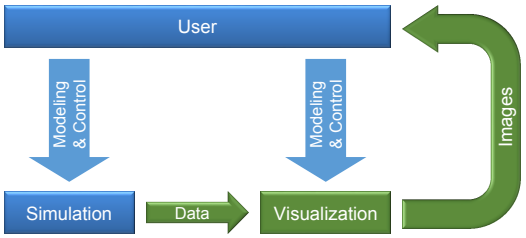
Two approaches that support the work with module workflows and help analyze their evolution were presented. The first approach (Section 2.2) uses a combination of visualization techniques and focuses on the temporal aspects of the modules and related events. The proposed branch view provides scalability with respect to the size and dynamic range of the time range covered by branches in the workflow evolution. Additional views enable a detailed exploration of events. The hierarchical representation of modules allows the exploration of workflows with a large number of modules. Although the approach was demonstrated for module workflows in VisTrails, the visualized elements—time spans of modules, occurrence of different events, changes of parameters—are part of workflows in general. Hence, besides loading the datasets, there are no elements in the implementation that were specifically created for VisTrails. Therefore, it can be assumed that the approach can be easily applied to other workflow systems that are used, e.g., to setup simulations.

As discussed in Section 2.3, evolving module workflows can be represented with dynamic graphs. Hence, techniques for visualizing dynamic graphs can also support the work with module workflows. Although the second approach employing visual adjacency lists (Section 2.4) can be applied to graphs with any kind of characteristics, it is especially suitable for sparse dynamic graphs like workflows. Like the underlying concept of adjacency lists, the visualization is space-efficient. It allows for a compact graph representation without the need for a complex layout algorithm. Furthermore, the representation of links requires only one spatial dimension. This allows a flexible usage of spatial position and extent, which enables the creation of visualizations for dynamic graphs similar to Gantt charts. The Gantt layout helps detect temporal clusters and outliers, and shows the evolution of connectivity and weights.

The case study (Section 2.5) demonstrated how both techniques complement each other. While the first approach focuses on modules and related events, the second approach visualizes the changes of connections between modules. The insights gained through these representations are useful for different application scenarios: from identifying potential improvements for the underlying system to helping in the retrospective analysis of previous application sessions.

However, there are still open problems that should be pursued in the future. Both approaches were designed to analyze individual sessions, but it would also be interesting to consider multiple sessions simultaneously. For example, the visualization could be extended to allow a visual comparison of different sessions. Furthermore, the visualization methods can be enhanced in an application-specific way. Currently, the semantics behind modules and events are visualized only at a very basic level. For example, the first method does not directly convey which aspects of the data are affected by a parameter change and what the effect on the result is. The second method only displays the existence of connections between modules but it is not shown what kind of data is transferred between them. Finally, it is difficult to follow paths in a visualization based on visual adjacency lists. Augmenting the link list with arrows, or integrating appropriate interaction techniques could facilitate such tasks.

Data Visualization The primary purpose of visualization in the context of simulations is to enable the visual analysis of data generated by simulations (marked green). Therefore, the largest part of this thesis was dedicated to this topic (Part II). The presented visualization approaches cover common classes of data generated by simulations in many different applications: scalar, vector, and tensor data. They were ordered according to increasing data dimensionality in this thesis.



There are many applications that require one to generate plots of a set of scalars. Scale-stack bar charts, presented in Chapter 3, combine the advantages of linear and logarithmic plots while avoiding most of their drawbacks. They exhibit the advantages of linear plots, in particular, easy comparison of values. Similar to logarithmic plots, they allow displaying large value ranges. Additionally, the technique can be directly used for typical variants of bar charts like stacked bar charts. The example applications show that the approach works especially well when quantitatively analyzing data with large value ranges. There is a recent publication by others [47] that builds on the basic concept behind scale-stack bar charts, showing the relevance of the topic. Future work could include the

application of the underlying concept of two-dimensional value representation to other chart types or concepts in visualization.

Vector fields are an important type of data used in many applications. For instance, computational fluid dynamics is a traditional research field employing simulations already for a long time. Visualization research related to vector fields is therefore an important and established research topic. Especially the visualization of time-dependent vector fields is an active field of research. Three visualization methods in this thesis cover this topic (Chapter 4): flow radar glyphs, pathline glyphs, and hierarchical line integration.

The strength of the presented glyphs is their ability to visualize unsteady flow in a static way. While flow radar glyphs (Section 4.3) represent vector fields in an Eulerian way, pathline glyphs (Section 4.4) provide a Lagrangian view on unsteady flow. The design of flow radar glyphs preserves angular periodicity and directional symmetries. In many cases, it is easier, faster, and more insightful to analyze time-dependent data with flow radar glyphs than with animated arrow glyphs. Even in cases where the animated approach might perform better, an implementation of flow radar glyphs can be used because they converge against arrow glyphs at the limit of infinitesimally small time ranges. Pathline glyphs were designed as a means of visualizing unsteady flow using pathlines with reduced visual clutter. The results show that they provide insight into additional aspects of the flow compared to the direct visualization of pathlines. The presented interaction techniques facilitate the detailed exploration in small areas without losing the context.

The examples in this thesis show that both glyphs allow a multi-scale visualization of flow data and support the analysis of time-dependent processes and phenomena. The flow domain can be densely covered, which allows for visual segmentation on the overview level. Zooming-in enables then a detailed analysis of flow properties. A further advantage of both glyphs is their suitability for parallelization, as there exists no dependency between individual glyphs. Because flow radar glyphs cover only single points in space, it is in this case even trivial to sub-divide and distribute the underlying data, e.g., for cluster computation and visualization on large high-resolution displays. For future work, the 3D variant of both glyphs could be improved, e.g., via shadows or other shading models that help improve depth perception. Furthermore, flow radar glyphs could be used to represent other directional properties besides flow direction. An example for this can be found in the work by Brix et al. [52]. The concept of pathline glyphs could be applied to other types of integral curves like streaklines. Finally, the glyphs are not restricted to uniform seeding. Seeding density might be locally varied to better cover the flow structure or clustering methods could be applied to the glyphs.

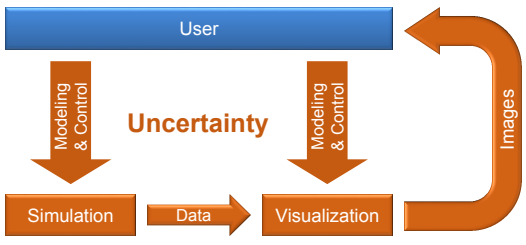
There are several visualization methods for vector fields that require the computation of a large number of densely seeded integral curves, e.g., LIC or FTLE-based visualizations. This is usually time-consuming, especially for 3D time-dependent vector fields, but hierarchical line integration (Section 4.5) reduces the computational complexity from linear to logarithmic in this case. This acceleration scheme is suitable for a large field of applications. It can also accelerate a certain class of computations of quantities along integral curves. Performance measurements of the implementation confirm the logarithmic computational complexity of the scheme. Since practical implementations are subject to some computational overhead, substantial acceleration can only be expected for medium to long trajectories. In contrast to other acceleration techniques for the computation of integral curves, the method is well suited for modern multi-core or many-core architectures like GPUs. Because every hierarchical computation yields an intermediate result, the user can be provided with a preview at no additional cost. Furthermore, advanced and costly integration methods can be readily integrated in hierarchical computation and affect only the construction of the zeroth level of hierarchy. Therefore, the scheme is particularly efficient for higher-order integration in higher-order data. In summary, hierarchical line integration enables the analysis of datasets based on trajectories in a reduced amount of time compared to straightforward methods, especially in the case of high resolutions together with large integration ranges. Considering the gain of computational speed, the drawbacks—reduced accuracy and increased memory consumption—are acceptable for many applications, e.g., providing quick access to flow visualization via web browsers in mobile scenarios [29]. This view is shared in the work by Agranovsky et al. [16, 17], in which similar approaches based on the interpolation of pathline segments are used. The reason for reduced accuracy is the interpolation of the coordinates when constructing longer trajectory segments from shorter ones.

In future work, tensor-product linear interpolation could be replaced by higher-order schemes with better reconstruction quality. Higher-order reconstruction of the coordinate and quantity maps may lead to an integration scheme with better error order. Unfortunately, it is not possible to use adaptive refinement directly with hierarchical line integration, but a hybrid approach seems promising, in which the hierarchical method is used as a first step before adaptive refinement with a direct integration method is applied. The reduced computation time would allow the computation of intermediate results at higher resolutions that can then be adaptively refined. For example, the metric by Matvienko and Krüger [189] could be used to estimate the visual quality of the computed LIC image and control this process. Future research could also investigate applications beyond LIC and the FTLE. For example, stream surfaces, flow level-

sets, or the Mz-criterion are expected to benefit from hierarchical computation, and further applications are conceivable. Finally, accelerating the integration in a subarea of the dataset, e.g., for region-of-interest methods or out-of-core computations, is another possible target for future work.

The visualization of tensor fields is a non-trivial task due to the high dimensionality of the data. Typically, the visualization employs some kind of projection or reduction of the data, e.g., by using topological approaches. In Chapter 5, a method for the visualization of separatrices of coherent regions (SCR), a generalized analog of Lagrangian coherent structures, was presented. As an application of this method, SCR were visualized for symmetric second order tensor fields: synthetic datasets, measured diffusion tensor images, and simulated stress tensor fields. Regarding DT-MRI data, the obtained results were compared to those of classical methods such as fractional anisotropy as well as derived gradient and ridge measures. It was exemplified that SCR tend to form a superset of the features visible with the methods derived from fractional anisotropy. Furthermore, the finite separation ratio (FSR) serves to structure areas of local linear anisotropy by means of a scalable semi-global analysis, a possibility that the other methods do not expose. Concerning measured datasets, features in the FSR appear to be of histological relevance. The results for the stress tensor field prove that the method is not restricted to the analysis of diffusion tensor fields. Ridges in the FSR field can indicate separating principal stress organization and thus support the analysis of stress tensor fields. Since eigenvector lines are used, the visualization is only suitable for areas of predominant linear anisotropy because these trajectories are likely to behave chaotically in regions with planar anisotropy or spherical isotropy. An equivalent analysis in areas of insufficiently linear anisotropy could be part of future work. The application of the method to other tensor fields, e.g., occurring in image processing or surface modeling, also appears promising. Lastly, it is also of interest to extend the method to asymmetric tensors or other generalized interrelations.

Uncertainty Considering uncertainty is essential for enabling an accurate interpretation of simulation results (Part III). Uncertainty cannot only exist in the simulation model but it can also appear at later steps in the simulation and analysis process (marked orange). Different methods for uncertainty in three important stages of this process were presented in Chapter 6: uncertainty in the simulation model, in the simulation result, and in the user interaction.



Section 6.2 focused on uncertainty in the simulation model, more precisely, unknown simulation parameters. The presented approach is strongly related to computational steering and allows for interactive exploration of the parameter space. While the basis for this is the model reduction in the simulation phase, which allows obtaining results for new parameters without rerunning the full simulation, the visualization part is also important. The visualization needs to evaluate the field of polynomials resulting from the model reduction before a result can be displayed. To achieve interactive frame rates, both steps were implemented on the GPU, which also avoids additional memory transfers. As a result, the presented system allows for a comfortable exploration of the parameter space. The user can interactively change the parameters with sliders and gets direct visual feedback. As part of future work, a more advanced processing of the polynomial data could be developed, e.g., sensitivity analysis with respect to the simulation parameters. Furthermore, the scalability to larger datasets could be improved with culling or level-of-detail methods, e.g., the polynomials could be evaluated only for visible parts of the data.

A method for visualizing data with uncertainty was presented in Section 6.3. The flow radar glyphs from Section 4.3 were extended to account for uncertainty in time-dependent vector fields. This extension to data without unique directions is straightforward and intuitive. With the presented scenario and results for a stochastic 2D groundwater simulation, it was demonstrated that the glyphs show not only the distribution of uncertainty, but also directional information and its temporal behavior. With respect to future work, the possible improvements for flow radar glyphs discussed above for data without uncertainty could also increase their utility when visualizing vector fields with uncertainty.

Finally, uncertainty in user interaction was considered in Section 6.4. Two approaches were presented for handling uncertainty in the feedback loop of interactive visualization: predictability-based adaptation of mouse motion for input uncertainty and predictability-based zooming for output uncertainty. Both methods can accelerate the convergence to the aimed visualization result, especially in combination. By adapting the user input to the predictability in the explored area of the data, the user is naturally guided and does not have to change between interaction styles when exploring areas of low predictability. The risk to miss important features is substantially lowered and the overall exploration is made more efficient. The approach was demonstrated for the interactive seeding of pathlines, a common tool in flow visualization. The results show that the analysis of important features in flow fields is actively supported. A further example showed that the application of the method to 3D data is straightforward. With the predictability-based zoom lens, there is no

need to manually adapt the zooming level and view, which can lead to context loss or missing important features. Future work could include the application of the adaptation concept to other explorative tools. With haptic devices, it would be possible to provide the user with force feedback, which may further improve exploration of flow fields. In 3D applications, perception of pathline motion is influenced by the view. Therefore, it would be interesting to adapt the viewing parameters to improve the perception of pathline motion.

7.2 Overarching Discussion

This thesis presented different visualization concepts that aim at improving the work with integrated simulation systems. It was shown that, even though the primary function of visualization lies in enabling the analysis of the simulation results, other parts of the work with simulation can be supported with visualization. Since integrated simulation systems are typically designed in a modular way, they offer a broad and complex functionality. Visualization can support the usage of such systems. By visualizing the evolution of the module workflows created by them, the users can recapitulate their previous work more easily and continue it or reuse parts of it. Since the users are then able to build more complex simulations, suitable tools for analyzing the generated data are required. The presented visualization methods can make the analysis process more efficient because, amongst others, the resulting images are generated faster with hierarchical line integration, or the analysis and comparison is easier due to static visualization of time-dependent processes. With an increasing complexity of simulations, more sources of uncertainty occur. Handling the uncertainty in a suitable way allows not only a more accurate interpretation of simulation results, e.g., because instead of showing the average flow direction the full directional range is shown. It also makes the work with simulations more efficient when the parameter space can be explored in an interactive way or when the uncertainty in interaction is compensated. In summary, it becomes apparent that visualization is crucial for an effective usage of integrated simulation systems.

However, it must be admitted that the presented concepts and methods each target only isolated steps of the whole simulation process. An interesting and important next step would therefore be the development of visualization methods that incorporate a broader part of the process. An example would be the visualization of uncertainty accumulated over the full simulation and visualization pipeline. Another possibility would be to develop methods that show the evolution of the data through the processing stages of the simulation and subsequent visualization. This would help understand the relation between simulation model and data, and the relation between data and resulting images from visualization.

Besides these aspects related to simulations, some general conclusions and suggestions for visualization research can be derived from this thesis. First, apparently simple approaches can be of interest from two different perspectives. On the one hand, even very common and rather simple techniques like bar charts have room for improvements. On the other hand, rather simple approaches can be in some cases very effective, as exemplified by the pathline

glyphs. Their simplicity has the advantage that it is easy to understand and apply them. Next, some of the presented methods employ multiple scales to represent data. They allow one to explore the data according to the concept "overview first, details on demand" [251], which often enables an efficient work with visualization. Finally, visualization research should be open for inspiration from other fields or disciplines. There seems to be still a gap between so-called information visualization and scientific visualization. However, both communities can benefit from each other. For instance, radial information visualization techniques had inspired the development of flow radar glyphs for flow visualization. However, this is not restricted to different groups inside the visualization community. Visual representation and communication is also used in arts, design, and the media. Their concepts can also inspire novel visualization methods.

Author's Work

[1] M. Hlawatsch, M. Burch, F. Beck, J. Freire, C. Silva, and D. Weiskopf. Visualizing the evolution of module workflows. In *Proceedings of the International Conference on Information Visualisation (IV) 2015*, pages 40–49, 2015.

[2] M. Hlawatsch, M. Burch, and D. Weiskopf. Visual adjacency lists for dynamic graphs. *IEEE Transactions on Visualization and Computer Graphics*, 20(11):1590–1603, 2014.

[3] M. Hlawatsch, F. Sadlo, M. Burch, and D. Weiskopf. Scale-stack bar charts. *Computer Graphics Forum*, 32(3):181–190, 2013.

[4] M. Hlawatsch, P. Leube, W. Nowak, and D. Weiskopf. Flow radar glyphs—static visualization of unsteady flow with uncertainty. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1949–1958, 2011.

[5] M. Hlawatsch, F. Sadlo, H. Jang, and D. Weiskopf. Pathline glyphs. *Computer Graphics Forum*, 33(2):497–506, 2014.

[6] M. Hlawatsch, F. Sadlo, and D. Weiskopf. Hierarchical line integration. *IEEE Transactions on Visualization and Computer Graphics*, 17(8):1148 –1163, 2011.

[7] 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):781–794, 2011.

[8] M. Hlawatsch, S. Oladyshkin, and D. Weiskopf. Employing model reduction for uncertainty visualization in the context of CO2 storage simulation. Presentation at the workshop on visualization for decision making under uncertainty (part of IEEE VIS 2015): `http://vda.univie.ac.at/uncertainty2015/submissions/hlawatsch_vdmu.pdf`, 2015,
last access: 2016-01-22.

[9] M. Hlawatsch, F. Sadlo, and D. Weiskopf. Predictability-based adaptive mouse interaction and zooming for visual flow exploration. *International Journal for Uncertainty Quantification*, 3(3):225–240, 2013.

[10] M. Hlawatsch, M. Burch, and D. Weiskopf. Visual analysis of eye movements by hierarchical filter wheels. In *Proceedings of the International Conference on Information Visualisation (IV) 2015*, pages 107–113, 2015.

[11] K. Kurzhals, M. Hlawatsch, F. Heimerl, M. Burch, T. Ertl, and D. Weiskopf. Gaze stripes: Image-based visualization of eye tracking data. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):1005–1014, 2016.

[12] M. Hlawatsch, M. Burch, and D. Weiskopf. Bubble hierarchies. In *Proceedings of the Workshop on Computational Aesthetics (CAe) 2014*, pages 77–80, 2014.

Bibliography

[13] W. Aalst, A. Hofstede, and M. Weske. Business process management: a survey. In W. Aalst and M. Weske, editors, *Business Process Management*, volume 2678 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg, 2003.

[14] Advanced Micro Devices Inc. Quarterly Earnings. `http://ir.amd.com/phoenix.zhtml?c=74093&p=quarterlyearnings`, 2015, last access: 2015-08-21.

[15] A. Agranovsky, C. Garth, and K. Joy. Extracting flow structures using sparse particles. In *Proceedings of Vision, Modeling and Visualization (VMV) 2011*, pages 153–160, 2011.

[16] A. Agranovsky, D. Camp, C. Garth, E. Bethel, K. Joy, and H. Childs. Improved post hoc flow analysis via Lagrangian representations. In *Proceedings of the IEEE Symposium on Large Data Analysis and Visualization (LDAV) 2014*, pages 67–75, 2014.

[17] A. Agranovsky, H. Obermaier, C. Garth, and K. I. Joy. A multi-resolution interpolation scheme for pathline based Lagrangian flow representations. In *Proceedings of the Conference on Visualization and Data Analysis (VDA) 2015, SPIE 9397*, pages 93970K1–93970K12, 2015.

[18] C. Ahlberg and B. Shneiderman. The Alphaslider: a compact and rapid selector. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI) 1994*, pages 365–371, 1994.

[19] W. Aigner, S. Miksch, B. Thurnher, and S. Biffl. PlanningLines: novel glyphs for representing temporal uncertainties and their evaluation. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis) 2005*, pages 457–463, 2005.

[20] W. Aigner, S. Miksch, H. Schumann, and C. Tominski. *Visualization of Time-Oriented Data*. Springer, 2011.

[21] R. S. Allendes Osorio and K. W. Brodlie. Uncertain flow visualization using LIC. In *Proceedings of EG UK Theory and Practice of Computer Graphics (TPCG) 2009*, pages 1–9, 2009.

[22] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock. Kepler: an extensible system for design and execution of scientific workflows. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM) 2004*, pages 423–424, 2004.

[23] S. Ameli, Y. Desai, and S. Shadden. Development of an efficient and flexible pipeline for lagrangian coherent structure computation. In P.-T. Bremer, I. Hotz, V. Pascucci, and R. Peikert, editors, *Topological Methods in Data Analysis and Visualization III*, Mathematics and Visualization, pages 201–215. Springer International Publishing, 2014.

[24] M. Ament, S. Frey, F. Sadlo, T. Ertl, and D. Weiskopf. GPU-based two-dimensional flow simulation steering using coherent structures. In *Proceedings of the International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering (PARENG)*, 2011, paper 18.

[25] M. Anand, S. Bowers, and B. Ludascher. Provenance browser: displaying and querying scientific workflow provenance graphs. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE) 2010*, pages 1201–1204, 2010.

[26] C. Appert and J.-D. Fekete. Orthozoom scroller: 1D multi-scale navigation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI) 2006*, pages 21–30, 2006.

[27] Apple Inc. Earning releases. `http://investor.apple.com/results.cfm`, 2015, last access: 2015-08-21.

[28] D. Archambault, H. Purchase, and B. Pinaud. Animation, small multiples, and the effect of mental map preservation in dynamic graphs. *IEEE Transactions on Visualization and Computer Graphics*, 17(4):539–552, 2011.

[29] M. Aristizábal, J. Congote, A. Segura, A. Moreno, H. Arregui, and O. E. Ruiz. Visualization of flow fields in the web platform. *Journal of WSCG*, 20(3):189–196, 2012.

[30] M. Ashraf, S. Oladyshkin, and W. Nowak. Geological storage of CO2: application, feasibility and efficiency of global sensitivity analysis and risk assessment using the arbitrary polynomial chaos. *International Journal of Greenhouse Gas Control*, 19:704–719, 2013.

[31] T. Athawale and A. Entezari. Uncertainty quantification in linear interpolation for isosurface extraction. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2723–2732, 2013.

[32] J. Bae and B. Watson. Developing and evaluating quilts for the depiction of large layered graphs. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2268–2275, 2011.

[33] P. Bak, F. Mansmann, H. Janetzko, and D. Keim. Spatiotemporal analysis of sensor logs using growth ring maps. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):913–920, 2009.

[34] S. Barakat and X. Tricoche. Adaptive refinement of the flow map using sparse samples. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2753–2762, 2013.

[35] S. Barakat, C. Garth, and X. Tricoche. Interactive computation and rendering of finite-time lyapunov exponent fields. *IEEE Transactions on Visualization and Computer Graphics*, 18(8):1368–1380, 2012.

[36] A. Barker and J. Van Hemert. Scientific workflow: a survey and research directions. In *Proceedings of the International Conference on Parallel Processing and Applied Mathematics (PPAM) 2008*, pages 746–753, 2008.

[37] P. Basser and C. Pierpaoli. Microstructural and physiological features of tissues elucidated by quantitative-diffusion-tensor MRI. *Journal of Magnetic Resonance*, 111(3):209–219, 1996.

[38] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, S. Kuttanikkad, M. Ohlberger, and O. Sander. The distributed and unified numerics environment (DUNE). In *Proceedings of the Symposium on Simulation Technique*, 2006.

[39] S. Bateman, R. Mandryk, C. Gutwin, A. Genest, D. McDine, and C. Brooks. Useful junk? The effects of visual embellishment on comprehension and memorability of charts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI) 2010*, pages 2573–2582, 2010.

[40] J. Bear. *Dynamics of Fluids in Porous Media*. Elsevier, 1972.

[41] F. Beck, M. Burch, and S. Diehl. Towards an aesthetic dimensions framework for dynamic graph visualisations. In *Proceedings of the International Conference on Information Visualisation (IV) 2009*, pages 592–597, 2009.

[42] F. Beck, M. Burch, S. Diehl, and D. Weiskopf. The state of the art in visualizing dynamic graphs. In *EuroVis - STARs*, pages 83–103, 2014.

[43] C. Bennett, J. Ryall, L. Spalteholz, and A. Gooch. The aesthetics of graph visualization. In *Proceedings of the Conference on Computational Aesthetics in Graphics, Visualization and Imaging (CAe) 2007*, pages 57–64, 2007.

[44] W. Berger, H. Piringer, P. Filzmoser, and E. Gröller. Uncertainty-aware exploration of continuous parameter spaces using multivariate prediction. *Computer Graphics Forum*, 30(3):911–920, 2011.

[45] F. H. Bertrand and P. A. Tanguy. Graphical representation of two-dimensional fluid flow by stream vectors. *Communications in Applied Numerical Methods*, 4(2):213–217, 1988.

[46] R. Borgo, J. Kehrer, D. H. Chung, E. Maguire, R. S. Laramee, H. Hauser, M. Ward, and M. Chen. Glyph-based visualization: foundations, design guidelines, techniques and applications. In *Eurographics State of the Art Reports*, pages 39–63, 2013.

[47] R. Borgo, J. Dearden, and M. Jones. Order of magnitude markers: an empirical study on large magnitude number detection. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2261–2270, 2014.

[48] E. Boring and A. Pang. Directional flow visualization of vector fields. In *Proceedings of IEEE Visualization 1996*, pages 389–392, 1996.

[49] R. Botchen, D. Weiskopf, and T. Ertl. Texture-based visualization of uncertainty in flow fields. In *Proceedings of IEEE Visualization 2005*, pages 647–654, 2005.

[50] U. Brandes and S. R. Corman. Visual unrolling of network evolution and the analysis of dynamic discourse? *Information Visualization*, 2(1):40–50, 2003.

[51] R. Brecheisen, A. Vilanova, B. Platel, and B. ter Haar Romeny. Parameter sensitivity visualization for DTI fiber tracking. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1441–1448, 2009.

[52] T. Brix, F. Lindemann, J.-S. Praßni, S. Diepenbrock, and K. H. Hinrichs. Visual analysis of polarization domains in barium titanate during phase transitions. In *Proceedings of the International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG) 2014*, pages 9–18, 2014.

[53] K. Brodlie, R. Allendes Osorio, and A. Lopes. A review of uncertainty in data visualization. In J. Dill, R. Earnshaw, D. Kasik, J. Vince, and P. C. Wong, editors, *Expanding the Frontiers of Visual Analytics and Visualization*, pages 81–109. Springer London, 2012.

[54] A. Brun, H. Knutsson, H.-J. Park, M. E. Shenton, and C.-F. Westin. Clustering fiber traces using normalized cuts. In *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI) 2004*, pages 368–375, 2004.

[55] S. L. Brunton and C. W. Rowley. Fast computation of finite-time lyapunov exponent fields for unsteady flows. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 20(1):017503, 2010.

[56] S. Bryson and C. Levit. The virtual wind tunnel. *IEEE Computer Graphics and Applications*, 12:25–34, 1992.

[57] M. Burch and S. Diehl. TimeRadarTrees: visualizing dynamic compound digraphs. *Computer Graphics Forum*, 27(3):823–830, 2008.

[58] M. Burch and D. Weiskopf. On the benefits and drawbacks of radial diagrams. In W. Huang, editor, *Handbook of Human Centric Visualization*, pages 429–451. Springer New York, 2014.

[59] M. Burch, F. Beck, and S. Diehl. Timeline Trees: visualizing sequences of transactions in information hierarchies. In *Proceedings of the working conference on Advanced Visual Interfaces (AVI) 2008*, pages 75–82, 2008.

[60] M. Burch, C. Vehlow, F. Beck, S. Diehl, and D. Weiskopf. Parallel edge splatting for scalable dynamic graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2344–2353, 2011.

[61] K. Bürger, P. Kondratieva, J. Krüger, and R. Westermann. Importance-driven particle techniques for flow visualization. In *Proceedings of IEEE Pacific Visualization Symposium (PacificVis) 2008*, pages 71–78, 2008.

[62] P. J. Burt. Fast filter transform for image processing. *Computer Graphics and Image Processing*, 16(1):20–51, 1981.

[63] B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In *Proceedings of SIGGRAPH 1993*, pages 263–270, 1993.

[64] S. Callahan, J. Freire, E. Santos, C. Scheidegger, C. Silva, and H. T. Vo. Managing the evolution of dataflows with VisTrails. In *Proceedings of the International Conference on Data Engineering Workshops (ICDEW) 2006*, 2006.

[65] S. K. Card, J. D. Mackinlay, and B. Shneiderman, editors. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers, 1999.

[66] J. V. Carlis and J. A. Konstan. Interactive visualization of serial periodic data. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST) 1998*, pages 29–38, 1998.

[67] C. R. Carvalho and M. D. McMillan. Graphic representation in managerial decision making: the effect of scale break on the dependent axis. Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, 1992.

[68] P. Caserta and O. Zendra. Visualization of the static aspects of software: a survey. *IEEE Transactions on Visualization and Computer Graphics*, 17(7): 913–933, 2010.

[69] C. Chen. Citespace II: detecting and visualizing emerging trends and transient patterns in scientific literature. *Journal of the American Society for Information Science and Technology*, 57(3):359–377, 2006.

[70] C.-M. Chen and H.-W. Shen. Graph-based seed scheduling for out-of-core ftle and pathline computation. In *Proceedings of IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV) 2013*, pages 15–23, 2013.

[71] L. Chittaro and C. Combi. Visualizing queries on databases of temporal histories: new metaphors and their evaluation. *Data & Knowledge Engineering*, 44(2):239–264, 2003.

[72] D. H. Chung, P. A. Legg, M. L. Parry, R. Bown, I. W. Griffiths, R. S. Laramee, and M. Chen. Glyph sorting: interactive visualization for multi-dimensional data. *Information Visualization*, 14(1):76–90, 2015.

[73] CIA. The World Factbook. `https://www.cia.gov/library/publications/the-world-factbook/index.html`, 2015, last access: 2015-08-21.

[74] W. Cleveland and R. McGill. Graphical perception: the visual decoding of quantitative information on graphical displays of data. *Journal of the Royal Statistical Society. Series A (General)*, 150(3):192–229, 1987.

[75] W. S. Cleveland and R. McGill. An experiment in graphical perception. *International Journal of Man-Machine Studies*, 25(5):491–500, 1986.

[76] A. Cockburn, J. Savage, and A. Wallace. Tuning and testing scrolling interfaces that automatically zoom. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI) 2005*, pages 71–80, 2005.

[77] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2009.

[78] S. B. Cousins and M. G. Kahn. The visual display of temporal information. *Artificial Intelligence in Medicine*, 3(6):341–357, 1991.

[79] B. Csébfalvi. An evaluation of prefiltered reconstruction schemes for volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 14(2):289–301, 2008.

[80] S. da Cruz, M. Campos, and M. Mattoso. Towards a taxonomy of provenance in scientific workflow management systems. In *Proceedings of the World Conference on Services 2009 - I*, pages 259–266, 2009.

[81] S. B. Davidson and J. Freire. Provenance and scientific workflows: challenges and opportunities. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD) 2008*, pages 1345–1350, 2008.

[82] W. C. de Leeuw and J. J. van Wijk. A probe for local flow field visualization. In *Proceedings of IEEE Visualization 1993*, pages 39–45, 1993.

[83] T. Delmarcelle and L. Hesselink. Visualizing second-order tensor fields with hyperstreamlines. *IEEE Computer Graphics and Applications*, 13(4): 25–33, 1993.

[84] T. Delmarcelle and L. Hesselink. The topology of symmetric, second-order tensor fields. In *Proceedings of IEEE Visualization 1994*, pages 140–147, 1994.

[85] G. di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[86] C. Dick, J. Georgii, R. Burgkart, and R. Westermann. Computational steering for patient-specific implant planning in orthopedics. In *Proceedings of the Eurographics Workshop on Visual Computing for Biomedicine (VCBM) 2008*, pages 83–92, 2008.

[87] 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):1399–1406, 2009.

[88] S. Diehl. *Software Visualization–Visualizing the Structure, Behaviour, and Evolution of Software*. Springer, 2007.

[89] S. Diehl and C. Görg. Graphs, they are changing. In *Proceedings of Graph Drawing 2002*, pages 23–31, 2002.

[90] S. Diehl, F. Beck, and M. Burch. Uncovering strengths and weaknesses of radial visualizations—an empirical approach. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):935–942, 2010.

[91] Z. Ding, J. C. Gore, and A. W. Anderson. Classification and quantification of neuronal fiber pathways using diffusion tensor MRI. *Magnetic Resonance in Medicine*, 49(4):716–721, 2003.

[92] T. Dombre, U. Frisch, J. M. Greene, M. Hénon, A. Mehr, and A. M. Soward. Chaotic streamlines in the ABC flows. *Journal of Fluid Mechanics*, 167:353–391, 1986.

[93] D. Dovey. Vector plots for irregular grids. In *Proceedings of IEEE Visualization 1995*, pages 248–253, 1995.

[94] G. Draper, Y. Livnat, and R. Riesenfeld. A survey of radial methods for information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(5):759–776, 2009.

[95] D. Eberly. *Ridges in Image and Data Analysis*. Kluwer Academic Publishers, 1996.

[96] W. Ehlers and A. Wagner. Multi-component modelling of human brain tissue: a contribution to the constitutive and computational description of deformation, flow and diffusion processes with application to the invasive drug-delivery problem. *Computer Methods in Biomechanics and Biomedical Engineering*, 18(8):861–879, 2015.

[97] K. Engel, M. Hadwiger, J. M. Kniss, C. Rezk-Salama, and D. Weiskopf. *Real-time Volume Graphics*. A. K. Peters, Ltd., 2006.

[98] C. Erten, P. J. Harding, S. G. Kobourov, K. Wampler, and G. V. Yee. Graphael: graph animations with evolving layouts. In *Proceedings of Graph Drawing 2003*, pages 98–110, 2003.

[99] M. Falk, A. Seizinger, M. Üffinger, F. Sadlo, and D. Weiskopf. Trajectory-augmented visualization of Lagrangian coherent structures in unsteady flow. In *Proceedings of International Symposium on Flow Visualization (ISFV)*, 2010.

[100] G. Farin. *Curves and Surfaces for CAGD: A Practical Guide*. Morgan Kaufmann Publishers, 2002.

[101] J.-D. Fekete. The InfoVis Toolkit. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis) 2004*, pages 167–174, 2004.

[102] S. Few. Save the pies for dessert. In *Perceptual Edge: Visual Business Intelligence Newsletter*, pages 1–14, 2007.

[103] S. Few. Our irresistible fascination with all things circular. In *Perceptual Edge: Visual Business Intelligence Newsletter*, pages 1–9, 2010.

[104] M. Üffinger, F. Sadlo, M. Kirby, C. Hansen, and T. Ertl. FTLE computation beyond first-order approximation. In *Proceedings of Eurographics 2012 - Short Papers*, pages 61–64, 2012.

[105] M. Üffinger, F. Sadlo, and T. Ertl. A time-dependent vector field topology based on streak surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 19(3):379–392, 2013.

[106] L. R. Frank. Characterization of anisotropy in high angular resolution diffusion-weighted MRI. *Magnetic Resonance in Medicine*, 47(6):1083–1099, 2002.

[107] Y. Frishman and A. Tal. Online dynamic graph drawing. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):727–740, 2008.

[108] R. Fuchs, R. Peikert, F. Sadlo, B. Alsallakh, and M. E. Gröller. Delocalized unsteady vortex region detectors. In *Proceedings of Vision, Modeling, and Visualization (VMV) 2008*, pages 81–90, 2008.

[109] R. Fuchs, B. Schindler, and R. Peikert. Scale-space approaches to FTLE ridges. In R. Peikert, H. Hauser, H. Carr, and R. Fuchs, editors, *Topological Methods in Data Analysis and Visualization II*, Mathematics and Visualization, pages 283–296. Springer Berlin Heidelberg, 2012.

[110] J. D. Furst and S. M. Pizer. Marching ridges. In *Proceedings of the IASTED International Conference on Signal and Image Processing (SIP) 2001*, pages 22–26, 2001.

[111] H. L. Gantt. *Work, Wages, and Profits*. The Engineering Magazine Co., 1913.

[112] C. Garth, F. Gerhardt, X. Tricoche, and H. Hagen. Efficient computation and visualization of coherent structures in fluid flow applications. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1464–1471, 2007.

[113] C. Garth, G.-S. Li, X. Tricoche, C. Hansen, and H. Hagen. Visualization of coherent structures in transient 2D flows. In H.-C. Hege, K. Polthier, and G. Scheuermann, editors, *Topology-Based Methods in Visualization II*, Mathematics and Visualization, pages 1–13. Springer Berlin Heidelberg, 2009.

[114] M. Ghoniem, J. D. Fekete, and P. Castagliola. A comparison of the readability of graphs using node-link and matrix-based representations. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis) 2004*, pages 17–24, 2004.

[115] S. H. Gilbert, A. P. Bensona, P. Li, and A. V. Holden. Regional localisation of left ventricular sheet structure: integration with current models of cardiac fibre, sheet and band structure. *European Journal of Cardio-Thoracic Surgery*, 32(2):231–249, 2007.

[116] T. Günther, A. Kuhn, B. Kutz, and H. Theisel. Mass-dependent integral curves in unsteady vector fields. *Computer Graphics Forum*, 32(3pt2):211–220, 2013.

[117] J. Goecks, A. Nekrutenko, and J. Taylor. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8):1–13, 2010.

[118] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

[119] J. H. Goldberg and J. Helfman. Eye tracking for visualization evaluation: reading values on linear versus radial graphs. *Information Visualization*, 10(3):182–195, 2011.

[120] M. Greilich, M. Burch, and S. Diehl. Visualizing the evolution of compound digraphs with TimeArcTrees. *Computer Graphics Forum*, 28(3): 975–982, 2009.

[121] S. Gumhold. Splatting illuminated ellipsoids with depth correction. In *Proceedings of Vision, Modeling, and Visualization (VMV) 2003*, pages 245–252, 2003.

[122] T. Günther, C. Rössl, and H. Theisel. Opacity optimization for 3D line fields. *ACM Transactions on Graphics*, pages 120:1–120:8, 2013.

[123] H. Guo, X. Yuan, J. Huang, and X. Zhu. Coupled ensemble flow line advection and analysis. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2733–2742, 2013.

[124] H. Guo, J. Zhang, R. Liu, L. Liu, X. Yuan, J. Huang, X. Meng, and J. Pan. Advection-based sparse data management for visualizing unsteady flow. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2555–2564, 2014.

[125] R. B. Haber and D. A. McNabb. Visualization idioms: a conceptual model for scientific visualization systems. In G. M. Nielson, B. Shriver, and L. Rosenblum, editors, *Visualization in Scientific Computing*, pages 74–93. IEEE Computer Society Press, 1990.

[126] D. Hackenberg, G. Juckeland, and H. Brunst. High resolution program flow visualization of hardware accelerated hybrid multi-core applications. In *Proceedings of the IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid) 2010*, pages 786–791, 2010.

[127] S. Hadlak, H. Schumann, C. H. Cap, and T. Wollenberg. Supporting the visual analysis of dynamic networks by clustering associated temporal attributes. *IEEE Transactions on Visualization and Computer Graphics*, 19(12): 2267–2276, 2013.

[128] G. Haller. Distinguished material surfaces and coherent structures in three-dimensional fluid flows. *Physica D: Nonlinear Phenomena*, 149(4): 248–277, 2001.

[129] G. Haller. Lagrangian coherent structures from approximate velocity data. *Physics of Fluids*, 14(6):1851–1861, 2002.

[130] G. Haller. An objective definition of a vortex. *Journal of Fluid Mechanics*, 525:1–26, 2005.

[131] G. Haller and G. Yuan. Lagrangian coherent structures and mixing in two-dimensional turbulence. *Physica D*, 147(3-4):352–370, 2000.

[132] C. Hansen and C. Johnson, editors. *The Visualization Handbook*. Elsevier, 2005.

[133] P. S. Heckbert. Filtering by repeated integration. *Computer Graphics*, 20(4): 315–321, 1986.

[134] J. Heer, S. K. Card, and J. Landay. Prefuse: a toolkit for interactive information visualization. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems (CHI) 2005*, pages 421–430, 2005.

[135] H.-C. Hege and D. Stalling. Fast LIC with piecewise polynomial filter kernels. In H.-C. Hege and K. Polthier, editors, *Mathematical Visualization – Algorithms and Applications*, pages 295–314. Springer, 1998.

[136] W. Heidrich, R. Westermann, H.-P. Seidel, and T. Ertl. Applications of pixel textures in visualization and realistic image synthesis. In *Proceedings of the ACM Symposium on Interactive 3D Graphics (i3D) 1999*, pages 127–134, 1999.

[137] J. Helman and L. Hesselink. Representation and display of vector field topology in fluid flow data sets. *IEEE Computer*, 22(8):27–36, 1989.

[138] J. Helman and L. Hesselink. Visualizing vector field topology in fluid flows. *Computer Graphics and Applications, IEEE*, 11(3):36–46, 1991.

[139] N. Henry. *Exploring Social Networks with Matrix-based Representations*. PhD thesis, Université Paris Sud, France, and University of Sydney, Australia, 2008.

[140] M. Hlawatsch. *Lagrangian Analysis of Symmetric Second Order Tensor Fields*. Diplomarbeit, Universität Stuttgart, 2008.

[141] M. Hlawitschka, G. Scheuermann, A. Anwander, T. Knösche, M. Tittgemeyer, and B. Hamann. Tensor lines in tensor fields of arbitrary order. In *Proceedings of the International Symposium on Visual Computing (ISVC) 2007*, pages 341–350, 2007.

[142] Y. Hu, S. Kobourov, and S. Veeramoni. Embedding, clustering and coloring for dynamic maps. In *Proceedings of the IEEE Pacific Visualization Symposium (PacificVis) 2012*, pages 33–40, 2012.

[143] D. Huff. *How to Lie With Statistics*. W. W. Norton & Company, 1993.

[144] S. Huling and B. Pivetz. In-Situ Chemical Oxidation. Technical report, EPA 600-R-06-072. US Environmental Protection Agency (USEPA), 2006.

[145] M. Hummel, H. Obermaier, C. Garth, and K. Joy. Comparative visual analysis of Lagrangian transport in CFD ensembles. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2743–2752, 2013.

[146] T. Igarashi and K. Hinckley. Speed-dependent automatic zooming for browsing large documents. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST) 2000*, pages 139–148, 2000.

[147] B. Jobard, G. Erlebacher, and M. Y. Hussaini. Hardware-accelerated texture advection for unsteady flow visualization. In *Proceedings of IEEE Visualization 2000*, pages 155–162, 2000.

[148] C. Johnson. Top scientific visualization research problems. *IEEE Computer Graphics and Applications*, 24(4):13–17, 2004.

[149] D. K. Jones, editor. *Diffusion MRI : Theory, Methods, and Applications*. Oxford University Press, 2011.

[150] A. Joshi and P. Rheingans. Illustration-inspired techniques for visualizing time-varying data. In *Proceedings of IEEE Visualization 2005*, pages 679–686, 2005.

[151] G. Karch, F. Sadlo, D. Weiskopf, and T. Ertl. Visualization of 2D unsteady flow using streamline-based concepts in space-time. *Journal of Visualization*, pages 1–14, 2015.

[152] J. Kasten, C. Petz, I. Hotz, B. Noack, and H.-C. Hege. Localized finite-time Lyapunov exponent for unsteady flow analysis. In *Proceedings of Vision, Modeling, and Visualization (VMV) 2009*, pages 265–274, 2009.

[153] M. Kaufmann and D. Wagner, editors. *Drawing Graphs: Methods and Models*. Springer, 2001.

[154] G. Kindlmann. Superquadric tensor glyphs. In *Proceedings of the Eurographics Symposium on Visualization (VisSym/EuroVis) 2004*, pages 147–154, 2004.

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

[156] G. Kindlmann, X. Tricoche, and C.-F. Westin. Anisotropy creases delineate white matter structure in diffusion tensor MRI. In *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI) 2006*, pages 126–133, 2006.

[157] G. Kindlmann, D. B. Ennis, R. T. Whitaker, and C.-F. Westin. Diffusion tensor analysis with invariant gradients and rotation tangents. *IEEE Transactions on Medical Imaging*, 26(11):1483–1499, 2007.

[158] G. Kindlmann, X. Tricoche, and C.-F. Westin. Delineating white matter structure in diffusion tensor MRI with anisotropy creases. *Medical Image Analysis*, 11(5):492–502, 2007.

[159] R. M. Kirby, H. Marmanis, and D. H. Laidlaw. Visualizing multivalued data from 2D incompressible flows using concepts from painting. In *Proceedings of IEEE Visualization 1999*, pages 333–340, 1999.

[160] R. V. Klassen and S. J. Harrington. Shadowed hedgehogs: A technique for visualizing 2D slices of 3D vector fields. In *Proceedings of IEEE Visualization 1991*, pages 148–153, 1991.

[161] P. Kondratieva, J. Krüger, and R. Westermann. The application of GPU particle tracing to diffusion tensor field visualization. In *Proceedings of IEEE Visualization 2005*, pages 73–78, 2005.

[162] E. M. Kornaropoulos and I. G. Tollis. Dagview: an approach for visualizing large graphs. In *Proceedings of Graph Drawing 2013*, pages 499–510, 2013.

[163] R. Kosara and S. Miksch. Metaphors of movement: a visualization and user interface for time-oriented, skeletal plans. *Artificial Intelligence in Medicine*, 22(2):111–131, 2001.

[164] M. Kraus and M. Strengert. Pyramid filters based on bilinear interpolation. In *Proceedings of the International Conference on Computer Graphics Theory and Applications (GRAPP) 2007*, pages 21–28, 2007.

[165] P. Kristensson, N. Dahlback, D. Anundi, M. Bjornstad, H. Gillberg, J. Haraldsson, I. Martensson, M. Nordvall, and J. Stahl. An evaluation of space time cube representation of spatiotemporal patterns. *IEEE Transactions on Visualization and Computer Graphics*, 15(4):696–702, 2009.

[166] D. Kroese, T. Taimre, and Z. Botev. *Handbook of Monte Carlo Methods*. Wiley Series in Probability and Statistics. Wiley, 2013.

[167] A. Kuhn, C. Rössl, T. Weinkauf, and H. Theisel. A benchmark for evaluating FTLE computations. In *Proceedings of the IEEE Pacific Visualization Symposium (PacificVis) 2012*, pages 121–128, 2012.

[168] A. Kuhn, W. Engelke, C. Rössl, M. Hadwiger, and H. Theisel. Time line cell tracking for the approximation of Lagrangian coherent structures with subgrid accuracy. *Computer Graphics Forum*, 33(1):222–234, 2014.

[169] G. Kumar and M. Garland. Visual exploration of complex time-varying graphs. *IEEE Transactions on Visualization and Computer Graphics*, 12(5): 805–812, 2006.

[170] D. Laidlaw and A. Vilanova, editors. *New Developments in the Visualization and Processing of Tensor Fields*. Mathematics and Visualization. Springer, 2012.

[171] D. H. Laidlaw, E. T. Ahrens, D. Kremers, M. J. Avalos, R. E. Jacobs, and C. Readhead. Visualizing diffusion tensor images of the mouse spinal cord. In *Proceedings of IEEE Visualization 1998*, pages 127–134, 1998.

[172] M. Lanza. The Evolution Matrix: recovering software evolution using software visualization techniques. In *Proceedings of the International Workshop on Principles of Software Evolution (IWPSE) 2001*, pages 37–42, 2001.

[173] R. S. Laramee. Interactive 3D flow visualization using a streamrunner. In *Extended Abstracts of the SIGCHI Conference on Human Factors in Computing Systems (CHI) 2002*, pages 804–805, 2002.

[174] R. S. Laramee. *Interactive 3D Flow Visualization Based on Textures and Geometric Primitives*. PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2004.

[175] R. S. Laramee, H. Hauser, H. Doleisch, B. Vrolijk, F. H. Post, and D. Weiskopf. The state of the art in flow visualization: dense and texture-based techniques. *Computer Graphics Forum*, 23(2):203–221, 2004.

[176] R. S. Laramee, G. Erlebacher, C. Garth, T. Schafhitzel, H. Theisel, X. Tricoche, T. Weinkauf, and D. Weiskopf. Applications of texture-based flow visualization. *Engineering Applications of Computational Fluid Mechanics*, 2 (3):264–274, 2008.

[177] D. Le Bihan. Looking into the functional architecture of the brain with diffusion MRI. In *Proceedings of the International Symposium on Functional and Molecular Imaging of Stroke and Dementia 2006*, pages 1–24, 2006.

[178] B. Lee, C. Plaisant, C. S. Parr, J.-D. Fekete, and N. Henry. Task taxonomy for graph visualization. In *Proceedings of the AVI workshop on BEyond time and errors: novel evaluation methods for information visualization (BELIV) 2006*, pages 1–5, 2006.

[179] Y.-Y. Lee, C.-C. Lin, and H.-C. Yen. Mental map preserving graph drawing using simulated annealing. In *Proceedings of the Asia-Pacific Symposium on Information Visualisation (APVis) 2006*, pages 179–188, 2006.

[180] F. Lekien, S. C. Shadden, and J. E. Marsden. Lagrangian coherent structures in n-dimensional systems. *Journal of Mathematical Physics*, 48(6): 065404, 2007.

[181] L. Li and H.-W. Shen. View-dependent multi-resolutional flow texture advection. In *Proceedings of the Conference on Visualization and Data Analysis (VDA) 2006, SPIE 6060*, pages 24–34, 2006.

[182] I. Liiv. Seriation and matrix reordering methods: an historical overview. *Statistical Analysis and Data Mining*, 3(2):70–91, 2010.

[183] S. K. Lodha, A. Pang, R. E. Sheehan, and C. M. Wittenbrink. UFLOW: Visualizing uncertainty in fluid flow. In *Proceedings of IEEE Visualization 1996*, pages 249–254, 1996.

[184] H. Löffelmann, L. Mroz, and E. Gröller. Hierarchical streamarrows for the visualization of dynamical systems. In *Proceedings of the Eurographics Workshop on Visualization in Scientific Computing (EG ViSC/EuroVis) 1997*, pages 203–211, 1997.

[185] E. Lum, B. Wilson, and K.-L. Ma. High-quality lighting and efficient pre-integration for volume rendering. In *Proceedings of the Eurographics Symposium on Visualization (VisSym/EuroVis) 2004*, pages 25–34, 2004.

[186] P. Macko and M. Seltzer. Provenance map orbiter: interactive exploration of large provenance graphs. In *Proceedings of the Workshop on the Theory and Practice of Provenance (TaPP) 2011*, pages 292–301, 2011.

[187] G. T. Mase and G. E. Mase. *Continuum Mechanics for Engineers, Second Edition*. CRC Press, 1999.

[188] T. Masui, K. Kashiwagi, and G. R. Borden. Elastic graphical interfaces to precise data manipulation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI) 1995*, pages 143–144, 1995.

[189] V. Matvienko and J. Krüger. A metric for the evaluation of dense vector field visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 19(7):1122–1132, 2013.

[190] T. McLoughlin, R. S. Laramee, R. Peikert, F. H. Post, and M. Chen. Over two decades of integration-based, geometric flow visualization. *Computer Graphics Forum*, 29(6):1807–1829, 2010.

[191] T. McLoughlin, M. Jones, R. Laramee, R. Malki, I. Masters, and C. Hansen. Similarity measures for enhancing interactive streamline seeding. *IEEE Transactions on Visualization and Computer Graphics*, 19(8):1342–1353, 2013.

[192] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages and Computing*, 6(2):183–210, 1995.

[193] S. Mori and P. C. M. van Zijl. Fiber tracking: principles and strategies - a technical review. *NMR in Biomedicine*, 15:468–480, 2002.

[194] C. Müller, G. Reina, M. Burch, and D. Weiskopf. Subversion statistics sifter. In *Proceedings of the International Symposium on Visual Computing (ISVC) 2010*, pages 447–457, 2010.

[195] J. Nese. Quantifying local predictability in phase space. *Physica D: Nonlinear Phenomena*, 35(1-2):237–250, 1989.

[196] R. Netzel and D. Weiskopf. Texture-based flow visualization. *Computing in Science & Engineering*, 15(6):96–102, 2013.

[197] S. C. North. Incremental layout in DynaDAG. In *Proceedings of Graph Drawing 1995*, pages 409–418, 1995.

[198] B. Nouanesengsy, T.-Y. Lee, K. Lu, H.-W. Shen, and T. Peterka. Parallel particle advection and FTLE computation for time-varying flow fields. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC) 2012*, pages 1–11, 2012.

[199] H. Obermaier and K. Joy. Function field analysis for the visualization of flow similarity in time-varying vector fields. In G. Bebis, R. Boyle, B. Parvin, D. Koracin, C. Fowlkes, S. Wang, M.-H. Choi, S. Mantler, J. Schulze, D. Acevedo, K. Mueller, and M. Papka, editors, *Advances in Visual Computing*, volume 7432 of *Lecture Notes in Computer Science*, pages 253–264. Springer Berlin Heidelberg, 2012.

[200] L. O'Donnell, W. E. L. Grimson, and C.-F. Westin. Interface detection in diffusion tensor MRI. In *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI) 2004*, pages 360–367, 2004.

[201] M. Ogawa and K.-L. Ma. StarGate: a unified, interactive visualization of software projects. In *Proceedings of the IEEE Pacific Visualization Symposium (PacificVis) 2008*, pages 191–198, 2008.

[202] M. Ogawa and K. L. Ma. code_swarm: a design study in organic software visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15 (6):1097–1104, 2009.

[203] J. Ogden, E. Adelson, J. Bergen, and P. Burt. Pyramid-based computer graphics. *RCA Engineer*, 30(5):4–15, 1985.

[204] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[205] S. Oladyshkin, H. Class, R. Hofmann, and W. Nowak. Highly efficient tool for probabilistic risk assessment of CCS joint with injection design.

In *Proceedings of the International Conference on Computational Methods in Water Resources (CMWR) 2010*, 2010.

[206] M. Otto and H. Theisel. Vortex analysis in uncertain vector fields. *Computer Graphics Forum*, 31(3pt2):1035–1044, 2012.

[207] M. Otto, T. Germer, H.-C. Hege, and H. Theisel. Uncertain 2D vector field topology. *Computer Graphics Forum*, 29(2):347–356, 2010.

[208] M. Otto, T. Germer, and H. Theisel. Uncertain topology of 3D vector fields. In *Proceedings of the IEEE Pacific Visualization Symposium (PacificVis) 2011*, pages 67–74, 2011.

[209] M. Otto, T. Germer, and H. Theisel. Closed stream lines in uncertain vector fields. In *Proceedings of the Spring Conference on Computer Graphics (SCCG) 2011*, pages 87–94, 2013.

[210] A. T. Pang, C. M. Wittenbrink, and S. K. Lodha. Approaches to uncertainty visualization. *The Visual Computer*, 13(8):370–390, 1997.

[211] T. Peeters, A. Vilanova, G. Strijkers, and B. ter Haar Romeny. Visualization of the fibrous structure of the heart. In *Proceedings of Vision, Modeling, and Visualization (VMV) 2006*, pages 309–316, 2006.

[212] Z. Peng, E. Grundy, R. Laramee, G. Chen, and N. Croft. Mesh-driven vector field clustering and visualization: an image-based approach. *IEEE Transactions on Visualization and Computer Graphics*, 18(2):283–298, 2012.

[213] C. Petz, K. Pöthkow, and H.-C. Hege. Probabilistic local features in uncertain vector fields with spatial correlation. *Computer Graphics Forum*, 31(3pt2):1045–1054, 2012.

[214] T. Pfaffelmoser, M. Reitinger, and R. Westermann. Visualizing the positional and geometrical variability of isosurfaces in uncertain scalar fields. *Computer Graphics Forum*, 30(3):951–960, 2011.

[215] R. Pickett and G. Grinstein. Iconographic displays for visualizing multidimensional data. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC) 1988*, pages 514–519, 1988.

[216] C. Pierpaoli and P. J. Basser. Toward a quantitative assessment of diffusion anisotropy. *Magnetic Resonance in Medicine*, 36(6):893–906, 1996.

[217] D. Pilar and C. Ware. Representing flow patterns by using streamlines with glyphs. *IEEE Transactions on Visualization and Computer Graphics*, 19 (8):1331–1341, 2013.

[218] M. Pinzger, H. Gall, M. Fischer, and M. Lanza. Visualizing multiple evolution metrics. In *Proceedings of the ACM Symposium on Software Visualization (SoftVis) 2005*, pages 67–75, 2005.

[219] C. Plaisant, B. Milash, A. Rose, S. Widoff, and B. Shneiderman. LifeLines: visualizing personal histories. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI) 1996*, pages 221–227, 1996.

[220] A. Pobitzer, R. Peikert, R. Fuchs, B. Schindler, A. Kuhn, H. Theisel, K. Matkovic, and H. Hauser. The state of the art in topology-based visualization of unsteady flow. *Computer Graphics Forum*, 30(6):1789–1811, 2011.

[221] F. H. Post, B. Vrolijk, H. Hauser, R. S. Laramee, and H. Doleisch. The state of the art in flow visualisation: feature extraction and tracking. *Computer Graphics Forum*, 22(4):775–792, 2003.

[222] K. Pöthkow, B. Weber, and H.-C. Hege. Probabilistic marching cubes. *Computer Graphics Forum*, 30(3):931–940, 2011.

[223] K. Potter, P. Rosen, and C. Johnson. From quantification to visualization: a taxonomy of uncertainty visualization approaches. In A. Dienstfrey and R. Boisvert, editors, *Uncertainty Quantification in Scientific Computing*, volume 377 of *IFIP Advances in Information and Communication Technology*, pages 226–249. Springer Berlin Heidelberg, 2012.

[224] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 2007.

[225] K. Pöthkow and H.-C. Hege. Nonparametric models for uncertainty visualization. *Computer Graphics Forum*, 32(3pt2):131–140, 2013.

[226] H. C. Purchase. Metrics for graph drawing aesthetics. *Journal of Visual Languages and Computing*, 13(5):501–516, 2002.

[227] N. Rio and P. Silva. Probe-it! Visualization support for provenance. In G. Bebis, R. Boyle, B. Parvin, D. Koracin, N. Paragios, S.-M. Tanveer, T. Ju, Z. Liu, S. Coquillart, C. Cruz-Neira, T. Müller, and T. Malzbender, editors, *Advances in Visual Computing*, volume 4842 of *Lecture Notes in Computer Science*, pages 732–741. Springer Berlin Heidelberg, 2007.

[228] F. Ritter, T. Boskamp, A. Homeyer, H. Laue, M. Schwier, F. Link, and H.-O. Peitgen. Medical image analysis. *IEEE Pulse*, 2(6):60–70, 2011.

[229] S. K. Robinson. Coherent motions in the turbulent boundary layer. *Annual Review of Fluid Mechanics*, 23:601–639, 1991.

[230] D. Rohmer, A. Sitek, and G. T. Gullberg. Reconstruction and visualization of fiber and sheet structure with regularized tensor diffusion MRI in the human heart. Technical report, Lawrence Berkeley National Laboratory, LBNL-60277, 2006.

[231] R. Rosenholtz, Y. Li, J. Mansfield, and Z. Jin. Feature congestion: a measure of display clutter. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI) 2005*, pages 761–770, 2005.

[232] Y. Rubin. *Applied Stochastic Hydrogeology*. Oxford University Press, 2003.

[233] S. Rufiange and M. J. McGuffin. Diffani: visualizing dynamic graphs with a hybrid of difference maps and animation. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2556–2565, 2013.

[234] F. Sadlo and R. Peikert. Efficient visualization of Lagrangian coherent structures by filtered AMR ridge extraction. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1456–1463, 2007.

[235] F. Sadlo and R. Peikert. Visualizing Lagrangian coherent structures and comparison to vector field topology. In H.-C. Hege, K. Polthier, and G. Scheuermann, editors, *Topology-Based Methods in Visualization II*, Mathematics and Visualization, pages 15–29. Springer Berlin Heidelberg, 2009.

[236] F. Sadlo and D. Weiskopf. Time-dependent 2-D vector field topology: an approach inspired by Lagrangian coherent structures. *Computer Graphics Forum*, 29(1):88–100, 2010.

[237] F. Sadlo, R. Peikert, and M. Sick. Visualization tools for vorticity transport analysis in incompressible flow. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):949–956, 2006.

[238] F. Sadlo, S. Bachthaler, C. Dachsbacher, and D. Weiskopf. Space-time flow visualization of dynamics in 2D Lagrangian coherent structures. In G. Csurka, M. Kraus, R. Laramee, P. Richard, and J. Braz, editors, *Computer Vision, Imaging and Computer Graphics. Theory and Application*, volume 359 of *Communications in Computer and Information Science*, pages 145–159. Springer Berlin Heidelberg, 2013.

[239] A. Sallaberry, C. Muelder, and K.-L. Ma. Clustering, visualizing, and navigating for large dynamic graphs. In *Proceedings of Graph Drawing 2013*, pages 487–498, 2013.

[240] T. Salzbrunn, C. Garth, G. Scheuermann, and J. Meyer. Pathline predicates and unsteady flow structures. *Visual Computer*, 24(12):1039–1051, 2008.

[241] T. Salzbrunn, H. Jänicke, T. Wischgoll, and G. Scheuermann. The state of the art in flow visualization: partition-based techniques. In *Proceedings of Simulation and Visualization (SimVis) 2008*, pages 75–92, 2008.

[242] J. Sanyal, S. Zhang, G. Bhattacharya, P. Amburn, and R. Moorhead. A user study to compare four uncertainty visualization methods for 1D and 2D datasets. *IEEE Transactions on Visualization and Computer Graphics*, 15(6): 1209–1218, 2009.

[243] J. Sanyal, S. Zhang, J. Dyer, A. Mercer, P. Amburn, and R. Moorhead. Noodles: a tool for visualization of numerical weather model ensemble uncertainty. *IEEE Transactions on Visualization and Computer Graphics*, 16 (6):1421–1430, 2010.

[244] S. E. Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.

[245] M. Schenke and W. Ehlers. On the analysis of porous media dynamics using a DUNE-PANDAS interface. In A. Dedner, B. Flemisch, and R. Klöfkorn, editors, *Advances in DUNE*, pages 157–167. Springer Berlin Heidelberg, 2012.

[246] G. Scheuermann and X. Tricoche. Topological methods for flow visualization. In C. Hansen and C. Johnson, editors, *The Visualization Handbook*, pages 341–356. Elsevier, 2005.

[247] S. Schlegel, N. Korn, and G. Scheuermann. On the interpolation of data with normally distributed uncertainty for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2305–2314, 2012.

[248] W. Schroeder, K. M. Martin, and W. E. Lorensen. *The Visualization Toolkit (2nd ed.): An Object-Oriented Approach to 3D Graphics*. Prentice-Hall, Inc., 1998.

[249] T. Schultz and G. Kindlmann. A maximum enhancing higher-order tensor glyph. *Computer Graphics Forum*, 29(3):1143–1152, 2010.

[250] T. Schultz and H.-P. Seidel. Estimating crossing fibers: a tensor decomposition approach. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1635–1642, 2008.

[251] B. Shneiderman. The eyes have it: a task by data type taxonomy for information visualizations. In *Proceedings of the IEEE Symposium on Visual Languages 1996*, pages 336–343, 1996.

[252] C. Sigg and M. Hadwiger. Fast third-order texture filtering. In M. Pharr and R. Fernando, editors, *GPU Gems 2*, pages 313–329. Addison-Wesley, 2005.

[253] C. Silva, J. Freire, and S. Callahan. Provenance for visualizations: reproducibility and beyond. *Computing in Science and Engineering*, 9(5):82–89, 2007.

[254] C. Silva, E. Anderson, E. Santos, and J. Freire. Using VisTrails and provenance for teaching scientific visualization. *Computer Graphics Forum*, 30(1):75–84, 2011.

[255] D. Sosnovik, R. Wang, G. Dai, T. Reese, and V. Wedeen. Diffusion MR tractography of the heart. *Journal of Cardiovascular Magnetic Resonance*, 11 (1):47, 2009.

[256] D. Stalling and H.-C. Hege. Fast and resolution independent line integral convolution. In *Proceedings of SIGGRAPH 1995*, pages 249–256, 1995.

[257] Stuttgart Research Centre for Simulation Technology. SimTech Website. `http://www.simtech.uni-stuttgart.de/`, 2015, last access: 2015-08-21.

[258] Y. Su. It's easy to produce chartjunk using Microsoft Excel 2007 but hard to make good graphs. *Computational Statistics & Data Analysis*, 52(10): 4594–4601, 2008.

[259] J. Talbot, S. Lin, and P. Hanrahan. An extension of Wilkinson's algorithm for positioning tick labels on axes. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1036–1043, 2010.

[260] J. Tao, J. Ma, C. Wang, and C.-K. Shene. A unified approach to streamline selection and viewpoint selection for 3D flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 19(3):393–406, 2013.

[261] C. Tominski, P. Schulze-Wollgast, and H. Schumann. 3D information visualization for time dependent data on maps. In *Proceedings of the International Conference on Information Visualisation (IV) 2005*, pages 175–181, 2005.

[262] J.-D. Tournier, F. Calamante, D. G. Gadian, and A. Connelly. Direct estimation of the fiber orientation density function from diffusion-weighted MRI data using spherical deconvolution. *NeuroImage*, 23(3):1176–1185, 2004.

[263] X. Tricoche, G. Kindlmann, and C.-F. Westin. Invariant crease lines for topological and structural analysis of tensor fields. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1627–1634, 2008.

[264] J. Trümper, J. Bohnet, and J. Döllner. Understanding complex multithreaded software systems by using trace visualization. In *Proceedings of the ACM Symposium on Software Visualization (SoftVis) 2010*, pages 133–142, 2010.

[265] D. S. Tuch. Q-ball imaging. *Magnetic Resonance in Medicine*, 52(6):1358–1372, 2004.

[266] E. R. Tufte. *Envisioning Information*. Graphics Press, 1990.

[267] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 1992.

[268] E. R. Tufte. *Beautiful Evidence*. Graphics Press, 2006.

[269] B. Tversky, J. B. Morrison, and M. Bétrancourt. Animation: can it facilitate? *International Journal on Human-Computer Studies*, 57(4):247–262, 2002.

[270] United Nations Scientific Committee on the Effects of Atomic Radiation. UNSCEAR 2008 Report. Sources and effects of ionizing radiation. Volume 2. Annex D—Health effects due to radiation from the Chernobyl accident. `http://www.unscear.org/docs/reports/2008/11-80076_Report_2008_Annex_D.pdf`, 2008, last access: 2015-08-21.

[271] M. Unser, A. Aldroubi, and M. Eden. B-spline signal processing: Part I–Theory. *IEEE Transactions on Signal Processing*, 41(2):821–833, 1993.

[272] M. Unser, A. Aldroubi, and M. Eden. B-spline signal processing: Part II–Efficient design and applications. *IEEE Transactions on Signal Processing*, 41(2):834–848, 1993.

[273] C. Upson, J. Faulhaber, T.A., D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. van Dam. The application visualization system: a computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, 9(4):30–42, 1989.

[274] R. van Liere, J. D. Mulder, and J. J. van Wijk. Computational steering. In H. M. Liddell, A. Colbrook, L. O. Hertzberger, and P. M. A. Sloot, editors, *HPCN Europe*, volume 1067 of *Lecture Notes in Computer Science*, pages 696–702. Springer, 1996.

[275] R. van Pelt, J. Olivan Bescos, M. Breeuwer, R. Clough, M. Gröller, B. ter Haar Romeny, and A. Vilanova. Interactive virtual probing of 4D MRI blood-flow. *IEEE Transactions on Visualization and Computer Graphics*, 17 (12):2153–2162, 2011.

[276] T. Van Walsum, F. Post, D. Silver, and F. Post. Feature extraction and iconic visualization. *IEEE Transactions on Visualization and Computer Graphics*, 2 (2):111–119, 1996.

[277] J. van Wijk and W. Nuij. A model for smooth viewing and navigation of large 2D information spaces. *IEEE Transactions on Visualization and Computer Graphics*, 10(4):447–458, 2004.

[278] J. J. van Wijk. Implicit stream surfaces. In *Proceedings of IEEE Visualization 1993*, pages 245–252, 1993.

[279] J. J. van Wijk. Image based flow visualization. *ACM Transactions on Graphics*, 21(3):745–754, 2002.

[280] C. Vehlow, F. Beck, P. Auwärter, and D. Weiskopf. Visualizing the evolution of communities in dynamic graphs. *Computer Graphics Forum*, 34(1): 277–288, 2015.

[281] F. B. Viégas, M. Wattenberg, and K. Dave. Studying cooperation and conflict between authors with history flow visualizations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI) 2004*, pages 575–582, 2004.

[282] A. Vilanova, G. Berenschot, and C. van de Pul. DTI visualization with streamsurfaces and evenly-spaced volume seeding. In *Proceedings of the Eurographics Symposium on Visualization (VisSym/EuroVis) 2004*, pages 173–182, 2004.

[283] A. Vilanova, S. Zhang, G. Kindlmann, and D. Laidlaw. An introduction to visualization of diffusion tensor imaging and its applications. In J. Weickert and H. Hagen, editors, *Visualization and Image Processing of Tensor Fields*, pages 121–153. Springer, 2006.

[284] L. Voinea and A. Telea. CVSgrab: mining the history of large software projects. In *Proceedings of the Eurographics Symposium on Visualization (EuroVis) 2006*, pages 187–194, 2006.

[285] L. Voinea, A. Telea, and J. J. van Wijk. CVSscan: visualization of code evolution. In *Proceedings of the ACM Symposium on Software Visualization (SoftVis) 2005*, pages 47–56, 2005.

[286] A. Wagner. *Extended modelling of the multiphasic human brain tissue with application to drug-infusion processes*. PhD thesis, Universität Stuttgart, 2014.

[287] S. Wakana, H. Jiang, L. M. Nagae-Poetscher, P. C. M. van Zijl, and S. Mori. Fiber tract-based atlas of human white matter anatomy. *Radiology*, 230(1): 77–87, 2004.

[288] T. Wang, C. Plaisant, B. Shneiderman, N. Spring, D. Roseman, G. Marchand, V. Mukherjee, and M. Smith. Temporal summaries: supporting temporal categorical searching, aggregation and comparison. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1049–1056, 2009.

[289] M. O. Ward. A taxonomy of glyph placement strategies for multidimensional data visualization. *Information Visualization*, 1(3-4):194–210, 2002.

[290] C. Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers, 2004.

[291] C. Ware, H. C. Purchase, L. Colpoys, and M. McGill. Cognitive measurements of graph aesthetics. *Information Visualization*, 1(2):103–110, 2002.

[292] J. Waser, R. Fuchs, H. Ribičić, B. Schindler, G. Blöschl, and M. E. Gröller. World lines. *IEEE Transactions on Visualization and Computer Graphics*, 16 (6):1458–1467, 2010.

[293] J. Waser, H. Ribicic, R. Fuchs, C. Hirsch, B. Schindler, G. Bloschl, and M. Gröller. Nodes on ropes: a comprehensive data and control flow for steering ensemble simulations. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1872–1881, 2011.

[294] M. Weber, M. Alexa, and W. Müller. Visualizing time-series on spirals. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis) 2001*, pages 7–13, 2001.

[295] T. Weinkauf and H. Theisel. Flow visualization and analysis using streak and time lines. *Computing in Science & Engineering*, 14(5):78–84, 2012.

[296] T. Weinkauf, H.-C. Hege, and H. Theisel. Advected tangent curves: a general scheme for characteristic curves of flow fields. *Computer Graphics Forum*, 31(2pt4):825–834, 2012.

[297] T. Weinkauf, H. Theisel, and O. Sorkine. Cusps of characteristic curves and intersection-aware visualization of path and streak lines. In R. Peikert, H. Hauser, H. Carr, and R. Fuchs, editors, *Topological Methods in Data Analysis and Visualization II*, Mathematics and Visualization, pages 161–175. Springer, 2012.

[298] D. M. Weinstein, G. L. Kindlmann, and E. C. Lundberg. Tensorlines: advection-diffusion based propagation through diffusion tensor fields. In *Proceedings of IEEE Visualization 1999*, pages 249–253, 1999.

[299] D. Weiskopf, M. Hopf, and T. Ertl. Hardware-accelerated visualization of time-varying 2D and 3D vector fields by texture advection via programmable per-pixel operations. In *Proceedings of Vision, Modeling, and Visualization (VMV) 2001*, pages 439–446, 2001.

[300] R. Westermann, C. Johnson, and T. Ertl. A level-set method for flow visualization. In *Proceedings of IEEE Visualization 2000*, pages 147–154, 2000.

[301] C. Westin, A. Vilanova, and B. Burgeth, editors. *Visualization and Processing of Tensors and Higher Order Descriptors for Multi-Valued Data*. Mathematics and Visualization. Springer Berlin Heidelberg, 2014.

[302] C.-F. Westin, S. Peled, H. Gudbjartsson, R. Kikinis, and F. A. Jolesz. Geometrical diffusion measures for MRI from tensor basis analysis. In *Proceedings of the International Society for Magnetic Resonance in Medicine (ISMRM) 1997*, page 1742, 1997.

[303] C.-F. Westin, S. E. Maier, H. Mamata, A. Nabavi, F. A. Jolesz, and R. Kikinis. Processing and visualization of diffusion tensor MRI. *Medical Image Analysis*, 6(2):93–108, 2002.

[304] H. Wickham and H. Hofmann. Product plots. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2223–2230, 2011.

[305] C. Wittenbrink, A. Pang, and S. Lodha. Glyphs for visualizing uncertainty in vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):266–279, 1996.

[306] J. Woodring and H.-W. Shen. Chronovolumes: a direct rendering technique for visualizing time-varying data. In *Proceedings of the Eurographics/IEEE TVCG Workshop on Volume Graphics (VG) 2003*, pages 27–34, 2003.

[307] J. Woodring, C. Wang, and H.-W. Shen. High dimensional direct rendering of time-varying volumetric data. In *Proceedings of IEEE Visualization 2003*, pages 417–424, 2003.

[308] H. Wright, R. Crompton, S. Kharche, and P. Wenisch. Steering and visualization: enabling technologies for computational science. *Future Generation Computer Systems*, 26(3):506–513, 2010.

[309] J. S. Yi, N. Elmqvist, and S. Lee. TimeMatrix: analyzing temporal social networks using interactive matrix-based visualizations. *International Journal of Human Computer Interaction*, 26(11-12):1031–1051, 2010.

[310] H. Yu, C. Wang, C.-K. Shene, and J. Chen. Hierarchical streamline bundles. *IEEE Transactions on Visualization and Computer Graphics*, 18(8):1353–1367, 2012.

[311] F. Zhang and E. R. Hancock. Edge detection and anisotropic diffusion for tensor-valued images. *Proceedings of the IEEE International Conference on Image Processing (ICIP) 2006*, pages 229–232, 2006.

[312] W. Zhang, Y. Wang, J. Zhan, B. Liu, and J. Ning. Parallel streamline placement for 2D flow fields. *IEEE Transactions on Visualization and Computer Graphics*, 19(7):1185–1198, 2013.

[313] J. Zhao, P. Forer, and A. S. Harvey. Activities, ringmaps and geovisualization of large human movement fields. *Information Visualization*, 7:198–209, 2008.

[314] X. Zheng and A. Pang. HyperLIC. In *Proceedings of IEEE Visualization 2003*, pages 249–256, 2003.

[315] X. Zheng and A. Pang. Topological lines in 3D tensor fields. In *Proceedings of IEEE Visualization 2004*, pages 313–320, 2004.

[316] X. Zheng, B. Parlett, and A. Pang. Topological structures of 3D tensor fields. In *Proceedings of IEEE Visualization 2005*, pages 551–558, 2005.

[317] L. Zhukov and A. H. Barr. Heart-muscle fiber reconstruction from diffusion tensor MRI. In *Proceedings of IEEE Visualization 2003*, pages 597–602, 2003.

[318] T. Zuk and S. Carpendale. Theoretical analysis of uncertainty visualizations. In *Proceedings of the Conference on Visualization and Data Analysis (VDA) 2006, SPIE 6060*, pages 66–79, 2006.

[319] T. Zuk, J. Downton, D. Gray, S. Carpendale, and J. Liang. Exploration of uncertainty in bidirectional vector fields. In *Proceedings of the Conference on Visualization and Data Analysis (VDA) 2008, SPIE 6809*, pages 68090B1–68090B10, 2008.