Advanced Visualization Techniques for Flow Simulations: from Higher-Order Polynomial Data to Time-Dependent Topology

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

Vorgelegt von

Markus Üffinger

aus Karlsruhe

Hauptberichter:	Prof. Dr. Thomas Ertl
Mitberichter:	Prof. Dr. Charles D. Hansen

Tag der mündlichen Prüfung: 17. Mai 2013

Visualisierungsinstitut
der Universität Stuttgart

2013

Turbulence is the most important
unsolved problem of classical physics.

Richard Feynman

Acknowledgments

First and foremost, I would like to thank my advisor Thomas Ertl for his support and his guidance, and for providing a great research environment at VIS and VISUS. I am also very thankful to Peter Bastian who acted as co-advisor until he moved to the University of Heidelberg and to Chuck Hansen for becoming referee of my doctoral thesis. Chuck Hansen also enabled me to enjoy a great research stay at the Institute for Scientific Computing and Imaging (SCI). My research was funded by the Germany Research Foundation (DFG) in the context of the cluster of excellence in simulation technology (EXC 310 SimTech).

I would like to thank Thomas Klein and Magnus Strengert for answering all my questions in the early stage of my doctoral studies. For our collaboration and the groundwork on higher-order visualization I want to thank Ralph Botchen. Also in the higher-order context, special thanks go to Steffen Frey, for joining me in the development of highly sophisticated CUDA kernels and parallelization strategies. I am very grateful to my long-term office mate Steffen Müthing and to Alex Schweitzer for the fruitful cooperations and for giving me deeper insight into numerical simulation. Further, I would like to thank my short-term office mates Mike Eissele, Benjamin Höferlin, and Tobias Schafhitzel for the good time we had. I owe many thanks to Filip Sadlo for sharing his extraordinary knowledge about flow visualization, and for his never ending support and endurance during paper deadlines.

I also had the pleasure to collaborate with Thomas Bolemann, Shymaa El-Leithy, Florian Hindenlang, Martin Falk (thanks for the helpful LaTeX templates), Marcel Hlawatsch, Thomas Heidlauf, Mike Kirby, Tom Schilli, Bertram Thomass, and Markus Wolff. I really appreciate the hours spent in proof-reading my thesis by Steffen Frey, Marcel Hlawatsch, Steffen Müthing, Alex Schweitzer, and my father. Moreover, I would like to thank all remaining colleagues—I had a great time at VIS and VISUS.

Above all, my deepest gratitude goes to my family who always supported and encouraged me during this work.

Stuttgart,
July 2013

Markus Üffinger

Contents

List of Abbreviations and Acronyms

1D	one-dimensional
2D	two-dimensional
3D	three-dimensional
API	application programming interface
AMR	adaptive mesh refinement
BVP	boundary value problem
CFD	computational fluid dynamics
CG	continuous Galerkin
CPU	central processing unit
CUDA	Compute Unified Device Architecture
CV	control volume
DG	discontinuous Galerkin
DNS	direct numerical simulation
DVR	direct volume rendering
FD	finite difference
FDM	finite difference method
FEM	finite element method
FFT	fast Fourier transform
fps	frames per second
FTLE	finite-time Lyapunov exponent
FVM	finite volume method
GFEM	generalized FEM
GLSL	OpenGL shading language
GPU	graphics processing unit
GS	geomery shader
HPC	high performance computing
IVP	initial value problem
LE	Lyapunov exponent
LES	large eddy simulation
LIC	line integral convolution
LCS	Lagrangian coherent structures
MPI	Message Passing Interface
MSF	maximum separation factor
ODE	ordinary differential equation
OpenCL	Open Compute Language
OpenGL	Open Graphics Libray
OpenMP	Open Multi-Processing
PDE	partial differential equation
pixel	picture element
PPUM	particle-partition of unity method
PU	partition of unity
PSE	problem solving environment
RAM	random access memory
RANS	Reynolds-averaged Navier-Stokes
RBF	radial basis function
ROI	region of interest
RK4	Runge-Kutta 4
SIMD	single instruction multiple data
SI	international system of units
voxel	volume picture element (cf. pixel)
XFEM	extended FEM

Abstract

Computational fluid dynamics (CFD) has become an important tool for predicting fluid behavior in research and industry. Today, in the era of tera- and petascale computing, the complexity and the size of simulations have reached a state where an extremely large amount of data is generated that has to be stored and analyzed. An indispensable instrument for such analysis is provided by computational flow visualization. It helps in gaining insight and understanding of the flow and its underlying physics, which are subject to a complex spectrum of characteristic behavior, ranging from laminar to turbulent or even chaotic characteristics, all of these taking place on a wide range of length and time scales. The simulation side tries to address and control this vast complexity by developing new sophisticated models and adaptive discretization schemes, resulting in new types of data. Examples of such emerging simulations are generalized finite element methods or hp-adaptive discontinuous Galerkin schemes of high-order. This work addresses the direct visualization of the resulting higher-order field data, avoiding the traditional resampling approach to enable a more accurate visual analysis. The second major contribution of this thesis deals with the inherent complexity of fluid dynamics. New feature-based and topology-based visualization algorithms for unsteady flow are proposed to reduce the vast amounts of raw data to their essential structure.

For the direct visualization pixel-accurate techniques are presented for 2D field data from generalized finite element simulations, which consist of a piecewise polynomial part of high order enriched with problem-dependent ansatz functions. Secondly, a direct volume rendering system for hp-adaptive finite elements, which combine an adaptive grid discretization with piecewise polynomial higher-order approximations, is presented. The parallel GPU implementation runs on single workstations, as well as on clusters, enabling a real-time generation of high quality images, and interactive exploration of the volumetric polynomial solution. Methods for visual debugging of these complex simulations are also important and presented.

Direct flow visualization is complemented by new feature and topology-based methods. A promising approach for analyzing the structure of time-dependent vector fields is provided by finite-time Lyapunov exponent (FTLE) fields. In this work, interactive methods are presented that help in understanding the cause of FTLE structures, and novel approaches to FTLE computation are developed to account for the linearization error made by traditional methods. Building on this, it is investigated under which circumstances FTLE ridges represent Lagrangian coherent structures (LCS)—the time-dependent counterpart to separatrices of traditional "steady" vector field topology. As a major result, a novel time-dependent 3D vector field topology concept based on streak surfaces is proposed. Streak LCS offer a higher quality than corresponding FTLE ridges, and animations of streak LCS can be computed at comparably low cost, alleviating the topological analysis of complex time-dependent fields.

German Abstract
—Zusammenfassung—

Die numerische Strömungsmechanik hat sich sowohl in der Forschung als auch in der Industrie zu einem wichtigen Werkzeug für die Vorhersage von Fluidströmungen entwickelt. Strömungssimulationen sind heutzutage oft hoch komplex und erzeugen riesige Datenmengen, die gespeichert und analysiert werden müssen. Bei der Analyse dieser Datenmengen ist die numerische Strömungsvisualisierung meist unentbehrlich. Visualisierungstechniken helfen zum Beispiel, einen Einblick und ein tieferes Verständnis komplexer Strömungen und ihrer zugrundeliegenden Physik zu gewinnen. Komplexität in Strömungen äußert sich unter anderem durch das große Spektrum an charakteristischen Verhaltensmustern, die auf unterschiedlichsten räumlichen und zeitlichen Skalen auftreten können. Die Strömung kann zum Beispiel laminaren, turbulenten oder gar chaotischen Charakter annehmen. Aufgrund dieser enormen Komplexität ist die Entwicklung besserer Simulationsmodelle sowie effizienterer adaptiver Diskretisierungsansätze ein aktuelles Forschungsthema auf Seiten der Simulation. Beispiele sind Verallgemeinerungen der Finite-Elemente-Methode sowie hp-adaptive diskontinuierliche Galerkin Verfahren höherer Ordnung.

Im ersten Teil dieser Arbeit werden neue Techniken für die direkte Visualisierung der erzeugten Felddaten höherer Ordnung vorgestellt. Diese ermöglichen eine präzisere interaktive Analyse der Felddaten, da der traditionell übliche Resampling-Ansatz vermieden wird. Der zweite Teil behandelt die inhärente Komplexität von Strömungen und deren Dynamik. In diesem Zusammenhang werden merkmals- und topologiebasierte Visualisierungstechniken für zeitabhängige Strömungen vorgestellt, um eine automatische Extraktion der grundlegenden Strömungsstruktur, und somit auch eine starke Datenreduktion, zu ermöglichen.

Für die direkte Visualisierung der Felddaten höherer Ordnung wird eine pixelgenaue Technik für 2D Felder aus verallgemeinerten Finite-Element-Verfahren präsentiert, die sowohl den stückweisen polynomiellen Lösungsansatz der Simulation als auch die dazugehörigen problembezogenen Ansatzfunktionen effizient behandelt. Danach wird ein interaktives Visualisierungssystem für direktes Volumenrendering hp-adaptiver Finite-Elemente vorgestellt, bei welchem eine adaptive Gitterdiskretisierung mit stückweise polynomiellen Näherungen kombiniert wird. Durch das geschickte Ausnutzen der hochparallelen GPU Rechnerarchitektur wird dabei eine Echtzeitdarstellung und somit eine interaktive Exploration der volumetrischen Polynomlösung möglich. Um zusätzlich die Entwicklung der Simulationscodes zu unterstützen, werden neue Ansätze für das visuelle Debugging der Felddaten hoher Ordnung präsentiert.

Die vorgestellten Methoden für die direkte Visualisierung komplexer Felddaten werden durch merkmals- und topologiebasierte Methoden ergänzt. Finite-time Lyapunov exponent (FTLE) Felder bieten dabei einen erfolgversprechenden Ansatz um die Struk-

tur zeitabhängiger Feldern zu analysieren. Zunächst werden interaktive Methoden gezeigt, die bei der Analyse der FTLE-Struktur helfen. Anschließend werden neue Algorithmen für die Berechnung von FTLE Feldern vorgestellt, die den Linearisierungsfehler der klassischen Methoden verringern. Darauf aufbauend wird untersucht, unter welchen Bedingungen Gratlinien und Flächen in FTLE Feldern, sogenannte FTLE-Ridges, Lagrange-kohärente Strukturen (LCS) repräsentieren. LCS gelten als das zeitabhängige Gegenstück zu den Separatrizen der klassischen "stationären" Vektorfeldtopologie. Diese Eigenschaft wird genutzt und als ein Hauptresultat wird ein zeitabhängiges 3D Vektorfeldtopologiekonzept präsentiert, welches auf Streichflächen basiert. Die LCS, die durch Streichflächen gewonnen werden, haben eine höhere Qualität als entsprechende FTLE-Ridges, und mit ihnen ist es weniger aufwändig, Animationen der Feldtopologie zu berechnen. Dadurch kann die Analyse komplexer, zeitabhängiger Vektorfelder deutlich erleichtert werden.

Chapter 1

Introduction

The advancements made in *scientific computing* during the last decades have led to an unprecedented progress in many research and engineering disciplines. Nowadays, numerical computing is accepted as the third paradigm of research, besides the classical approach of explaining natural phenomena with the tools of experimentation and the development of theories. To continue this overwhelming success in the future sophisticated simulation techniques and efficient analysis environments need to be developed, which harness the steadily increasing computational power provided by massively parallel high performance computing hardware. *Scientific visualization* plays an essential role in such complex environments. It helps the scientists to verify their hypotheses, but also assists in the discovery of new structures and correlations in the vast amounts of raw data generated by the simulation, leading to new insight and scientific progress. Interactive visualization of spatial data is one cornerstone of such a visual analysis, often combined with sophisticated feature extraction algorithms to reduce data size and complexity. The traditional role of visualization, solely as a post-processing step, might no longer be valid in the future. A tighter integration of simulation and visualization is desirable, for example, to enable interactive steering applications. In the case of petascale and exascale computing such an integration can even become a necessity, as the large amount of raw field data can no longer be stored, and thus has to be analyzed and reduced in size in-situ with the running simulation, for example, with topological or feature-based methods.

This thesis is the result of a visualization research project embedded within the cluster of excellence EXC 310 simulation technology (SimTech) at the University of Stuttgart. The goal is the development of novel high-quality visualization concepts and techniques that target several challenges in the analysis of data from computational fluid dynamics (CFD). While many types of fluids are accurately described by the Navier-Stokes equations, still today, the structure and dynamics of fluid flow is fully understood only for rather simple configurations. In general settings, effects on different length and time scales result in turbulence, or even chaotic behavior. As a practical consequence, simulation is often not able to resolve all relevant scales numerically and has to employ

approximate modeling and sophisticated discretization techniques. Adaptive *higher-order finite elements* and *generalized finite element simulation* are examples that produce novel types of field data that are given in a piecewise polynomial representation of high order and thus cannot be handled directly by traditional visualization methods based on low-order techniques, like triangle rendering and linear interpolation. The complex structure of this simulation data requires for resampling, resulting in a loss of detail, and in excessive storage and computation overhead. To avoid these limitations novel techniques for the *interactive direct visualization* of higher-order simulation results are presented, which allow for an accurate interactive exploration of the data. Moreover, new methods for visual debugging and quality analysis of such simulations are discussed, a topic that will increase in importance in future compute environments. Being able to visualize the flow data accurately is a prerequisite, but it is often not sufficient for gaining insight and understanding of the complex flow structure. To this end, the second major topic of this thesis focuses on *feature-based flow visualization* techniques. Besides supporting the user with a semi-automated analysis, these techniques allow for a reduction of the raw flow data to a more compact representation composed of the essential flow features. In this context, novel visualization techniques for finite-time Lyapunov exponent (FTLE) fields are presented. This includes new algorithms for improved FTLE computation, which account for linearization errors made by the traditional techniques. In recent years, the FTLE has gained popularity in the flow visualization community because it offers a promising approach for analyzing the global structure of time-dependent vector fields, similar to vector field topology. The classical vector field topology succeeds in revealing the abstract structure of steady flow fields by means of a compact flow skeleton consisting of separatrices, which separate the flow into regions of different behavior. However, for time-varying flow the "steady" topology fails. The third part of this thesis accounts for this shortcoming, addressing the ongoing challenge of developing a *time-dependent vector field topology*. As a main contribution, a novel generalized topology concept for time-dependent 3D flow is proposed, introducing distinguished streak surfaces as the counterpart to separatrices in classical vector field topology.

The presented topics and visualization techniques are highly relevant for future problem solving environments in the context of fluid dynamics—but are not limited to this area of application. The thesis adopts an integrated approach, giving insight into the multidisciplinary undertaking of scientific computing from a visualization viewpoint. A recurring theme is the parallel implementation of the presented algorithms on multi- and many-core architectures, like GPUs and GPU clusters. This enables an interactive exploration of the data produced by current and future compute environments.

1.1 Thesis Overview

Part I - Visualization in Scientific Computing gives an introduction to the principles of scientific computing with a strong focus on scientific visualization, providing the basis for the topics in the main parts of the thesis. After outlining the problem domain, the interactive visualization pipeline and its role in the context of scientific computing is covered in Chapter 2. This also includes a description of heterogeneous compute environments built from cost-effective off-the-shelve hardware with multi-core CPUs and many-core accelerators, like graphics processing units (GPUs). Chapter 3 introduces discretized field representations, which are employed at the core of computational fluid dynamics simulation and visualization—with grid-based field data being of central interest. Important numerical techniques, which help in solving problems on these data structures, are also covered. Chapter 4 discusses fundamental approaches of grid-based simulation techniques, like the finite element method, to provide the basis for the discussion of higher-order simulation data. This is followed by a description of the characteristic behavior of fluids to highlight the challenges in the analysis of flow data. Subgrid-scale simulation models are also discussed shortly, as a starting point for the development of more model-aware visualization algorithms, and the data sets used for evaluating the presented visualization techniques are listed. Finally, an overview of fundamental flow visualization concepts and techniques follows in Chapter 5. This includes interactive direct volume rendering as well as integral curves and surfaces—the basis of many feature-based and topology-based approaches for vector fields.

Part II - Direct Visualization of Higher-Order Fields covers interactive visualization techniques for data from higher-order simulations. An overview of higher-order field data, its generation, and the challenges it poses to visualization, is given in Chapter 6. New visualization techniques specifically tailored for this complex kind of data are required to avoid resampling and enable accurate visual representations. At first, methods for 2D field data are treated (Chapter 7). In this context, a novel technique is presented that enables an interactive direct visualization of data from generalized finite element simulations [199]. Its parallel GPU implementation allows for a pixel-accurate analysis of higher-order polynomial solutions and associated problem dependent ansatz functions. Novel techniques for 3D fields of higher-order are covered in Chapter 8. This includes the first interactive direct volume rendering system for hp-adaptive field solutions of arbitrary polynomial order computed on complex unstructured grids [194]. The GPU and cluster-based ray casting framework not only delivers high quality results at interactive display rates, i.e., it preserves all details contained in the raw simulation data, but also provides tools for analyzing the convergence quality of discontinuous Galerkin type simulations. Finally, results of model-aware flow interpolation are presented, harnessing the law of the wall simulation model to benefit from the improved accuracy, also on the visualization side (Chapter 9) [198].

Part III: Integral Curves for Feature-Based Flow Visualization discusses feature-based visualization techniques that automatically detect and extract field regions that

exhibit special characteristics, often by using derived field quantities. Feature-based flow visualization techniques help in reducing the immense information content of the raw field data, allowing for a more focused analysis (Chapter 10). An introductory example demonstrates a visualization technique that employs distinguished feature surfaces extracted from an associated scalar field for seeding integral lines that are then traced through the vector field [195]. This helps in understanding the interaction of multiple fields, e.g., the development of vortices near shear layers (Chapter 11). The underlying particle tracing is accelerated with a novel GPU-based algorithm for integral line computation (Chapter 12). Time-dependent CFD data are often visualized by means of animations. However, with this approach, clutter and occlusion exacerbate the interpretation of dynamics in 3D fields. The finite-time Lyapunov exponent (FTLE) field provides an alternative. It helps in analyzing the global structure of time-dependent vector fields (Chapter 13). First, an interactive technique is presented that enriches FTLE fields with path lines [47]. Moreover, FTLE accuracy is examined with respect to the linearization errors made by traditional algorithms. This leads to methods for nonlinear FTLE computation, which achieve higher quality results [197]. The computation of FTLE, as a global approach, involves integral lines, and it is used as the basis for the topology-based approaches to flow visualization presented in Part IV.

Part IV: Toward a Time-Dependent Vector Field Topology. Topological approaches aim at providing abstract field representations, which help in understanding the overall structure of the field. They also allow for a drastic reduction in data size. The classical 2D vector field topology structures the flow into regions of different behavior separated by special streamlines—the separatrices (Chapter 14). However, as the generalization to time-dependent fields is not straightforward, the development of time-dependent vector field topology concepts is still an active research topic. Ridges from FTLE fields provide a promising starting point, because, under certain circumstances, they represent LCS, the time-dependent counterpart to separatrices. Firstly, a new approach for validating the LCS property of ridges from 2D FTLE fields is presented (Chapter 15) [159]. Secondly, as a major contribution to time-dependent vector field topology research, a concept for 3D fields based on streak surfaces is proposed and its properties are investigated (Chapter 16) [196]. The streak surface manifolds are obtained by seeding at FTLE ridge intersections. They provide LCS representations of higher quality than the traditional FTLE ridges, and allow to produce animations of complex dynamic field structures at low computational cost.

The summary and **Concluding Remarks** are given in the last chapter. Three important cornerstones of successful visualization and analysis of CFD simulations are identified, and the contributions of the presented flow visualization techniques to scientific visualization and computational fluid dynamics are discussed in their context. Future trends are also identified and discussed.

Part I

Visualization in Scientific Computing

Chapter 2

Principles of Scientific Visualization

The success of the interdisciplinary field of *scientific computing* is reflected by its multitude of application domains, e.g., including biology, medicine, and physics. Numerical mathematics and computer science in addition to profound domain knowledge are required to ensure a smooth interplay between experiments, theoretical models, numerical simulation, and the analysis of the resulting data:

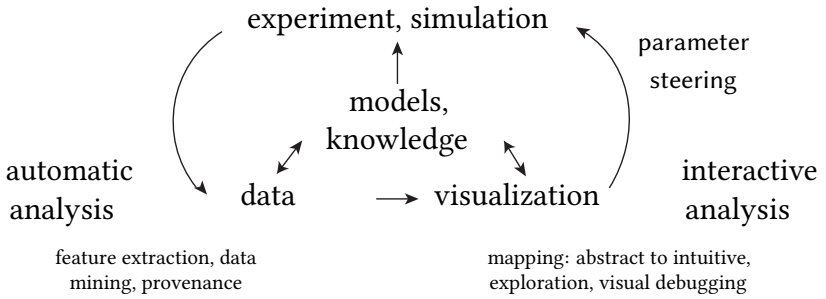


Figure 2.1 — Computational science requires automatic and interactive analysis tools.

Scientific visualization is an integral part of such scientific computing environments. It provides interactive and automatic techniques, enabling new insight and scientific breakthroughs [121]. Traditionally, the focus of scientific visualization lies on the analysis of spatial 2D and 3D data, distinguishing itself from *information visualization*, which often deals with abstract data of higher dimensionality. Still, both fields are closely interconnected and share many of their underlying principles. With the increasing amount of data to be analyzed the notion of *visual analytics* has become popular in recent years. Visual analytics combines the strengths of automatic feature extraction, machine learning or data mining techniques with the capabilities of human cognition, facilitating more transparent interactive problem solving environments (PSE) [91, 186].

In the context of this thesis, new feature extraction techniques and interactive visualization algorithms are presented. This chapter introduces the underlying principles.

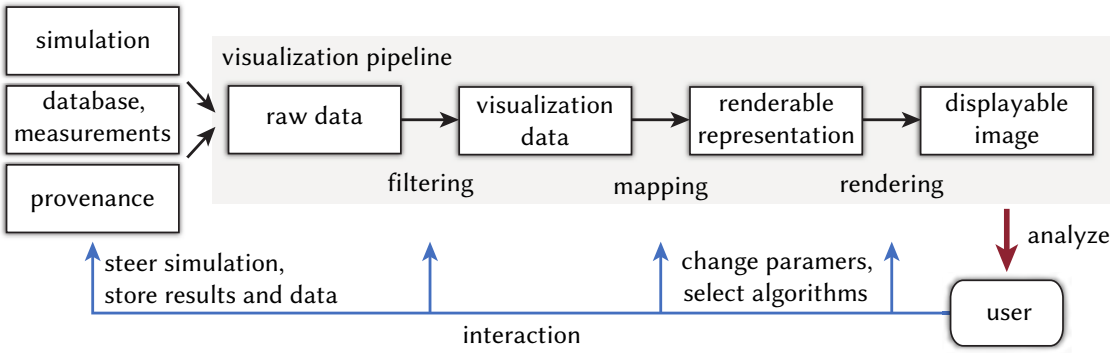


Figure 2.2 — Visualization pipeline in a scientific computing environment according to [206]. Interaction with the data is an important aspect of successful data exploration.

First, the traditional visualization pipeline and its role in the scientific computing environment are discussed (Section 2.1). Second, computer graphics techniques like the 3D rendering pipeline are required for generating meaningful images. Interactive display rates are achieved with GPU-accelerated rendering algorithms (Section 2.2). Finally, the prospects of using GPUs not only for graphics applications, but also to accelerate general-purpose computing, are analyzed. This includes a description of the current hardware architecture and their programming models (Section 2.3).

2.1 Visualization Pipeline

The success of a visualization is measured by its ability to convey the essential information contained in the raw numbers in a way that is easily amenable to the perceptual system of humans. A technical framework for achieving this goal is provided by the *visualization pipeline*. It uses a data flow paradigm to describe the transformation of the abstract raw data into images that can then be analyzed by the user. Figure 2.2 shows a reference model of the pipeline in the context of a scientific computing environment. The data flow model has become popular because it enables a modular implementation of problem solving environments (PSEs) that include visualization as well as simulation [46, 84, 200]. The raw data, i.e., the input to the pipeline, is obtained from real world measurements, or, like in our case, it is generated by a numerical simulation. The input data can be classified according to the dimension of the data domain (the independent variables), and the dimension, type, and range of data values given within this domain (the dependent variables) [21]. The techniques presented in this thesis concentrate on scalar and vector field data in two and three dimensional domains with discretized grid representations (Chapter 3). The traditional visualization pipeline applies three consecutive transformation stages to the raw data, namely *filtering*, *mapping*, and *rendering* [76].

The *filtering stage* refines the raw data resulting in the *visualization data*. This step can include the extraction of a subset of the data, like a single component of multi field data or a region of interest of the data domain. In the case of discretized or incomplete data, the reconstruction of a continuous data representation based on an appropriate model might become necessary, e.g., by using interpolation algorithms. Advanced filtering algorithms might compute derived data objects from the input data, for example, to enable a more abstract analysis.

The *mapping stage* transforms the visualization data into a representation that can be rendered. Amongst others, geometrical representations like points, lines, spheres are common choices to represent characteristic manifolds embedded in the data domain. The values of the associated dependent data variables are mapped to color and transparency using transfer functions. These are then assigned to the geometry.

Finally, the *rendering stage* generates the resulting image using computer graphics algorithms. Nowadays, raster graphics is the de-facto standard for interactive 3D computer games, and it is also commonly used by many scientific visualization techniques. After selecting an appropriate position and orientation, the scene is projected to image space and clipped. Shading is used to simulate lighting with optical models. Other operations include compositing, to combine multiple visualizations, and anti-aliasing, to further improve the quality of the raster image.

The traditional pipeline outlines the typical steps involved in visualization. However, those steps are not strictly followed by all visualization techniques. For example, an intermediate geometrical representation is not necessary in the case of direct volume rendering algorithms that use ray casting instead of rasterization (Section 5.3). Moreover, the input might consist of different types of data, e.g., multi-field data. Such data has to be fused, or, alternatively, for each individual data part a different pipeline instance can be forked. With each instance possibly implementing a different visualization technique, the results have to be compared side-by-side or composited in a final shared rendering stage.

For many visualization techniques, like volume rendering, appropriate transfer functions have to be defined in order to highlight the important characteristics of the data. Transfer function design often involves user interaction. This is in particular true for scenarios where the focus lies on the exploration of phenomena that are not already fully understood, and, therefore, a predefined transfer function is unavailable. Interactive visualization also allows for changing other pipeline parameters, or selecting another visualization algorithm during exploration. Advanced problem solving environments additionally enable the user to steer the simulation [128, 145, 217]. Moreover, reproducibility of analysis results can be ensured by provenance management [55]. User interaction is an essential component in such scientific problem solving environments. Interactive visualization, for example, can facilitate an efficient exploration of large parameter spaces [206]. Visualizing large data however can be computationally very expensive. Modern graphics processing units (GPUs) account for this with their

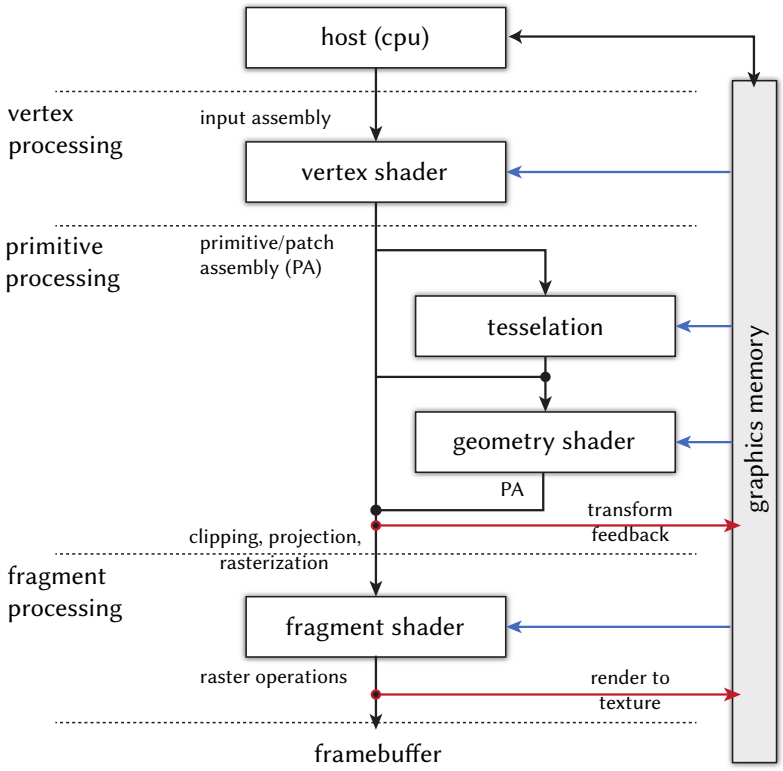


Figure 2.3 — Rendering pipeline implemented by graphics processing units conforming to the OpenGL 4 standard [174].

fast parallel many-core architecture, enabling an efficient execution of the rendering pipeline, which is introduced in the next section. GPUs therefore are often a prerequisite for achieving interactive display rates. Additionally, depending on the storage location and the size of the data, it might be beneficial, or even become a necessity, to execute different parts of the pipeline on different machines. Distributed visualization pipelines [30], for example, can be found in in-situ or computational steering environments, but are also used for cooperative visualization [22] (see discussion in Chapter 17).

2.2 The GPU Rendering Pipeline

The rendering pipeline describes the generation of a 2D raster image from a virtual 3D scene for display on a screen. In the context of the visualization pipeline it is usually used for implementing the rendering stage. A traditional scene representation is typically based on 3D boundary models consisting of a set of vertices and triangles with material attributes like color and texture attached. Additionally, light sources and a virtual camera need to be defined. Figure 2.3 shows a schematic view of the modern

programmable rendering pipeline which has evolved from the traditional fixed-function graphics architecture within the last decade [54, 216]. Because of the complexity of the modern pipeline the fundamental functionality of the traditional pipeline is described first.

The traditional *fixed function pipeline* performs *transform and lighting* operations in the *vertex processing* stage. First, the vertices are transformed from object space to world space (model transformation), followed by a transformation to eye space (view transformation), and finally a camera projection to clip coordinates. The involved affine and perspective transformations are implemented by matrix multiplications that are applied to the vertices given in a homogeneous coordinate representation. The traditional pipeline also applies lighting operations in the vertex processing stage. The following *primitive processing* stage first assembles the primitives (points, lines, or triangles) and clips them at the viewing frustum (primitives outside the frustum are completely discarded). The vertices of the clipped geometry are then transformed to normalized device coordinates, and subsequently to system dependent window coordinates. The rasterization engine then generates the fragments, i.e., the pixels of the output image which cover the clipped parts of the primitive. Attributes attached to the vertices, like color or texture coordinates, are interpolated linearly during this process. Texturing is a common operation performed in the *fragment processing* stage. Traditionally, it is used to determine the fragment color by a lookup in a texture image stored in graphics memory. Finally, the fragment information is combined with the color and opacity values found at its corresponding location in the framebuffer, i.e., it is combined with the contribution of fragments that have already been processed. The raster operations start with the scissor test, followed by the stencil, depth and occlusion test, and blending. The depth test is usually activated. It discards fragments that lie behind parts of the scene that have already been rendered. Only fragments that are not discarded contribute to the framebuffer. Those either completely replace the old fragment value or are combined with it by using linear blending operations.

While the first graphics cards only provided a simple hardware framebuffer, the advent of GPUs shifted the execution of the graphics pipeline completely to dedicated graphics hardware, effectively taking the load from the CPU. Nowadays, the CPU (host) uses APIs like OpenGL [174] (open standard) and Direct3D (Microsoft) to transfer geometry to the GPU, and to issue rendering calls, which are then executed completely on the GPU. Both APIs provide flexible C-like shading languages for programming several stages of the modern GPU pipeline shown in Figure 2.3 (white boxes indicate programmable parts). In order to emulate the traditional pipeline it is sufficient to implement a simple vertex shader, which performs the transformations and the lighting, and a fragment shader, which, for example, performs the texture lookup. However, as discussed below, advanced functionality can be easily realized with the flexible shader programming model. The tessellation and geometry shaders additionally provide means for modifying the geometry directly on the GPU. This further reduces CPU load and allows for significant CPU-GPU memory bandwidth savings. Some parts of the

pipeline, like the primitive assembly, the rasterization, and the raster operations, are still fixed functionality, and hence cannot be programmed directly.

The *vertex shader* processes the assembled vertices after the host has issued a draw call. One shader instance is executed for each vertex independently from other instances, enabling a large number of vertices to be processed in parallel. Vertex attribute information, or data read from graphics memory, like from vertex buffers or cached texture memory, can freely be used for computing a transformed vertex location. These coordinates and the possibly modified vertex attributes are then passed on to the following stages.

Tessellation shaders recently introduced subdivision techniques to the GPU, allowing for fast geometric refinement of quad and triangle patches. In contrast to the one-to-one vertex mapping of vertex shaders, a tessellation shader instance works on one input patch that is defined by a number of control vertices. Using a refinement scheme, the shader instance generates n output triangles from the input patch.

The *geometry shader* obtains one input primitive, possibly including neighborhood information, to generate n output primitives. All output primitives are of the same type, which may differ from the type of the input. It is also noteworthy, that the geometry shader is not meant for mass amplification of geometry, i.e., tasks that are better fitted to the dedicated tessellation stage.

The *transform feedback mode* optionally writes the generated geometry and its attached attributes to graphics memory. This geometry data can then be fed directly into a subsequent execution of the rendering pipeline, e.g., enabling recursive algorithms. In such cases the following fragment processing (rasterization) can be disabled completely.

Both, tessellation and geometry shaders are optional. If enabled, the *fragment shader* is executed after clipping, projection, and rasterization of triangles. The traditional task of this shader is to use the interpolated attribute information attached to the fragment to perform shading operations, e.g., compute the lighting of the fragment. The output always consists of a color tuple and an optional depth value. Note that the location where the fragment is written to the framebuffer has already been fixed by the rasterizer. Instead of using the framebuffer, it is also possible to write one or multiple color tuples to texture images residing in graphics memory. Similar to the transform feedback mechanism, such data can be used in subsequent render passes, for example, to realize deferred shading.

2.3 General Purpose GPU Computing

High performance computing has become a crucial tool for gaining new insight in many scientific research areas [2]. Having developed from dedicated graphics accelerators into general-purpose computing architectures, current GPUs provide an efficient many-core platform for high performance computing. Compared to an implementation on a

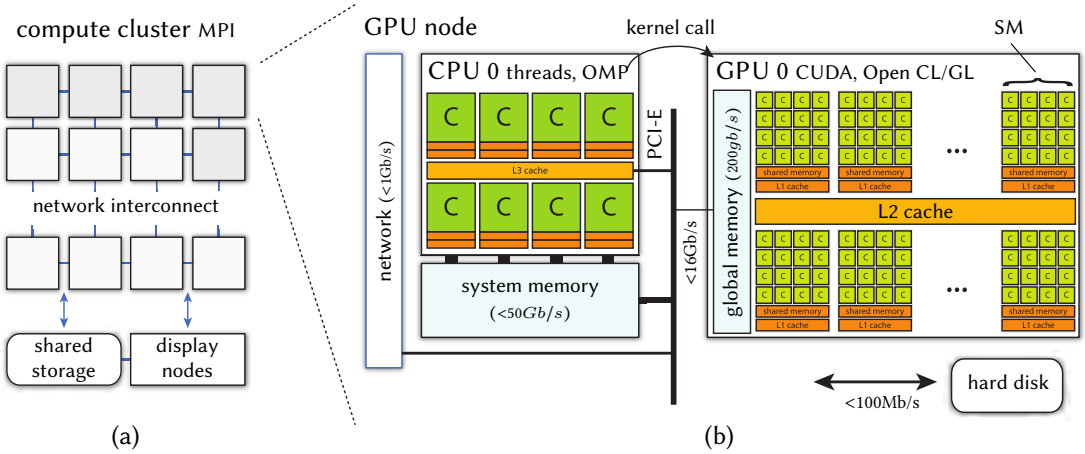


Figure 2.4 — (a) Heterogeneous cluster environment with GPU nodes (grey). (b) Typical architecture of a GPU node equipped with multi-core CPUs and many-core GPUs, here, Nvidia architecture. Local cache hierarchies (orange) provide a much larger memory bandwidth than system and GPU memory, enabling efficient parallel computation.

multi-core CPU, an acceleration of more than one order of magnitude can be achieved, if the investigated problem is amenable to massive parallelization. General-purpose GPU computing (GPGPU) was coined in the early days when researchers strived to harness the massively parallel graphics hardware for non-graphics applications. For achieving this goal, the algorithmic problem had to be formulated by means of the graphics APIs, for example, with GPU shading languages [140]. Early versions of the shading languages, however, where very limited in their functionality, as well as in the precision of their computation. With the introduction of the unified shader architecture in Direct3D 10 [20] the road was paved for more general C-like programming languages like CUDA (compute unified device architecture) from Nvidia [139] or OpenCL [130]. Today, double precision computation is supported by most GPUs. Our interactive visualization algorithms reflect this development. While an OpenGL GPGPU approach is employed by our first technique in order to accelerate particle tracing and ray casting (Chapter 11), later the more flexible CUDA language is used for such purposes.

The architecture of a modern compute node equipped with a multi-core CPU and a GPU is shown in Figure 2.4 (b) [136]. GPUs use a stream processing model that is related to the SIMD (single instruction multiple data) paradigm. Compute kernels, for example, shaders or CUDA kernels, consist of a set of operations that are applied to a large number of similar data elements in parallel. The current GPU architecture (here: Nvidia GK104) implements this paradigm with a set of identical *streaming multiprocessors* (SM) consisting of a large number of simple scalar processor cores (C), with each SM being dedicated to the same kernel operation at a specific point in time. In CUDA one *block of threads* is mapped to one SM. Being limited in the number of threads per block, multiple blocks can be grouped into *grids* for handling larger

compute tasks. A very high computational intensity and throughput can be achieved by restricting the computation to fast hardware registers within the kernel in order to avoid the latency introduced by memory accesses. The GPU architecture additionally allows to improve data locality by providing a hierarchy of caches (similar to CPUs), including very fast local *shared memory*. Efficient usage of shared memory can be crucial and has to be programmed explicitly when using the CUDA programming language. Only threads within the same thread block can communicate through shared memory. While branching and conditional expressions are possible in CUDA, as well in the shading languages, such constructs should be avoided, where possible. Due to the comparatively small bandwidth of the PCI-E bus, reading data from system memory during computation is also not recommended. In contrast to shaders, CUDA kernels additionally offer the flexibility of having free write access to global memory. The memory access pattern within kernels, however, can have a significant impact on global memory throughput. Only kernels that follow an aligned access pattern benefit from the very high bandwidth.[1] An introduction to GPGPU programming with CUDA is given in [139].

Current GPUs offer high performance as well as great flexibility in their programming, but they are still just "number crunching" accelerators which require a host CPU that issues the rendering and kernel calls. However, already today, both hardware architectures show some resemblance (Figure 2.4 (b)), and trends indicate a further convergence, i.e., an increasing flexibility of GPUs on the one hand and an increasing number of CPU cores on the other hand. Nevertheless, the two architectures will persist for some time, with GPUs being throughput-optimized and CPUs aiming for low latency of single threads [136].

Scientific computing applications like numerical simulation or interactive visualization of large data sets are computationally extremely demanding. They require for high-performance computing (HPC) environments like *compute clusters* consisting of a large number of compute nodes that are interconnected by a fast network (see Figure 2.4 (a)). In the context of interactive visualization, heterogeneous GPU cluster environments have become popular. They feature both, nodes with GPU accelerators and CPU-only nodes. Some are even additionally equipped with dedicated display nodes for output on a large power wall or a virtual reality environment. The efficient programming of such heterogeneous parallel environments is of great importance and very challenging [23]. Handling concurrency on the cluster level, for example, with MPI, has to be combined with multi-core CPU programming APIs, like OpenMP, and GPU programming languages, like CUDA or OpenCL. With the complexity of current and upcoming compute environments in mind, new programming paradigms and tools are required which make it easier to benefit from the huge parallel processing power of such systems [2]. CUDASA [181] or rCUDA [44], for example, provide additional abstraction layers to extend the CUDA API to multi-GPU systems and GPU-clusters. OpenCL directly supports heterogeneous systems by enabling OpenCL kernel execution also on

[1] The actual performance impact of non-coalesced memory access depends on the GPU architecture.

non-GPU architectures. By using device abstraction layers the maintainability of the parallel code is improved [143]. Moreover, the implementation effort can be reduced if frameworks are used that support an automatic workload distribution [57]. With the current rapid changes in hardware architectures, further interesting developments are to be expected in the context of the parallel programming challenge [83]. Most of the visualization techniques presented in this thesis benefit from GPU acceleration on single workstations. Some techniques additionally harness the immense compute power provided by GPU cluster environments programmed with MPI, OpenMP, and CUDA.

Chapter 3

Discretized Fields

The visualization techniques presented in this thesis focus on the analysis of physical phenomena that are modeled with continuous approaches, like partial differential equations (PDEs). In the following, the underlying mathematical model of continuous fields is introduced (Section 3.1). To be able to handle fields on the computer, discretized field representations are typically required. Common examples are grid-based and scattered point-based discretizations (Section 3.2). Interpolation algorithms help in reconstructing a continuous representation from the discrete data, for example, for visualization purposes (Section 3.3). Many visualization algorithms additionally require field gradients. These are often not directly available, and hence have to be estimated or reconstructed, as well (Section 3.4). The computation of particle traces is of great importance in the context of vector field visualization, a major topic of this thesis. The involved dynamical system, which consists of a simple ordinary differential equation (ODE), can be solved with the numerical tools presented in Section 3.5. Finally, numerical techniques for the solution of systems of linear equations, e.g., obtained from PDEs discretized on grids, are discussed (Section 3.6). This lays the foundation for the introduction to numerical simulation in Chapter 4.

3.1 Field Definition

A time-dependent field is defined as a function

$$f : V \subset \mathbb{R}^n \times t \to U, \tag{3.1}$$

mapping a subset V of Euclidean space $\mathbb{R}^n$ to values U for each point in time t. This thesis concentrates on real and complex-valued fields in 2D ($n = 2$) and 3D domains ($n = 3$). While continuous functions are suitable for representing many physical phenomenon, there are also important cases with field discontinuities, for example, in compressible flows with shocks or in materials with cracks. *Scalar fields* associate a single scalar value to every point $(\mathbf{x}, t) \in V$, for example, the density of a fluid.

Vector fields, in contrast, describe vector-valued quantities, like the direction and magnitude of flow. Fields with values of even higher dimensionality are called *tensor fields*. Second order tensors (matrices), for example, are often used to model linear direction-dependent field behavior. *Steady fields* do not change over time, i.e., $\partial f / \partial t = 0$. They can be used to describe *stationary* phenomena. Dynamic physical phenomena, in contrast, are characterized by changing field values, introducing time t as additional independent field parameter. For a general time-dependent vector field (also called *instationary* or *unsteady* field) we use the notation

$$\mathbf{u}(\mathbf{x}, t) = \begin{pmatrix} u_1(x_1, x_2, \cdots, t) \\ u_2(x_1, x_2, \cdots, t) \\ \vdots \end{pmatrix}, \tag{3.2}$$

with coordinates $\mathbf{x} = (x_1, x_2, \cdots)$ and scalar values u_i. In the following, Cartesian coordinates are assumed if not stated otherwise.

Given an Euclidean vector space W, the *codimension* allows to refer to certain classes of subspaces $V \subset W$ of equal or lower dimension. The codimension of a subspace V is defined as $\text{codim}(V) = \dim(W) - \dim(V)$. Hence, a plane embedded in 3D space has codimension 1, lines have codimension 2, and the codimension of points is 3—equal to the dimension of the embedding space. Some fields are restricted to codimension-N subspaces of $\mathbb{R}^n$. Most common are fields on manifolds of codimension 1, like surfaces in three-dimensional or curves in two-dimensional space. Important field regions or structures are often represented by such manifolds. Examples are domain boundaries in numerical simulation or field contours.

3.2 Domain Discretization

Analytical field representations are only available in few cases. In general they are difficult to handle on computers. Hence, discretized field representation are typically employed in scientific computing. When discretizing a continuous field one has to differentiate between the discretization of the domain and the representation of the corresponding field values. Though, both are not independent from each other. The finite resolution of the domain discretization (the sampling) is dictated by the characteristics and the variation of the field values and their representation (the signal). The domain can be discretized with *point-based* approximations or with *grids*, which partition the domain into a set of non-overlapping cells. The representation of the field values are attached to the elements of the discretization, for example, to the cells or the vertices. Traditionally, only a single field value or vector is stored per vertex or cell, and interpolation schemes are required to reconstruct a continuous field for visualization (Section 3.3). Some simulation techniques, like the finite element method (FEM) (Section 4.3), store local analytic representations in each cell. While such data already provides a dense field representation, new techniques are required for their visualization—the topic of Part II.

Scattered Data

Scattered data representations store the field data at a number of points distributed in the domain. Such representations are a perfect fit for problems in astrophysics or molecular dynamics where each massive object or atom can be represented by a point with associated properties. Storing no explicit topology has the advantage that points can easily be inserted into the data structure without the tedious process of adjusting the grid connectivity in the neighborhood of the inserted point. Scattered representations can also be used to describe continuous fields. The reconstruction of a continuous field from the discrete representation, however, is typically more complex than interpolation in grids (Section 3.3). This can mitigate the aforementioned advantage of easier adaptation of the domain discretization. Hybrid simulation methods, like the particle-partition of unity method (PPUM), combine the advantage of grids (topology) with the advantage of particle approaches (flexibility in adaption). A novel technique for visualizing such data is given in Section 7.1.

Grid-Based Data

Grid-based discretization, in contrast to scattered data, additionally defines a topology which describes the connectivity between sampling points. The topology of the grid (or mesh) partitions the domain into *cells*, the basic building block of the grid, with each cell covering a connected subset of the domain. *Structured grids* have regular topology with connectivity information given implicitly. The irregular topology of *unstructured grids* in contrast requires explicit storage of connectivity. While structured grids require less storage, and are easier to use with respect to interpolation (Section 3.3) and derivative estimation (Section 3.4), they are also less flexible in adapting to complex domain geometries. Usually, the choice of an appropriate grid type depends on the problem under investigation.

Structured Grids

Structured grids have implicit topology (Figure 3.1 (a)–(d)), which allows for the assignment of a unique index to each vertex. This enables a compact linear storage of the field and fast random access to the data, a property that is essential for efficient parallel GPU algorithms. Structured grids are further distinguished by the representation of their geometry. In the special case of a *Cartesian grid*, the domain is tessellated into equally sized squares (in 2D) or cubes (in 3D). This type of grid has regular geometry, meaning that its vertex coordinates are given implicitly. *Uniform grids*, or *regular grids*, are more general. They allow different spacing along the different dimensions. With *rectilinear grids*, irregular spacing along the dimensions becomes possible. A further generalization, and even greater flexibility, is provided by *curvilinear grids*, which are able to adapt to curved domain geometries more naturally.

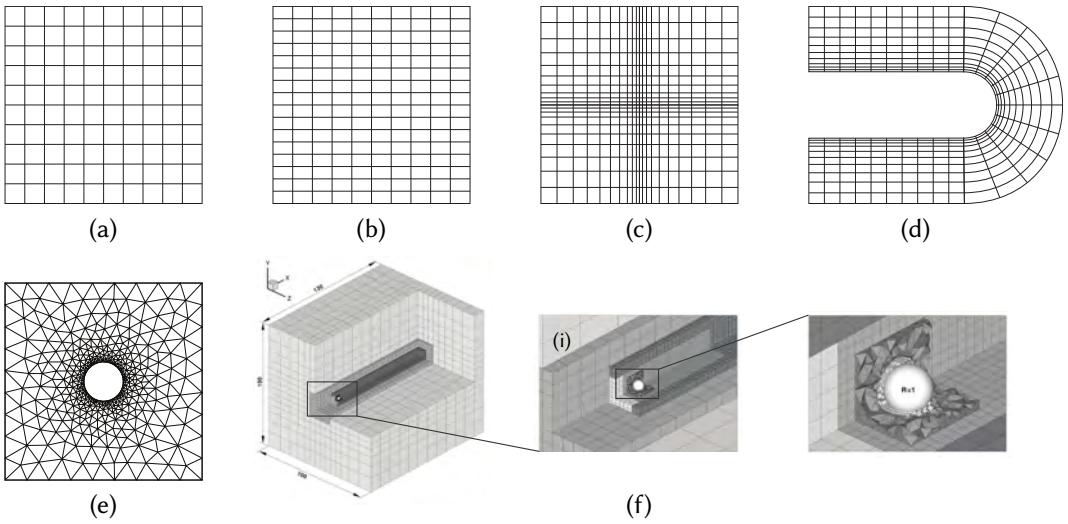


Figure 3.1 — Examples of structured grids: (a) Cartesian, (b) uniform, (c) rectilinear, and (d) curvilinear. Unstructured grids: (e) 2D triangle mesh, (f) hybrid 3D mesh with close-ups showing the use of non-conforming cells (i) and different cell types.

Unstructured Grids

Unstructured grids store their topology and typically also their geometry explicitly. This makes them more general than structured grids, with the benefit of real local grid refinement (Figure 3.1 (e) and (f)). Such flexibility comes at the cost of increased size and complexity of the data structure. As a result, multiple memory indirections are typically involved when accessing a specific field element during computation. Basic examples of unstructured grids are domain tessellations consisting of simplices, like triangle meshes in 2D and tetrahedral meshes in 3D. Quadrilaterals (quads) are another choice for 2D grids. Typical cell types in 3D include tetrahedra, hexahedra, pyramids, and prisms. While it is possible to construct a grid with a single cell type only, increased flexibility is offered by grids that allow different types of cells. Such meshes, for example, are employed by discontinuous Galerkin simulations (Figure 3.1 (f)). Some applications also benefit from grids with general *polyhedral cells*. Usually, convex cell geometries with planar faces are preferred when meshing a domain. However, being restricted to such cells, the approximation of smooth curved boundary geometry can lead to extremely fine sampling resolutions (Figure 3.1 (e)). In such cases, using coarser cells with *curved faces* might be the better choice. Curved geometry, however, can significantly complicate mesh processing. Concave cells, for example, hinder the visibility sorting of elements, which is required by many visualization algorithms. The topology of a grid is said to be *conforming*, if all faces of the mesh are shared by at most two neighboring cells. Loosening this restriction leads to *non-conforming mesh topology* with hanging nodes (T-nodes). An example is shown in the close-up of Figure 3.1 (f). Allowing non-conforming topology can simplify mesh generation. Multiple mesh parts

that have been generated independently from each other can be merged more easily. On the other hand, if T-nodes are involved, the representation of a globally continuous field becomes more complicated.

Adaptive, Hierarchical, and Parallel Grids

Many field phenomena have complex geometry and take place on a large range of spatial scales. Due to the natural limitations in computing resources, an adaptive sampling of the domain, for example, by means of *adaptive unstructured grids*, is required in such cases. It might even become necessary to adjust the spatial sampling dynamically during runtime. *Hierarchical grids* simplify the handling of such operations. Methods based on adaptive mesh refinement (AMR) subdivide grid cells hierarchically into finer grid cells, for instance, a quad cell is replaced by four smaller quads, like in quad trees. Adaptive and progressive visualization techniques can directly benefit from such hierarchical data structures. Regular grids, however, are often preferred to unstructured grids due to their simplicity. With *block structured grids* it is still possible to have a limited degree of adaption with structured grids. These grids decompose the domain into several non-overlapping regions (blocks), resulting in a coarse mesh. Then each block is meshed individually, for example, by using rectilinear and curvilinear grids. Decomposing the domain into smaller parts is also necessary in the case of large simulation problems, e.g., to enable parallel computation on distributed memory systems. The individual parts of the resulting distributed grid have to be coupled during computation in order to obtain a global field solution. In-situ visualization of such distributed grid data can be nontrivial. Distributed volume rendering, for example, requires convex subgrids that can be sorted with respect to the view direction (see Section 8.4). If this cannot be guaranteed, the data has to be redistributed for visualization.

Automatic Mesh Generation and Quality of Meshes

The goal of mesh generation (also called grid generation) is to produce high quality meshes suitable for the target application. Unfortunately, as already hinted, there is no general definition of mesh quality [56]. In the context of numerical simulation, a high quality mesh depicts a mesh that enables efficient computation of accurate simulation results. Low quality meshes, in contrast, might deteriorate the convergence rate of numerical solvers. This can lead to increased simulation runtimes, large residual error, or even convergence failure. Numerical issues introduced by low quality meshes are a general problem in computing. An absolute quality measure, however, can often not be given. These issues also play an important role in computer graphics and visualization. A grid resolution that is too low can become directly visible in regions of the mesh where curved geometry is approximated. Inadequate meshes can also be the cause of low quality gradient estimation, leading to a visually unpleasant appearance or even distracting artifacts in the generated images. In general, if no direct quality

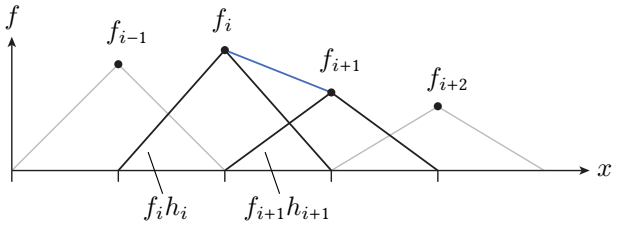


Figure 3.2 — The linear interpolant between sample points $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ (blue line) can be obtained as a sum of weighted spline basis functions $f_i h_i$.

measure is available, the quality of a mesh can still be rated by the properties of its cells. A high quality mesh typically must adhere to several criteria: while aiming for an appropriate sampling of the domain, the size of the cells should not vary too much, small angles should be avoided in cell shapes, and the overall number of grid cells should be minimized. The automatic generation of meshes based on such criteria is challenging and a research field on its own [18].

3.3 Reconstruction of the Continuous Field

A field given by a continuous function f can be represented exactly by a discretized field with a finite number of samples if f (the signal) is band-limited. According to the sampling theory a perfect reconstruction is possible if the signal has been sampled with the Nyquist frequency, i.e., two times the highest frequency of f [176]. In practice signals are pre-filtered (smoothed) to avoid aliasing artifacts that would otherwise appear. Low-pass filters, for example, are used in digital cameras as well as in medical imaging devices, and smoothing or regularization steps are applied by numerical simulations in order to restrict the bandwidth or the solution space. In theory, the reconstructed signal is obtained from the discrete data by the convolution with a sinc filter. However, in practice this is not feasible due to the infinite support of the sinc function, and hence filters with compact support have to be used.

Reconstruction can also be considered from a mathematically less rigorous point of view. In numerical analysis interpolation methods are used to determine values between the discrete samples. A linear behavior of the underlying function f between two neighboring sampling points x_i and x_{i+1}, with values f_i and f_{i+1}, is assumed in the case of univariate linear interpolation

$$f(x) = \frac{x_{i+1} - x}{|x_{i+1} - x_i|} f_i + \frac{x - x_i}{|x_{i+1} - x_i|} f_{i+1} \tag{3.3}$$

in 1D domains. Figure 3.2 illustrates that this corresponds to a weighted sum $f(x) = \sum f_i h_i(x)$ of normalized hat functions h_i, each covering the range of two sampling points. Consequently, only two hat functions have to be evaluated to obtain an interpolated field value at any point x. Linear interpolation can be extended to higher dimensions (multivariate interpolation) by forming a tensor product of linear 1D interpolations along the different dimensions. In 2D, this results in bilinear interpolation

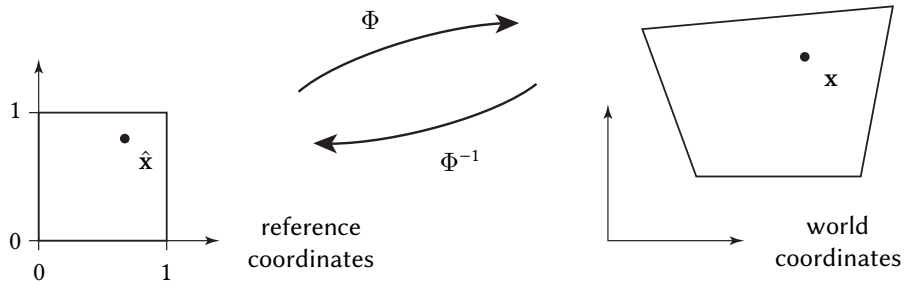


Figure 3.3 — Bilinear interpolation in a generic quadrilateral (right) has to be evaluated in a corresponding reference element (left). Iterative Newton methods are used to determine $\hat{\mathbf{x}} = \Phi^{-1}(\mathbf{x})$, if the inverse geometry mapping Φ^{-1} is not available analytically.

within a rectangular quad that consists of four field values stored at the vertices. Trilinear interpolation within a hexahedron is obtained in 3D. It is important to note that the tensor product introduces nonlinear terms into the interpolation rule.

Linear methods result in reconstructed fields that are C^∞ continuous within the cells, but only C^0 continuous at the cell boundaries. A higher order of continuity can be achieved by involving a larger neighborhood of samples. Convolving the hat function with itself, for example, yields quadratic interpolation functions with a support of three field samples. Repeating this process results in the family of spline interpolation methods of arbitrary order. For structured grids these interpolation techniques can also be formulated as a convolution of the grid data with a reconstruction filter, i.e., a convolution with a B-spline basis function of compact support. Barycentric interpolation provides the counterpart for linear interpolation in unstructured simplical grids, i.e., meshes consisting of triangles in 2D and tetrahedra in 3D. While general polyhedral cells cannot be handled with standard interpolation techniques, mean value coordinates offer one possibility to obtain a smooth interpolant in the case of such cells [85].

Direct application of bilinear and trilinear interpolation schemes is only possible for rectangular quads and hexahedra. Some extra effort is required for generic non-rectangular cells. Therefore, a geometry mapping function Φ is defined that relates the geometry of a generic quad (where we want to interpolate), to a quadratic reference quad (where we can interpolate), as shown in Figure 3.3. The inverse of the mapping function is required to compute the interpolated field value $f(\Phi^{-1}(\mathbf{x}))$. However, the inverse is often not available in a closed analytical form, and iterative Newton schemes (Section 3.6) have to be applied to compute the local coordinate $\hat{\mathbf{x}}$ in the reference quad. Consequently, the involved Jacobian $\nabla\Phi$ has to be non-singular, disqualifying quadrilaterals that are concave or degenerated to triangles, properties that are avoided by affine geometry transformations. Using reference elements is a common concept that is also employed by grid-based numerical simulations like the finite element method (FEM) (Section 4.3). Reference elements increase the flexibility in mesh generation and

simplifying the overall computation. It is also noteworthy that the FEM interpolation model is closely related to the aforementioned spline interpolation approach.

Linear interpolation methods are often preferred in visualization due to their simplicity and efficient evaluation scheme. Trilinear interpolation, for example, only involves eight samples and is directly supported by GPU hardware. However, using tri-quadratic (27 samples) or tri-cubic interpolation (64 samples) can result in superior results, as shown by the example of direct volume rendering [118]. Moreover, higher-order basis functions are increasingly used by numerical simulation, and data from such methods cannot be handled appropriately by the traditional linear visualization techniques. New methods that are able to visualize such data directly are presented in Part II. The misuse of standard interpolation techniques is also discussed in this context. Often popular techniques like bilinear and trilinear interpolation are used, potentially ignoring the constitutive continuous simulation models (Chapter 9).

Scattered data interpolation is only discussed briefly in the context of this work. One possibility is the construction of a triangulation and the subsequent application of cell-based interpolation schemes. Alternatively, meshless reconstruction kernels can be used to evaluate the field directly at arbitrary locations. Such kernels compute a weighted average of the field values, e.g., with the weight being proportional to the inverse distance, as proposed by Shepard [177]. Moving least squares (MLS) provides an alternative for obtaining polynomial approximations. These two techniques can be seen as special cases of a more generic *partition of unity* (PU) framework, which is also employed by the particle partition of unity simulation method (PPUM) described in Chapter 6.1.3. Radial basis functions (RBFs) are a popular alternative for scattered data interpolation because they can reproduce polynomial functions of a given degree accurately. The original RBF and Shepard algorithms are global, i.e., all field samples are involved in each evaluation, which can become very expensive. To account for this, local variants have been developed that consider only a small neighborhood of points. For an efficient detection of the local points located within a given search radius, spatial acceleration data structures are required. Keeping these data structures up-to-date can induce a significant overhead, which can make meshless interpolation rather expensive compared to grid-based interpolation.

3.4 Estimation of Derivatives

Derivatives of scalar and vector fields are required by many applications in scientific computing. Visualization, for example, uses the gradients of scalar fields as normals in lighting models for improved depth perception. Feature extraction techniques and differential equations solved by simulations also heavily depend on the availability of field derivatives. If an analytical continuous representation is not available, derivatives have to be approximated from the discretized representation of the field. The choice of an appropriate scheme depends on the data, like the grid type, and the requirements

of the application, like the accuracy or the smoothness of the reconstructed gradient field. The derivative of linear interpolation methods, for example, is not C^0 continuous, which can introduce visible artifacts. Moreover, without sufficient smoothing, noise can become an issue, in particular if derivatives of second and higher order are computed. Finally, consistency of the reconstructed field and the reconstructed continuous field might be of concern. In the case of polynomial field solutions of high order (Chapter 6), consistency can be easily ensured, but for traditional linear data often different reconstruction approaches are combined.

There are multiple approaches in derivative filter design. The derivative can be computed first at the vertices of the grid, for example, by using a discrete finite difference scheme for structured grids (see below), and continuous interpolation can be applied afterward. Interpolating first also works. A third alternative is the convolution with the continuous derivative of an analytical reconstruction filter [126, 127]. Gradient estimation in unstructured grids poses additional challenges, e.g., due to highly irregular cell structure, and usually also involves higher costs. There are methods based on the weighted average of neighboring cell gradients, as well as regression-based methods which employ least squares fitting [36]. In the following, only discrete finite difference gradient estimation is discussed in detail. The other approaches are covered extensively in the aforementioned literature.

Finite Difference Schemes for Regular Grids

The finite difference approach is applicable to regular grids only. It is based on a Taylor expansion of the field u at a given vertex $\mathbf{x}$. In a 2D grid, the Taylor expansion

$$u_{i+1,j} = u_{i,j} + \left.\frac{\partial u}{\partial x}\right|_{i,j} \delta x + \left.\frac{\partial^2 u}{\partial x^2}\right|_{i,j} \frac{\delta x^2}{2} + \cdots + \left.\frac{\partial^n u}{\partial x^n}\right|_{i,j} \frac{\delta x^n}{n!} \tag{3.4}$$

at a vertex $\mathbf{x}_{i,j}$ describes the value at a neighbor vertex $\mathbf{x}_{i+1,j}$ which is offset in x-direction by δx (see Figure 4.2). The first partial derivative

$$\left.\frac{\partial u}{\partial x}\right|_{i,j} = \frac{u_{i+1,j} - u_{i,j}}{\delta x} + \epsilon_T \tag{3.5}$$

at $\mathbf{x}_{i,j}$ is obtained by restructuring the expansion. Here, all terms with derivatives of order higher than one are merged into ϵ_T. In the following, the partial derivative is denoted as $u_x^{i,j}$. Discarding the higher-order terms ϵ_T results in the *forward difference* approximation

$$u_x^{i,j} \approx \frac{u_{i+1,j} - u_{i,j}}{\delta x} \tag{3.6}$$

of the first spatial derivative. By employing the expansion to the left neighbor vertex $\mathbf{x}_{i-1,j}$ *backward differences* are obtained.

The *truncation error* ϵ_T depends on u and δx, and for a general continuous function u only vanishes in the limit case $\delta x \to 0$. Consequently, forward and backward differences

are accurate for constant and linear functions only, because higher-order derivatives of such functions are zero. The approximation order of a scheme is defined as the smallest power p of δx terms in ϵ_T. Hence, forward and backward differences are first-order accurate $\mathcal{O}(\delta x)$. An improved approximation quality is offered by *central difference* approximations

$$u_x^{i,j} \approx \frac{u_x^{i+1,j} - u_x^{i-1,j}}{2\delta x} \tag{3.7}$$

of the first derivate, which is of second-order accuracy $\mathcal{O}(\delta x^2)$. Second order derivatives can be derived by combining two first order approximations

$$u_{x,x}^{i,j} \approx \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{(\delta x)^2}. \tag{3.8}$$

In general, approximations of arbitrary derivatives and accuracy can be obtained as a weighted sum of function samples at a local vertex neighborhood, the so-called stencil. While larger stencils typically provide improved accuracy, they are also more costly to evaluate, and their use might be problematic near domain boundaries.

3.5 Dynamical Systems and Solution of IVPs

An abstract dynamical system is characterized by the set of all possible states it can take—denoted as the *phase space*, its current state, and its dynamics expressed by a set of rules that describe state transitions. Starting with a given initial state a solution is defined as a *trajectory* in phase space that adheres to the rules of dynamics. Differential equations are a popular mathematical tool for describing the dynamics of continuous physical systems, e.g., using PDEs (Section 4.2). The discussion in the following is restricted to initial value problems (IVPs).

The dynamics of first-order, time-dependent IVPs are captured by *non-autonomous ODEs*

$$\frac{d\mathbf{s}(t)}{dt} = \mathbf{f}(\mathbf{s}, t), \tag{3.9}$$

with the rate of change of the system state $\mathbf{s}$, and initial values $\mathbf{s}(t_0) = \mathbf{s}_0$. *Autonomous ODEs*, in contrast, model time-independent systems, missing the parameter t. In the context of flow visualization, the right hand side $\mathbf{f}$ is considered to be a vector field with $\mathbf{s}$ representing a point in Euclidean space $\mathcal{R}^n$. The system then describes trajectories $\mathbf{s}(t)$ of massless particles that are advected with the flow field (Section 5.4).

In general, the IVP has to be solved numerically. Single step methods only use the information of the previous time step to advance the system state. The *explicit Euler method* of first order, for example, is derived from the forward difference approximation (3.6), resulting in

$$\mathbf{s}(t + \delta t) = \mathbf{s}(t) + \delta t \mathbf{f}(\mathbf{s}(t), t). \tag{3.10}$$

This equation directly advances $\mathbf{s}$ by a given step of size δt, involving only simple evaluations of $\mathbf{f}$ — hence explicit. Its first-order approximation quality is often not sufficient, and very small step sizes have to be chosen. For practical purposes, Runge-Kutta (RK) schemes of higher approximation order are recommended. The popular fourth order (RK4) scheme provides a good performance accuracy tradeoff, in particular if $\mathbf{f}$ involves linearly interpolated field values from discrete grid data [183]. The RK4 integrator consists of four stages, i.e., involves four evaluations of $\mathbf{f}$ per time step. *Adaptive step size control* can further improve the accuracy. Using an error estimator the step size is chosen such that the error stays below a prescribed threshold.

The backward difference scheme results in an *implicit method* of first order

$$\mathbf{s}(t + \delta t) = \mathbf{s}(t) + \delta t \mathbf{f}(\mathbf{s}(t + \delta t), t + \delta t). \tag{3.11}$$

Implicit schemes are more stable and allow for larger time steps. However, equation (3.11) has to be solved for $\mathbf{s}(t + \delta t)$, which is much more time consuming than explicit time stepping. For time-critical applications, like interactive visualization, explicit RK methods are typically preferred.

3.6 Solving Systems of Algebraic Equations

Systems of equations appear in many problems in science and engineering, like in RBF interpolation or the solution of PDEs on discretized domains. Such systems consist of a collection of equations that are coupled by a number of unknown variables, for example, by the field values at the grid vertices. In the following, we only consider systems with a matching number of equations and unknowns.

Systems of nonlinear equations are the most general case. They can be formulated as a multivariate root finding problem $\mathbf{f}(\mathbf{u}) = \mathbf{0}$, which can also be written as a system of individual equations

$$\begin{aligned} f_1(u_1, u_2, \ldots, u_N) &= 0 \\ f_2(u_1, u_2, \ldots, u_N) &= 0 \\ &\vdots \\ f_M(u_1, u_2, \ldots, u_N) &= 0, \end{aligned} \tag{3.12}$$

with N unknowns $\mathbf{u} = (u_1, u_2, \ldots, u_N)$. The space of solutions to a nonlinear system can be very complex, and no general method exists which can guarantee to find a single exact solution. However, in practice *Newton's method* offers a fast generic approximation scheme which is able to numerically compute a single local solution $\mathbf{u}$, given an initial approximation $\mathbf{u}_0$. It is based on a Taylor expansion of $\mathbf{f}$ from which one can derive the linear system

$$\mathbf{J}\delta\mathbf{u}_i = -\mathbf{f}(\mathbf{u}_i), \tag{3.13}$$

with the Jacobian matrix $\mathbf{J} = \partial f_m / \partial u_n|_{\mathbf{u}_i}$, and the approximate solution $\mathbf{u}_i$. The linear system (3.13) is solved for $\delta\mathbf{u}_i$ (see below) and then an (hopefully) improved approximation $\mathbf{u}_{i+1} = \mathbf{u}_i + \delta\mathbf{u}_i$ is computed. These steps are performed iteratively until the

sequence $\mathbf{u}_i$ converges. Convergence against a true solution of (3.12) is not guaranteed, but typically occurs after a few iterations if the initial "guess" $\mathbf{u}_0$ was good enough. In the context of this thesis Newton's method is used for interpolating in generic quadrilaterals (Section 3.3), the solution of nonlinear differential equations (Section 4.3), and the intersection of rays with isosurfaces and parametric patches (Section 8.2.3).

Systems of linear equations are a special case of the general nonlinear system (3.12). They only consist of linear equations

$$a_{m,1}u_1 + a_{m,2}u_2 + \cdots + a_{m,N}u_N = b_m \quad m = 1, \ldots, N. \tag{3.14}$$

In linear algebra this system is commonly written in matrix vector notation $\mathbf{Au} = \mathbf{b}$, with the system matrix $\mathbf{A} = a_{m,n}$, a vector $\mathbf{b}$ on the right hand side, and the vector of unknowns $\mathbf{u}$. In comparison to the nonlinear case, the solution space is much simpler. A unique solution exists in the case of non-singular systems with $\mathbf{A}$ having full rank. However, in general, solving linear systems is a nontrivial problem. Many problems yield ill-conditioned system matrices that are close to singular. Solving such systems numerically can be very inefficient or even fail, because rounding errors might render the system singular during its solution. Preconditioners promise to improve the condition of the system matrix prior to its solution.

A multitude of different solver algorithms exist with efficient CPU and GPU implementations often being available freely. The choice of the solver typically depends on the problem. Small systems, for example, can be directly solved using Gauss elimination. More sophisticated *direct solvers* harness the structure of sparse system matrices, e.g., diagonal or block structured. They factorize the matrix to allow for a more efficient and robust solution. The inversion of $\mathbf{A}$, however, often results in dense matrices which require huge amounts of memory. *Iterative solvers* avoid the inversion and promise a faster solution by improving an initial approximation iteratively, e.g., by using a conjugate gradient approach. This allows for handling very large systems, and additionally can avoid numerical stability issues of direct solvers.

Chapter 4

Numerical Simulation

Close cooperation with the simulation scientists and a basic understanding of simulation ideas has proven to be a key in the development of successful visualization techniques. This chapter gives an overview of simulation concepts and methods in the context of CFD. Section 4.1 starts with a general overview of the numerical problem, leading to the mathematical PDE-based modeling in continuum mechanics (Section 4.2). The discretization and the solution of PDE problems is topic of Section 4.3, with the finite element (FEM) approach being of particular interest for our higher-order visualization techniques. Section 4.4 covers specific challenges and approaches in fluid dynamics, and Section 4.4 concludes with an overview of data sets analyzed by our visualization techniques. This introduction follows the presentation of [51, 148].

4.1 Problem Overview

Classically, scientists gain insight into physics by conducting experiments and by developing theories. By controlling parameters and boundary conditions, the behavior of a real physical system can be analyzed. Virtual experiments on computers can replace real world experiments that are infeasible to conduct, for example, because they are too expensive or risky, or simply would take too long. The simulation of the involved physical phenomena is a complex procedure, which involves multiple disciplines.

First, a mathematical model describing the physics of the real world has to be developed, for example, a PDE model in continuum mechanics. Secondly, the experimental setup has to be described. This includes the definition of a simulation domain, typically a bounded region in space and time. Appropriate initial values and boundary conditions have to be chosen to obtain a *well-posed problem*, which guarantees the existence of a unique solution, and ensures that small changes in those parameters only lead to small changes in the solution. However, many problem descriptions do not come with such guarantees, e.g., chaotic behavior in fluid dynamics and meteorology. Being able

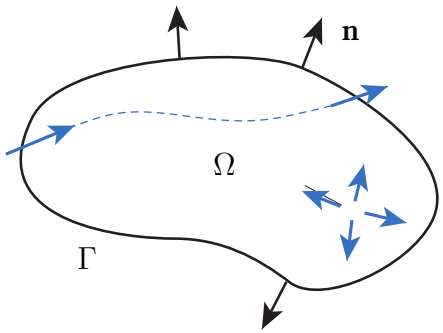


Figure 4.1 — The concept of a control volume Ω is helpful to illustrate the principle of conservation laws: the rate of change of a physical quantity contained within Ω has to be compensated by an according flux across its boundary Γ.

to estimate the quality of the model and its modeling errors is crucial and requires a profound knowledge of the application domain and the mathematical modeling tools.

The second step is concerned with the implementation of the mathematical model on computers. The continuum PDE model has to be discretized, e.g., by using grids, to be able to solve it with numerical algorithms. Here, additional approximation errors are introduced. Numerical analysis, a field at the interface of mathematics and computer science, is concerned with the development of efficient discretization and solver algorithms and their analysis.

4.2 Continuum Mechanics

In continuum mechanics, solids and fluids are modeled as a continuum. This stands in direct contrast to the atomistic approach which studies the dynamics and interaction of individual atoms. Continuum mechanics hides the complexity on these small length scales by providing averaged (or upscaled) descriptions of particle dynamics. The resulting PDE models provide accurate descriptions of many physical phenomena observed at larger length scales. The unknown physical quantities like the momentum or the temperature of the material are represented as fields $\phi(\mathbf{x}, t)$. Corresponding averaged material properties, like viscosity or thermal conductivity that are part of the model, can often be obtained from experiments or from simulations on smaller length scales.

Below, we review how PDE models are derived from fundamental conservation laws, like the conservation of mass, momentum, or energy, following the description of [148]. Examples are given for the simple heat equation, and the Navier-Stokes equations, which describe the complex dynamics of fluids, analyzed by most of our visualization techniques.

Generic Conservation Law

The rate of change of a field quantity ϕ contained within a volume Ω is equal to the production rate minus the flux across the boundary of the volume (see Figure 4.1). The

integral form (4.1) of the balance equation

$$\frac{\partial}{\partial t}\int_{\Omega}\phi(\mathbf{x},t)dV = \int_{\Omega} f(\phi,\mathbf{x},t)dV - \int_{\Gamma}\mathbf{g}(\phi,\nabla\phi)\cdot\mathbf{n}dS, \tag{4.1}$$

also states this, with the production rate f, a vector valued function $\mathbf{g}(\phi,\nabla\phi)$, and a surface integral describing the flux through the boundary Γ, according to equation (2) in [148]. Often a field gradient is used to model the flux. The integral formulation is the basis of the finite volume method (FVM) (Section 4.3).

The divergence theorem $\int_{\Omega}\nabla\cdot\mathbf{g}dV = \int_{\Gamma}\mathbf{g}\cdot\mathbf{n}dS$ allows to express a surface integral by an integral over the enclosed volume. Applying it to the flux integral in (4.1) and bringing all terms to the same side yields $\int_{\Omega}(\frac{\partial}{\partial t}\phi + \nabla\cdot\mathbf{g} - f)dV = 0$, assuming that Ω does not change over time. Given a sufficiently smooth function ϕ, this equation will hold for arbitrary Ω, and the integral can be omitted. This results in the strong pointwise differential formulation

$$\frac{\partial}{\partial t}\phi + \nabla\cdot\mathbf{g} - f = 0, \quad \forall\mathbf{x}\in\Omega, \tag{4.2}$$

which is directly discretized in the finite difference method (Section 4.3).

A third formulation is employed by the method of weighted residuals, an older, but more general technique for solving PDEs, from which the Galerkin FEM (Section 4.3) approach has been derived. By multiplying (4.2) with a weight function $w(\mathbf{x})$ one obtains

$$\int_{\Omega}(\frac{\partial}{\partial t}\phi + \nabla\cdot\mathbf{g} - f)wdV = 0, \tag{4.3}$$

which is still satisfied for all solutions of (4.2). Often, the divergence theorem is applied one more time to obtain the *weak formulation*

$$\int_{\Omega}[(\frac{\partial}{\partial t}\phi - f)w - \mathbf{g}\cdot\nabla w]dV + \int_{\Gamma}\mathbf{g}\cdot\mathbf{n}wdS = 0 \quad \forall w\in V, \tag{4.4}$$

according to [148]. This reduces the order of differentiation and hence increases the space of admissible solution functions ϕ. Moreover, the integration of the Neumann boundary conditions becomes easier.

Heat Equation

The heat equation serves as a simple example. It is derived from the principle of energy conservation in combination with Fourier's law of heat conduction $\delta Q = -k\nabla T$, which relates the flux of energy δQ across a surface element to the negative gradient of the temperature distribution T. Here, the constant k represents the thermal conductivity of the material. From experiments it is also known that the internal energy per volume

element is proportional to its temperature. This results in the integral formulation of the balance equation

$$\frac{\partial}{\partial t}\int_{\Omega} c_p \varrho T dV = \int_{\Gamma} k \nabla T \mathbf{n} dS, \tag{4.5}$$

describing the conservation of energy, with material density ϱ and specific heat capacity c_p. Applying the divergence theorem to the surface integral yields the differential formulation

$$\frac{\partial T}{\partial t} - \alpha \Delta T = 0 \tag{4.6}$$

with ($\alpha = k/(c_p \varrho)$), assuming that c_p, ρ, and k are constant. This is a simple linear PDE of second order with time-dependent solutions $T(\mathbf{x}, t)$ describing the temperature distribution. Its stationary variant with $\partial T/\partial t = 0$ results in the well known Laplace equation $\alpha \Delta T = 0$ with steady state solutions $T(\mathbf{x})$. A full specification of the problem still requires the application of appropriate boundary conditions, as discussed in Section 4.2.

The Navier-Stokes Equations of Fluid Dynamics

The Navier-Stokes equations are an example of a more complicated system of non-linear PDEs. They describe the motion of Newtonian fluids, like water, for which shear stresses are proportional to the shear strain rate the fluid is exposed to. The fluid contained within a volume is described by its density distribution $\varrho(\mathbf{x})$ and its motion, which is encoded in a velocity field $\mathbf{v}(\mathbf{x}, t)$. Given a finite control volume Ω, conservation of momentum of a fluid

$$\frac{\partial}{\partial t}\int_{\Omega} \varrho \mathbf{v} dV + \int_{\Gamma} \varrho \mathbf{v}\mathbf{v} \cdot \mathbf{n} dS = \int_{\Gamma} \mathcal{T}(\mathbf{v}) \cdot \mathbf{n} dS + \int_{\Omega} \varrho \mathbf{f} dV \tag{4.7}$$

can be derived from the generic conservation law (4.1) with the conserved quantity $\phi = \varrho \mathbf{v}$ (momentum), the flux term $\mathbf{g} = \varrho \mathbf{v}\mathbf{v} - \mathcal{T}(\mathbf{v})$, and body forces $\mathbf{f}$, according to [51]. The stress tensor

$$\mathcal{T}(\mathbf{v}) = -(p + \frac{2}{3}\mu \nabla \cdot \mathbf{v})I + \mu(\nabla \mathbf{v} + (\nabla \mathbf{v})^{\top}) \tag{4.8}$$

models the transport rate of momentum induced by internal molecular forces reacting to external deformations of the Newtonian fluid. The first term describes the static pressure p of the fluid. The others model the linear relation between the expansion and strain rate to the stress rate, with the dynamic viscosity μ of the fluid being the factor of proportionality. Applying the divergence theorem and the steps described in the section about generic conservation laws leads to the momentum conservation part of the Navier-Stokes equations

$$\underbrace{\frac{\partial(\varrho \mathbf{v})}{\partial t} + (\mathbf{v} \cdot \nabla)(\rho \mathbf{v})}_{=: D(\rho \mathbf{v})/Dt} = \nabla \cdot \mathcal{T} + \varrho \mathbf{f}. \tag{4.9}$$

The material derivative $D(\rho\mathbf{v})/Dt$ can be used to describe the rate of change in a reference frame moving with the fluid. This highlights the similarity to Newton's law of motion. In the 3D case the equation consists of three nonlinear PDEs but contains more than three unknowns, for example, the three components of the velocity field $\mathbf{v}$, the pressure, and the density. Depending on the assumptions that can be made about these quantities or about other properties of the fluid, like temperature, more equations have to be added to the system, e.g., an equation of energy conservation. In general, at least the *continuity equation* describing the conservation of mass $\frac{\partial \varrho}{\partial t} + \nabla \cdot (\varrho \mathbf{v}) = 0$ is required. Assuming incompressibility $d\varrho/dt = 0$ leads to the special case of the *incompressible Navier-Stokes equations*

$$\frac{\partial}{\partial t}\mathbf{v} + (\mathbf{v} \cdot \nabla)\mathbf{v} = -\frac{1}{\varrho}\nabla p + \frac{\mu}{\varrho}\nabla \cdot \left(\nabla \mathbf{v} + (\nabla \mathbf{v})^{\mathsf{T}}\right) + \mathbf{f} \tag{4.10a}$$

$$\nabla \cdot \mathbf{v} = 0, \tag{4.10b}$$

which is valid for many types of fluids, like air at low speeds. Ignoring viscous effects, in particular the nonlinear convection term $(\mathbf{v} \cdot \nabla)\mathbf{v}$, results in the Stokes equations. However, these nonlinear terms play a major role in vorticity generation and cannot be neglected in many practical flows, in particular if solid boundaries are involved.

Boundary Conditions and Initial Values

A multitude of different types of boundary conditions exists, and their choice typically depends on the application. *Dirichlet conditions* are used to specify the value of the solution on the boundary of the domain. In the case of equation (4.6) this is useful to model a heated wall by prescribing its temperature distribution. *Neumann conditions*, in contrast, specify the derivatives of the solution on the boundary, like zero temperature change to model perfect thermal insulation. Another example are *periodic boundaries*. They allow to simulate large problems by mirroring the finite domain at some of its boundaries. Finally, the time domain of evolution problems is often treated as an IVP (Section 3.5), requiring the specification of initial field values. In general, boundary conditions have to be chosen carefully to obtain a well-posed problem.

4.3 Simulation Methods

Only few PDE problems can be solved analytically. The differential operators of the chosen mathematical formulation (Section 4.2) impose restrictions on the space of the possible solution functions $u \in U$. As an example, C^k continuity is required if k-th derivatives of u are involved in the PDE. To be able to solve for u numerically a further restriction is necessary—the number of degrees of freedom has to be made finite. This can be achieved by selecting a finite set of N basis functions ψ_i to span a solution

space, i.e., an arbitrary function represented as

$$u_\psi(\mathbf{x}) = \sum_{i=1}^{N} c_i \psi_i(\mathbf{x}), \tag{4.11}$$

which suits the problem and its equations. Spectral methods for instance select basis functions with global support on the whole domain Ω, e.g., from a Fourier series. In the following, we restrict the discussion to methods like the finite element method, which uses basis functions with local support, i.e., that are nonzero in small sub domains only. Note the similarity to the interpolation problem discussed in Section 3.3. Here, the task of the simulation is to compute the set of c_i belonging to the solution of the problem. The finite difference method (FDM), the finite volume method (FVM), and the finite element method (FEM) use grids to partition the domain into small sub domains. The equations are then discretized on the grid and the boundary conditions are incorporated, yielding a system of algebraic equations that can be solved for the c_i (see Section 3.6).

The solution approach typically depends on the type of the problem. For stationary problems a system of equations has to be solved, see Section 3.6. Time-dependent problems described by hyperbolic PDEs can be considered as IVPs consisting of an ODE system and initial values. This makes them amenable to *explicit time-marching schemes*, which directly propagate the initial solution, see Section 3.5. Explicit schemes do not require the solution of equation systems, but the chosen time step Δt has to satisfy the CFL condition to ensure that the transported quantity is not advected further than the size of one grid cell within a single step. Consequently, using a finer spatial resolution also demands for smaller time steps. This negates the benefit of adaptive grid refinement if used together with a globally fixed time step. *Fully implicit schemes*, in contrast, are more stable and allow for larger step sizes. They employ backward time discretization, resulting in a system of algebraic equations. In the case of a nonlinear system of equations, the solution of the previous time step can provide a good initial guess for the Newton method, allowing for faster solver convergence. Strong or weak coupling approaches can be used to solve problems consisting of multiple PDEs.

In the following, the FDM based on the strong formulation (4.2), the FVM based on the integral formulation (4.1), and the FEM based on the weak formulation (4.4) are discussed in more detail.

Finite Difference Method

The finite difference method (FDM) is used to approximate differential equations of PDE problems in the strong formulation (4.2). The solution is computed at a finite number of points that are most often organized as a rectangular grid with regular topology. This limits the geometric complexity of the domain. While the method does not construct a set of basis functions like (4.11), it can still be considered as a special case of the more general FEM (Section 4.3) for particular cases using a FDM grid.

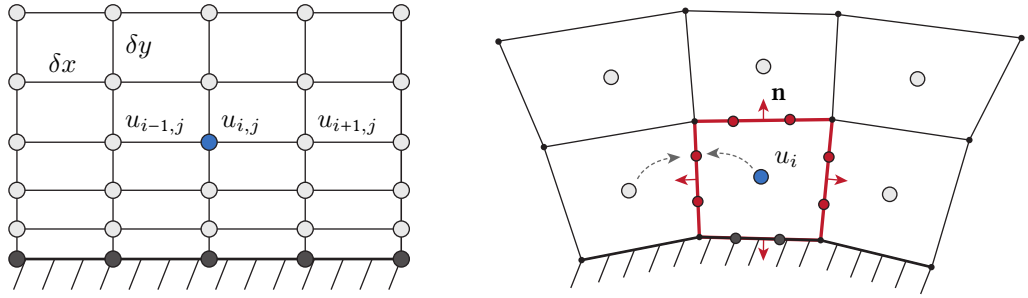


Figure 4.2 — (a) The finite difference method approximates PDEs by discretizing the equations on structured grids with the unknowns $u_{i,j}$ placed at the grid vertices $\mathbf{x}_{i,j}$. (b) Cell-centered finite volumes are based on integral formulations of the conservation laws and store the unknown field value at the center of the control volumes.

An example of a typical grid used by the FDM is shown in Figure 4.2 (a). The terms of the differential equation are discretized at the N interior vertices $\mathbf{x}_{i,j}$. Spatial derivatives are expressed with values at a set of neighbor vertices L_i by means of appropriate finite difference schemes (Section 3.4). For linear problems this yields a linear algebraic equation

$$a_{i,i}u_i + \sum_{j \in L_i} a_{i,j}u_j = b_i \quad i = 1, \ldots, N \tag{4.12}$$

for each vertex $\mathbf{x}_{i,j}$, with weights $a_{i,j}$ depending on the chosen finite difference scheme and unknowns u_i. Dirichlet boundary conditions can be integrated naturally by simply using the imposed values at the corresponding boundary vertices, whereas Neumann conditions are more complicated. The system of equations (4.12) can also be written as $\mathbf{M} \cdot \mathbf{u} = \mathbf{b}$, with the system matrix $\mathbf{M}$ of size $N \times N$ and the vector of unknown vertex values $\mathbf{u}^T = (u_1, u_2, \ldots, u_N)$.

The Laplace equation (Section 4.2) serves as an example. Using the second-order scheme (3.8) to discretize the Laplacian operator $\Delta = (\partial_{x,x} + \partial_{y,y})$ yields N equations of the form

$$\frac{a_{i-1,j}u_{i-1,j} - 2a_{i,j}u_{i,j} + a_{i+1,j}u_{i+1,j}}{(\mathbf{x}_{i+1,j} - \mathbf{x}_{i-1,j})^2} + \frac{a_{i,j-1}u_{i,j-1} - 2a_{i,j}u_{i,j} + a_{i,j+1}a_{i,j+1}}{(\mathbf{x}_{i,j+1} - \mathbf{x}_{i,j-1})^2} = 0. \tag{4.13}$$

Each equation involves the field values at four neighbor vertices of $\mathbf{x}_{i,j}$, resulting in a sparse system matrix $\mathbf{M}$ with five entries per row. The regular topology of FDM grids often yield matrices with simple structure, like band matrices, that can be solved efficiently with one of the solvers described in Section 3.6. The regularity of structured grids, however, adversely prevents local grid refinement. In order to reduce the truncation error of the chosen FD scheme the grid has to be refined globally.

Finite Volume Method

The finite volume method (FVM) partitions the domain into a finite number of N *control volumes* (CV), and within each CV the solution is approximated with a constant field value, such that the conservation principle (4.1) is fulfilled. In contrast to FDM, flexible unstructured grids are supported naturally. The placement of the degrees of freedom is also more flexible. A common choice are cell-centered finite volumes, where CVs coincide with grid cells and unknowns are located at cell centers (Figure 4.2 (b)). For each CV the integral equation (4.1) is approximated individually with quadrature rules. This results in one equation per CV, and in the case of an implicit approach, in a global system of N coupled algebraic equations with N unknowns.

With the unknowns u_i representing the cell-average of the solution the volume integral can be approximated with the mid-point quadrature rule. The surface integrals (the fluxes through the CV boundaries) are also evaluated with the mid-point rule or with higher-order rules using more than one quadrature point. To obtain the missing field values at the quadrature points (the red samples in Figure 4.2 (b)) interpolation techniques have to be employed. The quality of the involved gradient estimation typically determines the approximation order of the whole method as well as the number of non-zero entries in the rows of the sparse system matrix.

The FVM reproduces the finite difference scheme if regular grids are chosen in combination with appropriate numerical fluxes and interpolation techniques. The FVM is locally conservative. This is achieved by coupling neighboring cells through the flux across their shared boundary: the loss of one cell is exactly compensated by the gain of neighboring cells. Methods like the FEM, in contrast, are globally conservative only. This can deteriorate the approximation quality for certain problems. It is noteworthy that the FVM can still be derived as a special case from the weighted residual form (the basis of the FEM) by choosing piecewise constant weight functions $\varphi_i = 1$ within each cell c_i, and $\varphi_i = 0$ outside. The piecewise constant solutions of the basic finite volume approach necessarily produce discontinuities at the cell boundaries. A visualization system devoted to visualize the true computed solution, e.g., for visual debugging purposes, should refrain from applying interpolation techniques.

Finite Element Method

The finite element method (FEM) belongs to the class of Galerkin methods. The general goal of these methods is to find a solution function $u(\mathbf{x})$ that fulfills the weak formulation (4.4) within the whole domain Ω. First, the domain is discretized, e.g., with an unstructured grid, as shown in Figure 4.3. To derive a finite set of algebraic equations the solution space U is restricted to the space of *piecewise polynomial functions* of degree p, and the weight functions φ_i are chosen from the test space V which is typically equal to U (Bubnov-Galerkin). Note that these spaces have to be at least C^0 continuous. Hence the FEM is also called a continuous Galerkin method. In 1D the polynomial solution u_i of cell i is given as the linear combination of $p+1$ polynomial basis functions

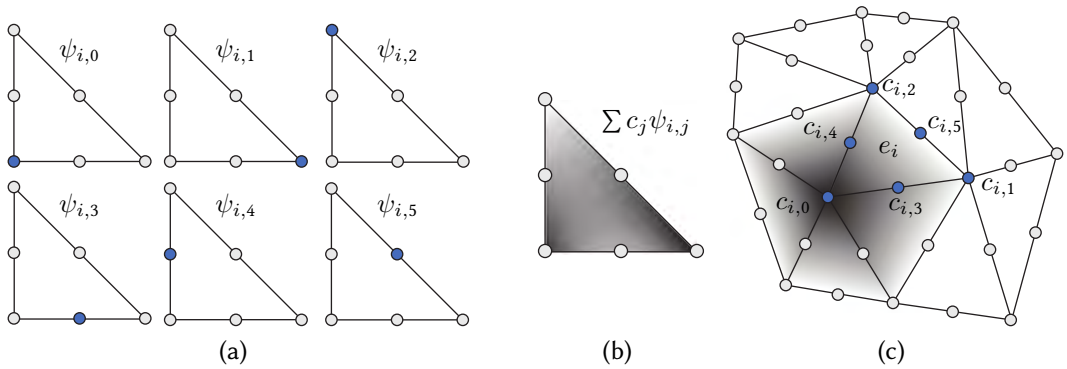


Figure 4.3 — (a) The six characteristic P_2 FEM basis functions $\psi_{i,j}$ have value one (black) at corresponding nodes (blue) and zero (white) at the other nodes. (b) Example of a quadratic solution within the reference triangle. (c) Triangular FEM grid and $\psi_{i,0}$.

$\psi_{i,j}$ with compact support and unknown coefficients $c_{i,j}$, according to equation (4.11). In the 2D case $(p+2)(p+1)/2$ basis functions are required. Figure 4.3 (a) shows the six basis functions of a quadratic nodal P_2-basis ($p = 2$), which contribute to a single triangular cell. The corresponding six nodes are located at the vertices and on the edges of the cell, e.g., of cell e_i in Figure 4.3 (c). The basis functions overlap multiple neighboring cells to ensure continuity in the global solution, as shown at the example of the basis function $\psi_{i,0}$. Each $\psi_{i,j}$ additionally has the property that it has a value of one at its corresponding node and zero at all other nodes of the mesh.

A global ordering of all N nodes in the grid allows for writing the global solution as a linear combination of the N basis functions

$$u(\mathbf{x}) = \sum_{k=1}^{N} c_k \psi_k(\mathbf{x}), \quad \mathbf{x} \in \Omega. \tag{4.14}$$

A full set of fixed c_i represents a solution if (4.4) is satisfied for the N basis functions $\varphi_i \in V$, which are typically equal to the ψ_i. This again yields a system of N equations that has to be solved. The respective entries of the system matrix depend on the integrals in (4.4), which are evaluated during the so-called matrix assembly stage. Fortunately, most entries are zero due to the local support of the basis functions ψ_i, leading to sparse systems.

The integrals depend on the shape of the grid cells, which can become rather complex. Curved cells, for example, might be used to improve the approximation of curved boundary geometry. Constructing the polynomial basis explicitly in world space is often not feasible in practice. This is why many FEM approaches use *reference elements*, with additional geometry mappings Φ that map the polynomial ansatz functions from the reference element to world space. This comes with the benefit that the basis only has to be constructed once for each type of grid cell, each type represented by a simple reference geometry (Figure 4.3 (a)). Hence, the storage of additional basis information

for each individual cell is avoided. The reference element concept is similar to the concept used for interpolation in generic quadrilaterals (see Figure 3.3). The integrals in (4.4) still need to be evaluated in physical space, e.g., with numerical quadrature rules. Thus, the geometry mapping Φ has to be inverted. This is also required when the FEM solution is evaluated at arbitrary points, for example, for visualization and analysis purposes. Methods for visualizing polynomial FEM data are discussed in Part II.

4.4 Challenges in Fluid Dynamics and Data Sets

Characteristic Fluid Behavior

Fluid flow exhibits a wide range of characteristic behavior—from smooth laminar flow to turbulent flow with a chaotic structure [51, 52]. The different flow regimes are classified by the dimensionless *Reynolds number*

$$Re = \frac{\varrho |\mathbf{v}| L}{\mu}, \tag{4.15}$$

which represents the ratio of inertial forces to viscous forces, with fluid velocity $\mathbf{v}$, fluid viscosity μ, and density ϱ. The quantity L specifies the characteristic size of the investigated system, e.g., the size of an obstacle. Consequently, different fluid behavior is observed if the same parameters are used for scaled problems.

Figure 4.4 (a)–(e) depict the different regimes of a flow around a cylinder that appear when increasing Re. A steady *laminar flow* is observed at very small $Re \approx 10^{-2}$. The flow is able to follow the contour of the cylinder without detaching from it. Separation happens at $Re \approx 20$ and vortices form behind the obstacle, but still the flow stays steady. Unsteady behavior develops after a further increase of the Reynolds number ($Re \approx 100$), e.g., at higher velocities. Now, vortices detach from the upper and the lower part of the obstacle in a periodic manner, called vortex shedding. This phenomenon is known as the *von Kármán-vortex street*. At larger Re the flow enters a transitional phase. The vorticity generated near the solid boundary of the cylinder is no longer able to diffuse into a larger region of the domain and a boundary layer with chaotic structure begins to develop ($Re \approx 10^4$). At even higher Reynolds numbers flow separation is delayed and the diameter of the turbulent wake decreases ($Re \approx 10^6$). This actually reduces the drag exerted on the cylinder.

In *free flow* regions, where the influence of solid walls is negligible, *inviscid* flow behavior is observed in the case of many fluids, like water and air. In those regions, shear stresses can be neglected. *Bernoulli's principle* provides a formulation of the principle of energy conservation for inviscid steady flows. The principle states that an increase in fluid velocity—the fluid's kinetic energy—goes along with a decrease in pressure—the fluid's potential energy.

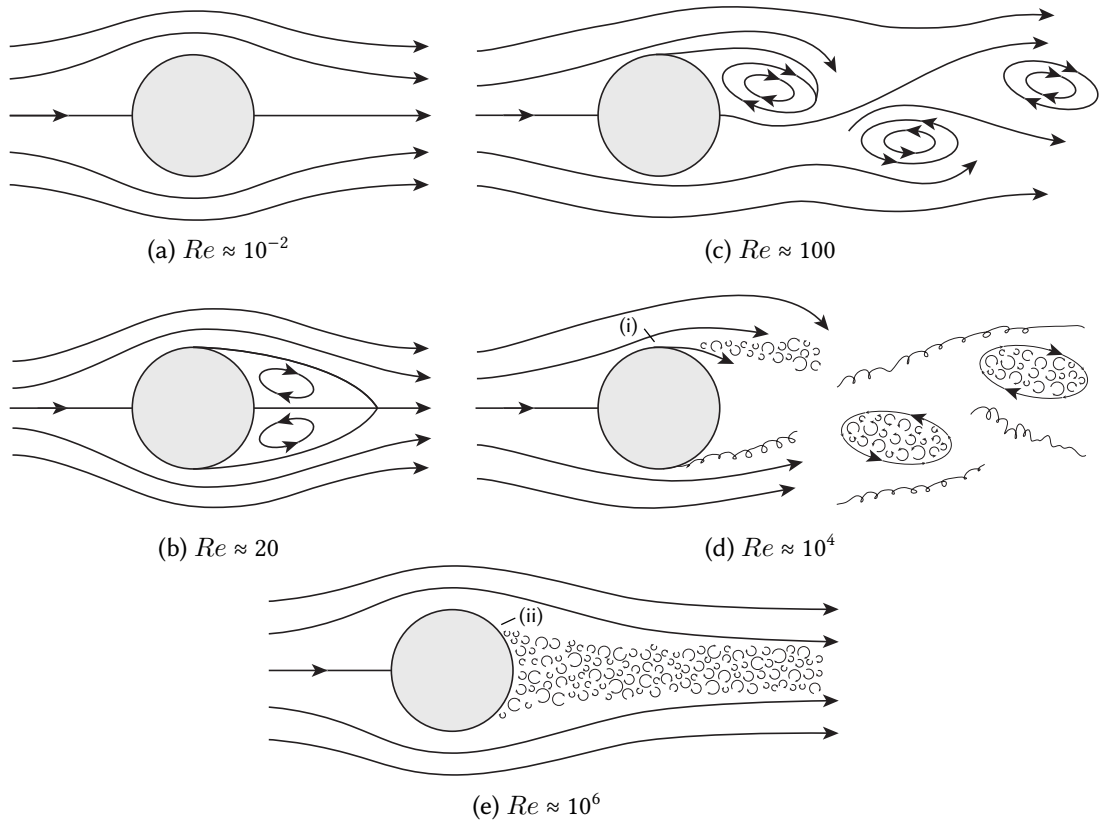


Figure 4.4 — Cylinder flow at increasing Reynolds numbers. (a) Steady laminar, no separation. (b) Steady laminar, separation. (c) Periodic laminar, vortex shedding. (d) Periodic turbulent, laminar boundary layer and wide wake. (e) Turbulent boundary layer and delayed separation (ii), resulting in narrow wake [52].

However, this simple relation no longer holds if the magnitude of viscous forces becomes of the same order as the inertial forces. This, for example, happens in the vicinity of solid boundaries. Directly on the surface, the fluid is forced to have zero velocity due to *no-slip boundary conditions*. This induces strong shear forces and a strong velocity gradient normal to the surface, resulting in the formation of a so-called *boundary layer*, where velocity magnitude is smaller than 99% of the free flow velocity (see Figure 4.5) [167]. The shear stresses are also the main source of *vorticity* $\nabla \times \mathbf{v}$ generated near the surface. Vorticity represents the "local rotation" of fluid parcels (the rotation axis and magnitude). Similar to momentum, vorticity is conserved. It advects and diffuses with the fluid and might develop into vortices and turbulence.

Laminar boundary layers appear at low Re, and exchange momentum only between adjacent flow layers. The transition to *turbulent boundary layers*, i.e., macroscopic mixing, happens at higher Re when the laminar flow is no longer able to sustain the dissipation of "macroscopic kinetic energy" into internal energy, which represents the random motion of particles on the microscale. A turbulent layer can additionally delay

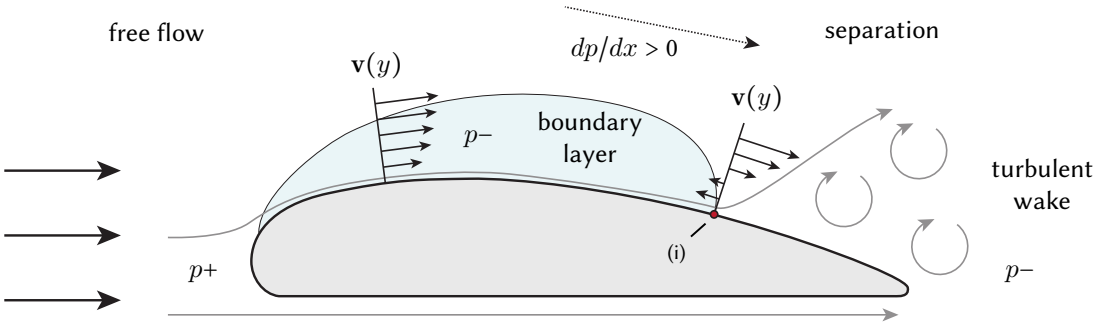


Figure 4.5 — Boundary layer (blue) and flow separation near a no-slip boundary.

the separation of the flow (compare (i) and (ii) in Figure 4.4), or lead to reattachment (Figure 4.6), a beneficial property in the case of airfoils. Even for strongly turbulent flows, a very thin *laminar sublayer* can persist directly at the boundary. *Flow separation* happens because of the *adverse pressure gradient*, which decelerates the fluid while traveling along the surface. The retardation can reverse the flow direction (see (i) in Figure 4.5) leading to separation and the formation of a turbulent wake. As this also goes along with increased pressure drag, airfoils are constructed with a streamlined geometry to prevent early separation. This can additionally be supported by vortex generators to facilitate the transition from laminar to turbulent boundary layers. Turbulent mixing accelerates the transfer of energy from larger to smaller scales, i.e., the kinetic energy is effectively transported from larger to smaller vortices, until its final dissipation into internal energy on the microscales, e.g., going along with an increase in temperature. This characteristic behavior is explained by Kolmogorov's *cascade theory*. Global weather phenomena, for example, consist of very large vortices (or eddies). These vortices are transient structures that break up and distribute their energy onto a number of smaller vortices. The process repeats itself recursively until dissipation on the Kolmogorov microscales.

At low Mach numbers $M = |\mathbf{v}|/c$, where the fluid velocity is much smaller than the speed of sound c, compressible fluids, like air, can be treated as incompressible flows, typically for $M < 0.3$. Fluid compressibility, however, becomes important when $|\mathbf{v}|$ approaches c (transonic flow) or becomes larger than c (supersonic flow). Phenomena like *shocks*, which are characterized by very high gradients and discontinuities in the field quantities, like in the pressure, begin to form in these cases. While liquid flow typically takes place at comparatively small Mach numbers, shock phenomena play a major role in aero- and gas dynamics.

The Navier-Stokes equations (4.9) have proven to model many practical applications correctly. However, even in the case of the simple cylinder flow, it is practically impossible to predict the rich characteristic flow behavior by simply studying the mathematical equation. Complex flow phenomena, like turbulence, are still not fully understood, and even very basic mathematical properties of the nonlinear equations, like the existence

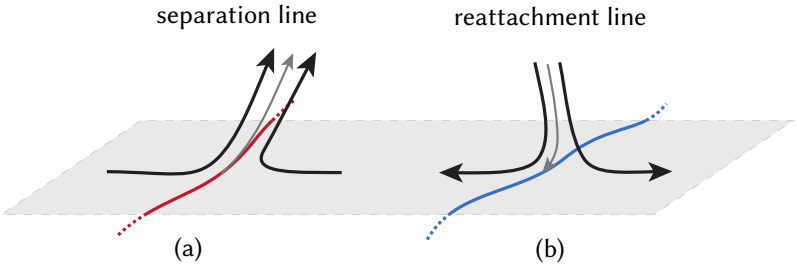


Figure 4.6 — Lines of flow separation (red) (a), and reattachment (blue) (b)

of a unique solution for a given set of boundary conditions, has not been proven yet. Finding such an elementary proof has become one of the millennium prize problems [49]. A comparison that describes the difficulties introduced by nonlinearity in an intuitive way was given by Gleick in his popular book on chaos theory:

> "Analyzing the behavior of a nonlinear equation like the Navier-Stokes equations is like walking through a maze whose walls rearrange themselves with each step you take" [70].

While the numerical solution of the equations (Section 4.3) does not directly help in consolidating the aforementioned mathematical foundation, it does enable numerical experiments. Even if a proof should be found, numerical techniques are still required to actually compute a solution. These solutions have to be studied to gain more insight, i.e., a better understanding of the structure and the dynamics of fluid flow. In retrospect, illustrations like the bifurcation diagram and the Lorenz attractor helped to reveal structure and patterns in the seemingly chaotic behavior induced by simple deterministic nonlinear equations [115, 109]. There are many opportunities where modern visualization techniques, like the feature extraction methods in Part III, can support the analysis of more complex multi-dimensional systems, like vector fields from CFD. The topological methods for unsteady flow discussed in Part IV are even directly related to the aforementioned dynamical systems or chaos theory.

Direct Numerical Simulation and Turbulence Modeling

Direct numerical simulation (DNS) discretizes the Navier-Stokes equations (Section 4.2) directly. In the case of turbulent flow, however, this implies that all temporal and spatial scales down to the Kolmogorov microscales have to be resolved, typically leading to extremely fine grids and small time steps. In practice this is often not feasible, in particular for flows with high Reynolds numbers, which are found in many engineering applications, like flow in turbo machines or around airplanes. Hence, DNS is nowadays mostly employed in fundamental research and less in engineering. Moreover, DNS can be used to validate the turbulence models employed by RANS solvers.

The *Reynolds averaged Navier-Stokes* (RANS) solution approach is often employed in engineering. It decomposes the velocity field of a turbulent flow $\mathbf{u}(\mathbf{x},t) = \bar{\mathbf{u}}(\mathbf{x}) + \mathbf{u}'(\mathbf{x},t)$ into a time-averaged mean velocity $\bar{\mathbf{u}}$ and time-dependent velocity fluctuations $\mathbf{u}'(\mathbf{x},t)$.[1] RANS only solves for the averaged field component and hence allows for comparably coarse grids. The subgrid-scale field fluctuations, however, cannot simply be neglected, as they introduce additional unknowns into the Reynolds averaged Navier-Stokes equations. In order to solve the resulting closure problem the unknown Reynolds stress tensor needs to be approximated with *turbulence models*. Popular examples are the $k-\epsilon$, or the $k-\omega$ *models*, which use two additional equations and scalar quantities for approximating the Reynolds stress tensor. More accurate approaches, like the *Reynolds stress model* (RSM), compute the Reynolds stresses directly. In the case of wall-bounded flows the boundary layer is typically very thin and would require a comparably high grid resolution, even with RANS. Hence, analytic *wall functions*, like the *law of the wall*, are often used together with turbulence models in order to model the characteristic near-wall velocity distribution. Chapter 9 investigates how wall functions can be harnessed to improve the interpolation schemes used on the visualization side, and make them more consistent with the employed simulation model.

Large eddy simulation (LES), a third solution approach, resides between RANS and DNS with respect to its accuracy and computational costs. In contrast to RANS, a fully time-dependent flow is computed. The smallest length scales, however, are not resolved, like DNS does. LES removes the small scales from the velocity field by applying a low pass filter to the Navier-Stokes equations. Hence, similar to RANS, the sub-filter scales have to be modeled, e.g., with the Smagorinsky model. This makes LES computation amenable to high Reynolds number flows.

The finite-volume method (Chapter 4.3) is the most popular discretization scheme for CFD applications, as the conservation of field quantities is automatically satisfied. Finite element methods (Chapter 4.3) are also used in CFD. However, making such schemes conservative requires additional effort. The handling of discontinuous phenomena, like shocks, is also more complicated with the continuous Galerkin FEM approach. Discontinuous Galerkin (DG) methods are becoming more popular, as they harness features of both finite volume and finite element methods (see Chapter 6, Part II).

Data Sources

Wall-bounded Flows

Flow past an obstacle or near walls can be used to study the boundary layer, flow separation, and the development of turbulence. Interactive higher-order visualization techniques are developed in this thesis (Chapter 8) for data from explicit discontinuous Galerkin simulations of arbitrary order [68] (Chapter 6). Laminar to turbulent transitions computed with DNS [11] are visualized with feature-based techniques

[1] Spatial averaging or ensemble averaging can be used in the case of statistically non-stationary flows.

near vortex cores (Chapter 11). Our novel approach on time-dependent vector field topology is applied to unsteady flows, including flow around an obstacle computed with a simulation using turbulence-models (Chapter 16). Wall-bounded flows from a RANS computation with wall function models are investigated in the discussion of how subgrid-scale models can help to improve visualization (Chapter 9).

Free Turbulent Shear Flows

In the absence of walls turbulence is generated in flows with strong velocity gradients, e.g., due to an injection of a jet of air or due to heating. Breakup at free surfaces of two-phase injection problems computed with DNS [163] are investigated with feature-based visualization techniques in Chapter 11. Topology-based methods are applied to two-phase flow of air around droplets from DNS [168] and unsteady buoyant flow computed with CFX and turbulence models (Chapter 16).

Flow in Porous Media and Fracture Mechanics

Flow in porous media takes place at small Re. A simple porous two-phase flow is computed on the GPU complemented with simultaneous in-situ visualization (Chapter 17). Non-flow data from a particle partition of unity (PPUM) fracture mechanics computation [171] (Chapter 6) yielding 2D vector field data is visualized with a novel GPU-based visualization technique (Chapter 7).

Chapter 5

Gaining Insight Through Visualization

This chapter introduces basic visualization techniques and concepts the presented methods build upon. The focus lies on the visualization of 3D fields, if not stated otherwise. Indirect volume visualization (Section 5.1) provides techniques for a geometry-based visualization of scalar fields, and appropriate lighting models are used to improve depth perception (Section 5.2). A volumetric representation of scalar fields can be generated with direct volume rendering, which is based on a physical model of light transport in a participating medium (Section 5.3). Moreover, integral curves and surfaces are introduced as the basic building block (Section 5.4) of more complex vector field visualization methods, like feature-based techniques (Part III) and vector field topology (Part IV).

5.1 Indirect Volume Visualization

Indirect volume visualization extracts and renders geometric representations of isosurfaces, which are defined as manifolds of constant scalar value within a given 3D scalar field. Such surfaces often represent relevant features like tissue layers in medicine or material interfaces in fluid flows. Efficient algorithms have been devised for solving the isocontour root finding problem locally per grid cell. The *marching cubes* [114] and the *marching tetrahedra* are popular algorithms that extract geometric isosurface representations from data given on regular grids and unstructured tetrahedral grids, respectively. Both algorithms combine linear approximation within the cells with lookup tables for a fast triangulation of the surface mesh. Parallelization is straightforward and further acceleration can be achieved with domain decomposition approaches based on octrees [114, 212], or with non spatial range-query trees [34].

Isosurface visualization with ray casting [146] offers a pixel-accurate alternative to the indirect mesh-based techniques (Section 5.3). An intermediate geometric representation

is no longer required, and visualizing multiple isosurfaces in a semi-transparent manner can be faster because the costly visibility sorting of triangle meshes is avoided. Opaque rendering of surface meshes, however, is still very efficient with the traditional OpenGL rasterization pipeline.

5.2 Illumination and Light Transport

Applying plausible lighting models plays an important role in scientific visualization because it adds crucial shape and depth information to the rendered scene. *Local lighting* based on the Blinn-Phong [19] shading model is a common choice in scientific visualization, which typically does not strive for photorealism. These models basically calculate a single reflection of the radiance incident at a rendered surface point from the light sources. In the case of isosurfaces, the gradient of the scalar field can be used as surface normal. Line-type structures, like streamlines, do not have a distinguished normal and require adapted models [223]. *Global illumination*, e.g., with ray tracing, is much more expensive but can result in photorealistic images by including indirect lighting effects. Light transport in amorphous three-dimensional phenomena like fluids is discussed in the following.

5.3 Direct Volume Visualization

Visualization of 3D fields inherently has to deal with occlusion issues due to the mapping from the 3D field to a 2D image representation. *Direct volume rendering* (DVR) enables a direct dense visualization by rendering the volumetric field in a semi-transparent manner. This can reduce occlusion issues of indirect geometry-based techniques. *Transfer functions* provide an intuitive interface for the desired exploration task. Interesting features, for example, can be highlighted by mapping them to opaque color and choosing a higher transparency for their spatial context. DVR is based on the volume rendering integral with its underlying physical models of light transport. Volume rendering is a global problem, which makes it computationally very expensive. While an accurate solution of the problem is often not possible, a multitude of methods and optimizations have been developed that allow for interactive volume rendering on modern compute hardware. The end of this section gives a short overview of existing methods with a focus on ray casting.

Light Transport in Participating Media

The transport of energy through space by means of electromagnetic waves or particles can be measured in terms of radiance. The SI unit of radiance $W/sr \cdot m^2$ indicates that it is a measure defined with respect to the direction of incidence or exitance of the flux (given by a solid angle, sr), and the projected unit area (m^2) at a corresponding point on a reference surface. Radiance is conserved along unobstructed light rays, accounting

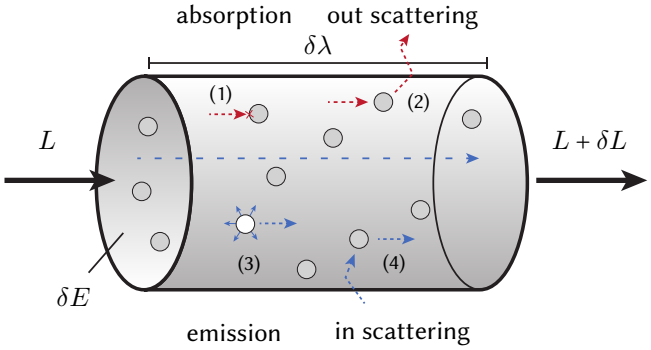


Figure 5.1 — Transfer of incident light L through a participating medium modeled as a bulk of particles that interact with the photons (1)–(4).

for the undisturbed propagation in vacuum. This simplifies the study of light transport, because only rays interacting with matter, like solid surfaces or traversed fluids, have to be analyzed further. For the derivation of the volume rendering equation, the participating media is modeled as an accumulation of small particles [119, 155]. For the sake of simplicity, wavelength dependent effects are ignored in the following. A slab of a participating medium of volume $\delta V = \delta\lambda\delta E$ is shown in Figure 5.1. The figure illustrates how a parallel beam of incident radiance L is affected by the medium. Photons hitting particles might be absorbed (1) or scattered out of the slab (2), leading to extinction of radiance, i.e., negative δL. On the other hand, an increase of radiance is induced either by particles that convert other types of energy into light energy (3), called emission, or by in-scattering (4). In the following, scattering effects are neglected, and the particle sizes σ as well as the particle density ϱ are assumed to be relatively small. With these assumptions, the chance of a photon of being absorbed by a particle can be directly related to the fraction of the slab cross section that is covered by particles $\delta E_f = \frac{\sigma\varrho\delta V}{\delta E}$. The limit case $\delta\lambda \to 0$, where the slab becomes infinitesimally thin, leads to the ODE of absorption

$$\frac{dL}{d\lambda} = -\kappa(\lambda)L(\lambda), \tag{5.1}$$

with the material dependent *extinction coefficient* $\kappa = \sigma\varrho$. The solution to this equation is known as the *Beer-Lambert law* in optics. For a given ray segment $[\lambda_0, \lambda_1]$, it can be stated as $L(\lambda_1) = L_0 T(\lambda_0, \lambda_1)$, with the *optical transparency* of the participating medium

$$T(\lambda_0, \lambda_1) = e^{-\int_{\lambda_0}^{\lambda_1} \kappa(\lambda')d\lambda'} \tag{5.2}$$

describing the fraction of the incident radiance L_0 that is absorbed along the ray segment. According to Planck's law, the increase of radiance due to emission in a local thermodynamic equilibrium is related linearly to the particle density. Thus, the additional flux

$$g(\lambda) = c(\lambda)\kappa(\lambda) \tag{5.3}$$

can be modeled as the product of a material dependent emission coefficient c times the extinction coefficient κ, which is proportional to the particle density, leading to the

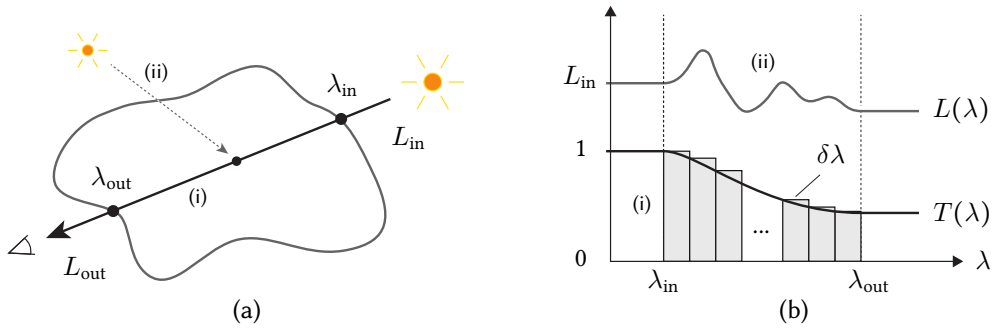


Figure 5.2 — (a) The emission absorption model of light transport (5.4) only depends on optical properties observed along a line of sight (i). Local shading is used to add perceptual cues (ii). (b) Riemann sum approximation of the optical transparency T. While T is a continuously decreasing function (i), luminance L, in general, is not (ii).

more realistic light transport model

$$\frac{dL}{d\lambda} = g(\lambda) - \kappa(\lambda)L(\lambda), \tag{5.4}$$

accounting for both emission and absorption effects.

The Volume Rendering Integral

Direct volume rendering uses the emission absorption model (5.4) to simulate the transport of light through a 3D scalar field to the viewer. The volume rendering integral

$$L(\lambda) = L_0 T(\lambda_0, \lambda) + \int_{\lambda_0}^{\lambda} g(\lambda')T(\lambda', \lambda)d\lambda' \tag{5.5}$$

is obtained as the analytical solution [119] to the ODE model (5.4), with T defined in (5.2). The volume rendering integral computes the amount of radiance that arrives at the viewer along a given line of sight (Figure 5.2 (a)). It solely depends on the incident background radiance L_0 and the optical properties of the participating medium observed along the corresponding ray.

The values of the scalar field $s(\mathbf{x})$ have to be mapped to optical properties. In the simplest case, a density field can be mapped directly to the extinction coefficient, and the emission coefficient can be set to zero, resulting in X-ray like images. Using application dependent *transfer functions* however provides much more flexibility, allowing the user to highlight the features of interest. In the case of medical visualization, the user is often interested in classifying the volume into different regions, for example, by using different colors for tissue and the bone structures. *Post classification* applies transfer functions after reconstructing the continuous field. For isotropic materials,

two mappings $\tilde{\kappa}, \tilde{c} : R \to R$ from scalar values s to extinction and emission coefficients are sufficient to establish the optical property fields

$$\kappa(\lambda) = \tilde{\kappa}(s(\mathbf{r}(\lambda))) \tag{5.6}$$

$$g(\lambda) = \big(\tilde{c}(s(\mathbf{r}(\lambda))) + r(s(\mathbf{r}(\lambda)))\big)\tilde{\kappa}(\mathbf{r}(\lambda)). \tag{5.7}$$

Multidimensional transfer functions often additionally depend on the gradient and the Hessian of the scalar field s, e.g., to highlight material boundaries [96]. The source term g in (5.7) is based on the aforementioned assumption that the radiance of the emitted light is proportional to κ or the density of the medium, see (5.3). It is also common to add a contribution r to the source term to simulate the reflection of radiance coming from extra light sources, as shown in Figure 5.2 (a)(ii). Local lighting models are a cost-effective way for computing r because attenuation is ignored. Typically, the gradient of the scalar field is used as the surface normal in the Blinn-Phong shading model, which has ambient, diffuse, and specular color contributions [107]. To account for the attenuation of the extra radiance, a secondary shadow ray has to be shot through the participating medium. This leads to more realistic images with volumetric shadowing, but also heavily increases the computational complexity. A technique providing a fast approximation to these effects is presented in [97].

Discretized Volume Rendering Integral

Analytical closed form solutions of the volume rendering integral are only available for few special cases. One example is the visualization of piecewise linear scalar fields with piecewise linear transfer functions [213]. The combination of general fields and nonlinear transfer functions, however, requires a numerical approximation of the integral (5.5), e.g., with quadrature rules. While Riemann sums are restricted in their accuracy, they are still a popular choice, as they result in efficient iterative evaluation schemes.

The rendering integral consists of three integrals that have to be approximated. Figure 5.2 (b) illustrates the discretization of the optical transparency (5.2) using a left *Riemann sum*

$$T(\lambda_0, \lambda_1) \approx e^{-\sum_{k=0}^{n-1} \kappa(\lambda_k)\delta\lambda} = \prod_{k=0}^{n-1} e^{-\kappa(\lambda_k)\delta\lambda} \tag{5.8}$$

with n intervals of constant size $\delta\lambda$ and the ray parameter $\lambda_k = \lambda_0 + k\delta\lambda$. The accumulated transparency $t_i = e^{-\kappa(\lambda_k)\delta\lambda}$ of slab k is used in the following, and $c(\lambda_k)$ as well as $\kappa(\lambda_k)$ are abbreviated with c_k, and κ_k, respectively. With these definitions the *discrete volume rendering integral*

$$L(\lambda_t) \approx L_0 \prod_{k=0}^{n-1} t_k + \sum_{k=0}^{n-1} c_k \kappa_k \delta\lambda \prod_{i=k+1}^{n-1} t_k, \tag{5.9}$$

is obtained from (5.5).[1] The volume rendering methods discussed later are based on a variant of (5.9) that involves an approximation of the *opacity*

$$\alpha_i \approx 1 - t_k \approx -\kappa_k \delta\lambda \tag{5.10}$$

of slab k. This results in the opacity formulation of the discrete volume rendering integral that can be evaluated more efficiently

$$\begin{aligned} L(\lambda_t) &\approx L_0 \prod_{k=0}^{n-1}(1-\alpha_k) + \sum_{k=0}^{n-1} c_k \alpha_k \prod_{i=k+1}^{n-1}(1-\alpha_i) \\ &= \alpha_{n-1}c_{n-1} + (1-\alpha_{n-1})\Big(\alpha_{n-2}c_{n-2} + (1-\alpha_{n-2})\Big(\alpha_{n-3}c_{n-3} + \\ &\qquad \cdots + (1-\alpha_1)\Big(\alpha_0 c_0 + (1-\alpha_0)L_0\Big)\Big)\ldots\Big). \end{aligned} \tag{5.11}$$

The expanded equation directly translates into the iterative *back-to-front compositing* scheme

$$L_{i+1} = \alpha_i c_i + (1-\alpha_i)L_i \tag{5.12}$$

with an evaluation order that corresponds to the natural direction of light propagation. Alternatively, the same results can be obtained with *front-to-back compositing*

$$L_{i-1} = (1-\tilde{\alpha}_i)\alpha_i c_i + L_i \tag{5.13}$$

$$\tilde{\alpha}_{i-1} = (1-\tilde{\alpha}_i)\alpha_i + \tilde{\alpha}_i. \tag{5.14}$$

Sampling Errors and Optimizations

The left Riemann sum (5.8) converges against the accurate solution in the limit case $\delta\lambda \to 0$. Estimates of the upper error limit depend on the maximum gradient of the approximated function and the size of the intervals. To obtain exact results the step size has to be chosen such that the sampling frequency is equal or larger than the Nyquist frequency of the sampled signal. This frequency does not only depend on the scalar field, but also on the product of the scalar field's Nyquist frequency ν_s and the Nyquist frequency of the transfer function ν_{TF} [17]. Consequently, complex nonlinear transfer functions can easily require very small step sizes, even in the case of linear scalar fields.

Sampling the whole volume with a small fixed step size often oversamples large parts of the volume, which is not acceptable due the immense costs—in particular if interactive display rates are desired. *Locally adaptive sampling* strategies provide solutions to the oversampling dilemma. A special case of adaptive sampling is offered by *empty space skipping*, which detects and skips regions of the volume that do not contribute to the integral, i.e., regions that are completely transparent [218]. Using adaptive sampling in the whole domain is even more effective [39]. A conservative approximation of the required sampling frequency for volume rendering with arbitrary transfer functions

[1] This formulation does not include external light sources, i.e., the term r in (5.7) is omitted.

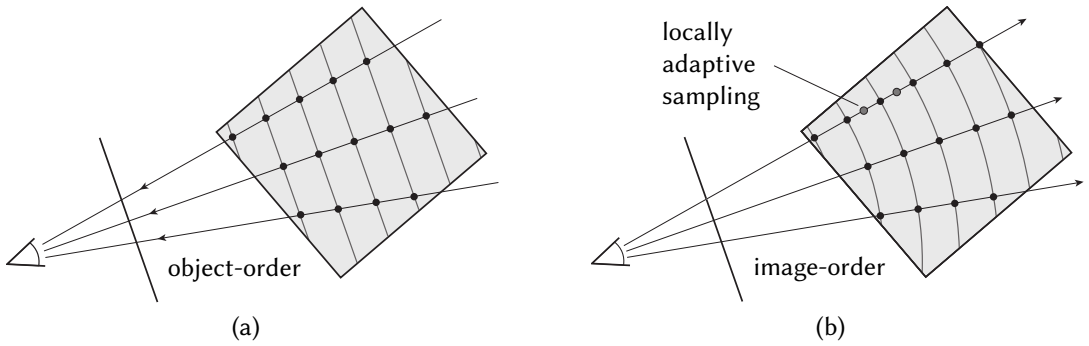


Figure 5.3 — (a) Slice-based volume rendering (object-order algorithm) projects the sampled volume slices to the image plane. (b) Ray casting (image-order algorithm) accumulates radiance on a per pixel basis—globally constant step sizes in perspective projections, as well as locally adaptive sampling, are supported naturally.

is given by Bergner at al. [17]. This is further discussed in Section 8.2.4. The opacity transfer function $\alpha : R \to R$ involved in the compositing formulas (5.12) and (5.13) is defined with respect to a fixed step size $\delta\lambda$, and thus cannot directly be applied to adaptive sampling. *Opacity correction*

$$\alpha' = 1 - e^{-\kappa\delta\lambda'} = 1 - \left[e^{-\kappa\delta\lambda}\right]^{\frac{\delta\lambda'}{\delta\lambda}} = 1 - \left[1-\alpha\right]^{\frac{\delta\lambda'}{\delta\lambda}} \tag{5.15}$$

is based on (5.10) and ensures correct transparencies for slabs of width $\delta\lambda' \neq \delta\lambda$. The problematic multiplication of the Nyquist frequencies in the signal $c(s(\lambda))\alpha(s(\lambda))$ can be avoided if a piecewise linear behavior of the scalar field is assumed within slabs. The sampling rate required for computing the contribution within a slab now solely depends on ν_{TF}. This is exploited by *pre-integrated volume rendering*, which precomputes the contributions of all possible slab configurations [45]. This allows for larger step sizes resulting in high quality images at interactive display rates. *Early ray termination* is an optimization that stops the costly evaluation of the integral as soon as the accumulated opacity approaches one. It can be used together with front-to-back compositing schemes (5.13).

Direct Volume Rendering Methods

Volume rendering looks back on a long tradition in scientific visualization. A multitude of methods have been devised in the last decades, and there is a common agreement that volume exploration significantly benefits from interactive display rates. This is reflected by the methods and their sophisticated ways of evaluating the discretized volume rendering integral, often by harnessing specific parallel hardware architectures available at their time. During the last decade, GPU-based implementations became very popular due to the increased flexibility with respect to their programmability and their good price-performance ratio. Unfortunately, volumetric primitives cannot be

rendered directly with the rasterization pipeline offered by the GPU. Thus, early GPU methods where based on the rasterization of 2D primitives and benefited from the fast 2D hardware texturing and compositing capabilities. Nowadays, with general-purpose GPU programming languages being available (Section 2.3), more flexible and powerful volume rendering techniques based on ray casting are possible.

The different methods can be classified into three major classes. *Domain-based methods* solve the problem in a non-spatial domain, e.g., the frequency or wavelet domain. They are not discussed any further in this work. *Object-order* techniques project the radiance emitted by the volumetric elements from the object space onto the image plane (Figure 5.3 (a)). *Image-order* techniques, in contrast, gather the radiance on a pixel basis, as illustrated in Figure 5.3 (b).

Object-order techniques require a sorting of the volume elements prior to projecting them to the image plane to guarantee a correct accumulation of radiance according to the compositing equations (5.12) or (5.13). Examples are *splatting* [208] and *slice-based volume rendering* for regular grids [28, 38], which allow for efficient GPU implementations. Some unstructured grid types can be handled with *cell projection* techniques, like projected tetrahedra for tetrahedral grids [179]. Visibility cycles, however, can prevent a correct visibility sorting [99].

Object-order techniques have their amenities, but they are restricted to particular grid types, and they may fail to deliver high quality results in certain cases. *Image-order* algorithms based on *ray casting* [42, 107] account for these limitations by offering a more natural way of evaluating the volume rendering integral. The volume is sampled along rays shot through the pixels of the image plane, e.g., in front-to-back order (5.13), avoiding the costly sorting step, see Figure 5.3 (b). Ray casting is also a more general approach in the sense that it works with arbitrary field representations if efficient point location [64] and appropriate interpolation algorithms are provided, e.g., for volume rendering of unstructured grids [62, 119, 213]. More general light transport models can be integrated by tracing along secondary rays. Early ray termination, adaptive sampling, and parallelization are also straightforward to implement, because the rays can be treated independently from each other. Still, for large complex volumes ray casting is computationally very demanding due to the large number of samples that have to be evaluated. GPU ray casters typically harness the trilinear interpolation functionality of the rasterization pipeline to enable interactive visualization of data on structured grids [102, 180]. A more extensive discussion of ray casting of unstructured grids and a framework for interactive high quality rendering of higher-order finite elements are given in Part II.

5.4 Geometric Flow Visualization

The Navier-Stokes equations (Section 4.2) represent the motion of a fluid with a vector field $\mathbf{v}(\mathbf{x}, t)$, encoding the direction and the velocity of the flow within the domain.

The volume visualization methods of the previous section can be used to visualize the magnitude of velocity, but this fails to reveal the complete structure of the flow dynamics. A simple example of *direct flow visualization* techniques are methods that plot arrows at a large number of locations within the domain, e.g., at the grid points, resulting in so-called hedgehog plots. Such visualizations enable a local analysis of the flow. In order to understand the global structure of the flow, as depicted in Figure 4.4, a viewer of such visualizations is forced to trace a particle along the vectors in his mind—a demanding task that typically does not allow for accurate interpretations. *Geometric flow visualization* [122] takes this burden from the user by computing geometries that are tangential to the vector field, like streamlines or path lines, which represent the trajectories of massless particles. This corresponds to the *Lagrangian* point of view of a vector field, in contrast to the *Eulerian* view described above, which observes fluid flow at fixed locations. Lagrangian methods also play an important role in the extraction of global flow features, a topic of Part III. Further insight into flow structure is provided by the topological methods discussed in Part IV. The topological flow skeleton, for example, consists of a distinguished set of integral lines or surfaces, providing an abstraction that can result in visualizations similar to the hand-drawn images in Figure 4.4.

Integral Curves and Surfaces

The trajectory $\mathbf{c}(t)$ of a massless particle is obtained as the solution of the IVP of an ODE

$$\frac{d\mathbf{c}(t)}{dt} = \mathbf{u}(\mathbf{c}(t), t), \qquad \mathbf{c}(t_0) = \mathbf{x}_0, \tag{5.16}$$

with an initial seed location $\mathbf{x}_0$ specified at time t_0. The ODE enforces the parametric curve $\mathbf{c}(t)$ to stay tangential to the vector field $\mathbf{u}$ at all of its points. In order to obtain $\mathbf{c}(t)$, (5.16) has to be integrated—thus, these curves are often referred to as *integral curves*. An analytical solution is available only in very simple scenarios where the vector field is given in an analytical representation. In the case of discretized fields, like from CFD simulations, numerical integration schemes, as described in Section 3.5, have to be employed. The resulting curve is called *path line* if the vector field is time-dependent, and in the special case of a steady vector field, where $\mathbf{u}(\mathbf{x}, t) = \mathbf{u}(\mathbf{x})$, it is called *streamline*. It is noteworthy that the streamlines of a C^1 continuous vector field do not intersect. Often, concepts from dynamical systems theory are used for classifying trajectories according to their characteristics. Trajectories that return to one of their earlier locations are called *periodic orbits*—they capture periodic field behavior. *Stationary points or critical points* are defined as trajectories that are degenerated to a point. Such special cases of trajectories appear less often, but they provide the basis for extracting the global structure or topology of the vector field (see Chapter 14).

A *streak line* is defined as the curve that connects all advected particles that have been released continuously at a seed location over a given time period. In real world flow, streak lines are made visible by injecting dye at a fixed location into the field. They are equivalent to streamlines in steady vector fields. The streak line concept can be generalized by letting the seed point move along a curve while releasing particles, resulting in *generalized streak lines* [209]. Instead of releasing particles at a single point, manifolds of higher dimension can also serve as seeding constructs. In the case of seed lines, the advected curves are called *time lines* and reflect the underlying characteristic of the flow through their deformation. Using seeding curves in 3D vector fields results in *stream surfaces*, *path surfaces*, or *streak surfaces*, respectively.

Efficient Computation

Computing integral curves and surfaces in large time-varying flow data sets is challenging if interactivity is a requirement. As independent particles can be advected independently from each other, GPUs are very well suited for tracing a large number of trajectories in parallel, in particular if regular grids are used [27]. In Chapter 12 we present a method using geometry shaders [195]. Particle tracing in unstructured grids is more intricate and time consuming due to the point location issue and missing hardware interpolation support. Many techniques, including ours, rely on explicit solvers based on RK4 integration schemes. They are fast and offer a good approximation quality, and, if necessary, adaptive time-stepping can be employed [183]. Acceleration with an hierarchical approach is also possible [81]. Exact analytical solutions are available in the special case of linear tetrahedral grids [94, 137]. Garth et al. present techniques for accurate generation of path and stream surfaces [65, 66]. The robust computation of streak surfaces was also subject to intense research in recent years, due to its intricacy. Krishnan et al. [100] present adaptive techniques for computing time surfaces and streak surfaces in large data. Interactive computation of streak surfaces on GPUs is discussed by Bürger et al. [26]. Recently, methods for computing traditional streak lines as integral curves inside a derived vector field have been proposed [203].

Seeding and Placement

The selection of appropriate seed points or curves and advection times is crucial to obtain significant visualizations. In the case of time-dependent fields, the user additionally has to choose a starting point in time. Interactive placement is one possibility, offering an important exploration tool for the experienced user, e.g., in virtual environments [24]. This approach, however, is prone to missing important structures. Figure 4.4 shows a hand drawn image of flow around a cylinder depicting the characteristic flow behavior. Algorithms have been developed that compute a uniform distribution of streamlines to mimic such visualizations. However, even for stationary 2D flow, this is a difficult task [192]. Adding time dependency makes the problem even more complicated, due to the possibility of crossing lines. *Dense texture-*

based flow visualization [105], like line integral convolution (LIC), offers an alternative approach to geometric flow visualization. It avoids the seeding problem by providing a dense visualization covering the whole domain. These techniques work well in the case of 2D fields [29] and on curved surfaces embedded in 3D fields [207], but suffer from the same perception issues as geometric flow visualization when applied to the whole 3D domain [48]. Moreover, LIC on curved surfaces is only able to visualize the tangential component of the vector field. In 3D domains, the selection of meaningful seed structures becomes even more critical. While occlusion and clutter are problematic if the coverage is chosen too dense, important features might be missed in the contrary case. Techniques that account for these issues, e.g., by steering the seeding with feature regions contained in Chapter 11 of Part III. Inspired by the more holistic approach of dynamical system theory we also present our research on time-dependent vector field topology, using distinguished manifolds for streak surfaces seeding (Part IV).

Part II

Direct Visualization of Higher-Order Fields

Chapter 6

Introduction to Higher-Order Fields

Higher-order fields belong to a new type of data that employs piecewise continuous analytical functions to represent the variation of the field variables. Such representations offer a higher flexibility with respect to the regularity of the field solution and its derivatives compared to traditional sampled field data, which requires interpolation (Chapter 3). On the other hand, additional complexity is introduced. Traditional visualization techniques cannot handle higher-order data directly. Hence, new sophisticated techniques are required to enable an efficient and accurate visual analysis of this data. This chapter introduces the representation of higher-order field data, the sources of this class of data—in particular numerical simulation, and the challenges it poses to interactive visualization. This serves as the basis for the description of our new visualization techniques for higher-order fields of arbitrary order in Chapter 7 and 8.[1]

6.1 Numerical Simulation and Data Representation

Traditional FEM techniques of low order have to use grid refinement (*h-refinement*) in order to improve their approximation space. The influence of the grid on the associated approximation space is illustrated in Figure 6.1. In this example, the approximation space $V_{h/2}$ obtained after grid refinement (dashed region) enables a better approximation of the true solution, when compared to the original approximation space V_h. Here, the approximation spaces are embedded within the Sobolev space $H^1(\Omega)$ of "once differentiable" functions in the domain Ω. Some higher-order finite element methods additionally allow for increasing the polynomial order of their approximation (*p-refinement*). Schemes that harness both grid and polynomial refinement are increasingly used in modern numerical solvers. One example are hp-adaptive discontinuous Galerkin (DG) methods. Section 6.1.1 shortly summarizes the properties of the DG

[1] Parts of this chapter have previously been published in Sadlo et al. [160], Üffinger et al. [194], and Üffinger et al. [199].

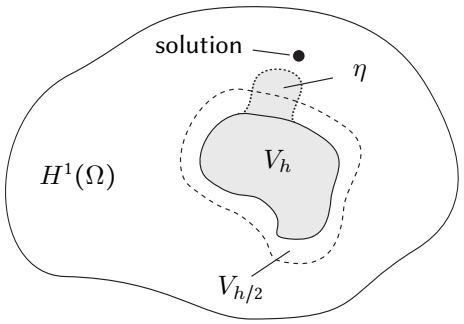


Figure 6.1 — A FEM approximation space V_h can be improved by increasing the polynomial order or by grid refinement, e.g., leading to $V_{h/2}$. Algebraic refinement with enrichment functions η can provide a more stable and efficient alternative—harnessing a priori knowledge of the solution.

formulation. For an extensive introduction to DG methods, the reader is referred to the book of Cockburn et al. [35]. DG methods offer a great flexibility. They are able to handle complex domain geometries, allow for an easy integration of hp-refinement, and support high polynomial orders. Moreover, they enable efficient parallel solvers on compute clusters and can be applied to many applications, ranging from hydrodynamics and aeroacoustics to magnetohydrodynamics [35]. Phenomena with steep gradients or discontinuities, like cracks, however, are still difficult to handle with techniques restricted to h- and p-refinement. Generalized FEM methods (Section 6.1.2) overcome this limitation with a so-called *algebraic refinement* approach. They harness problem-dependent a priori knowledge of the field solution. The benefit of this approach is illustrated in Figure 6.1. The a priori knowledge allows for a more goal-oriented refinement of V_h toward the true solution, for example, by adding the subspace $\eta \subset H^1$ in the example. Additionally, algebraic refinement can be combined with traditional hp-refinement techniques.

6.1.1 Discontinuous Galerkin Method

Discontinuous Galerkin (DG) methods [35] borrow ideas from both FEM and FVM schemes. Similar to FEM (Section 4.3) the DG approach is based on the weak formulation (4.4) of the conservation laws, using a finite-dimensional function space. The equation, however, is not enforced on the whole domain Ω, but only locally on the elements, for example, on the cells of an unstructured grid. This approach introduces discontinuities into the field solution at inter-element boundaries, similar to the FVM (Section 4.3), but allows to use polynomial basis functions of very high order, similar to FEM. Moreover, using local ansatz functions, it is easier to construct an orthogonal basis of high order with DG. The polynomial order of traditional continuous Galerkin FEM, in contrast, is comparably limited in practice because of a deteriorating matrix condition when increasing the order. The local polynomial functions within the cells are coupled amongst each other through the fluxes across cell boundaries. This makes the DG method locally conservative, like the FVM. Using a local polynomial ansatz in each element yields matrices with a block structure. This alleviates a parallel solution on massively parallel architectures like GPUs [95]. Using polynomial field approximations of high order has the advantage that a comparable accuracy to low-order

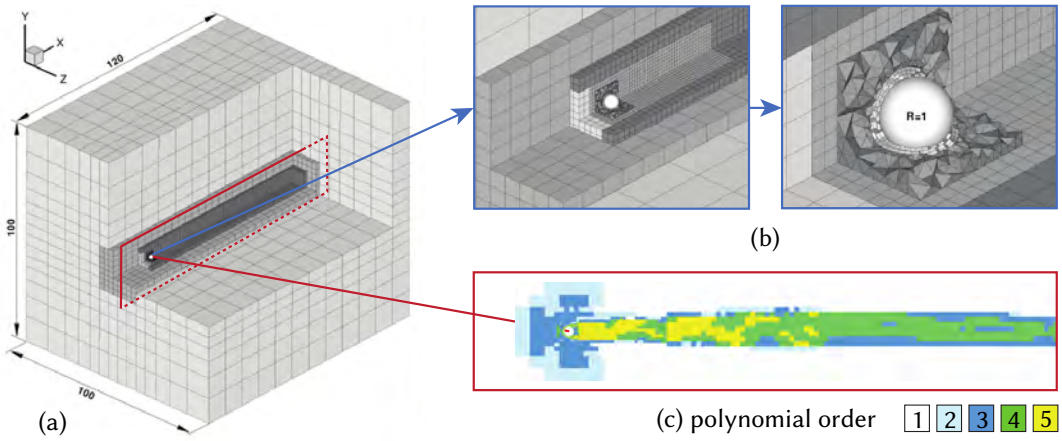


Figure 6.2 — (a) A non-conforming mesh, discretizing the domain of a simulation of fluid flow past a sphere. (b) Close-ups of the sphere. (c) The adaptive degree of the polynomial solution is depicted on a planar cut marked in the 3D illustration.

simulation methods can be achieved with significantly fewer grid cells. With respect to time-stepping, DG also provides low dispersion and dissipation errors. Moreover, by allowing discontinuities at cell boundaries, mesh generation is simplified, e.g., non-conforming topologies can be handled more easily.

Our DG data is generated by the *space-time expansion Discontinuous Galerkin* approach presented in [67, 68, 87]. The complexity of this data is illustrated in Figure 6.2. Hp-adaptivity is supported, allowing for unstructured non-conforming grids with varying cell sizes and varying polynomial degree p. Additionally, curved elements are used to improve the approximation of curved boundary geometry, e.g., of the sphere obstacle. The scalar field solution is represented by piecewise polynomial solutions of high order with discontinuities at the cell boundaries. Similar to traditional FEM the local solution of a cell can be written as

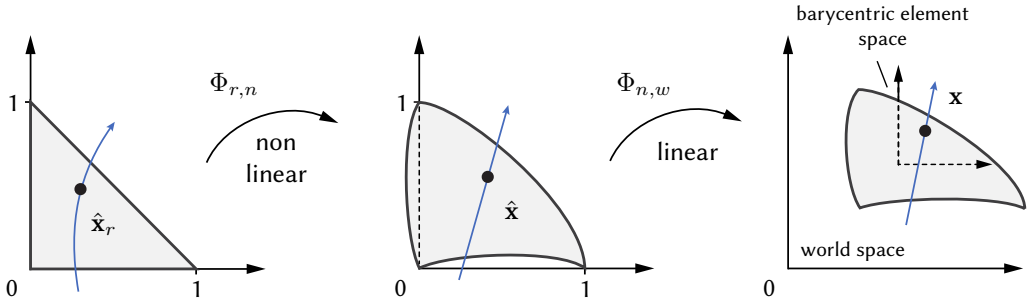
$$u(\mathbf{x}) = \sum_{i=1}^{N} c_i \psi_i^{(p)}(\mathbf{x}), \tag{6.1}$$

with $\{\psi_i^{(p)}\}$ being a basis of the space of polynomials up to degree p and c_i being the corresponding coefficients. As shown in Figure 6.3, the space-time DG method uses two geometry mappings, one from the reference element to the normalized element, followed by a mapping to physical space.[2] To increase the efficiency of the scheme, the solution—denoted as $\hat{u}$ in the following—is represented in the normalized element (Figure 6.3 (b)), using an orthonormal basis. The orthonormal basis $\tilde{\psi}_i$ is constructed from a monomial basis

$$\{b_i^{(p)}\} = \{x^k y^l z^m | k, l, m \in \mathbb{N}_0, k + l + m \leq p\}, \tag{6.2}$$

with $b_i^{(p)}$ given in order of increasing degree.

[2] The reference space approach with two mappings is described in more detail in [87].



(a) reference element (b) normalized element (c) physical element

Figure 6.3 — The element spaces used by the space-time DG simulation [87]. The normalized element (middle) is mapped to its world space geometry (right) by the linear mapping $\Phi_{n,w}$. The mapping $\Phi_{r,n}$ (left) is nonlinear for curved elements, i.e., straight lines (blue) in physical space (right) correspond to curved lines in the reference element.

Orthonormalization of the monomial basis is carried out with the Gram-Schmidt algorithm

$$\begin{aligned} \xi_i(\mathbf{x}) &= b_i^{(p)}(\mathbf{x}) - \sum_{j=0}^{i-1} \langle b_i^{(p)}(\mathbf{x}), \tilde{\psi}_j(\mathbf{x}) \rangle \tilde{\psi}_j(\mathbf{x}) \\ \tilde{\psi}_i(\mathbf{x}) &= \frac{\xi_i(\mathbf{x})}{\sqrt{\langle \xi_i(\mathbf{x}), \xi_i(\mathbf{x}) \rangle}}, \end{aligned} \tag{6.3}$$

where ξ_i denotes an intermediate state and $\tilde{\psi}_i$ the orthonormalized basis polynomials. Orthonormality of two basis functions is defined as $\langle \varphi_i, \varphi_j \rangle = \delta_{i,j}$, involving the L_2 inner product $\langle \varphi_i, \varphi_j \rangle := \int_{\Omega_c} \varphi_i(\mathbf{x}) \varphi_j(\mathbf{x}) d\mathbf{x}$. Hence, in contrast to the monomial basis, the structure of the orthonormal basis depends on the geometrical area Ω_c of the cell. As depicted in Figure 6.3 (b), the normalized elements are allowed to have curved boundaries. An affine transformation

$$\Phi_{n,w}(\hat{\mathbf{x}}) = A\hat{\mathbf{x}} + \mathbf{t}_w = \mathbf{x} \tag{6.4}$$

maps their geometry to their world space counterpart (see Figure 6.3 (c)). Hence, for visualization, this mapping has to be used to transform the normalized element space solution of the simulation $\hat{u}$ to world space.

6.1.2 Generalized Finite Element Methods

The popular classical finite element method (Section 4.3) and the related *hp*-adaptive DG method, covered in the previous section, employ grid-based domain discretization combined with piecewise polynomial ansatz functions. This makes them suitable for physical phenomena with regular field solutions, i.e., functions with certain differentiability properties. Problems with microstructure, discontinuities, singularities, or other non-smooth properties, however, can often not be handled properly with

these techniques. The classical method, for example, needs to align the mesh to the discontinuities to avoid instabilities and reduced convergence rates in the solver. This is a complex task, especially if re-meshing is required during the simulation to capture moving discontinuities. Singularities on the other hand induce very high gradients, which require an appropriate refinement of the mesh. This additionally hinders an efficient solution of the problem.

Generalized finite element methods, a field of active research, account for the limitations of the classical finite element method [43, 58, 173]. Early approaches removed the restriction to piecewise polynomial ansatz functions [9, 15] by utilizing *problem-dependent enrichment functions* to model characteristic non-smooth behavior independently from the underlying grid. The enrichment approach requires a partition of unity (PU) $\sum_{i=1}^{N} \varphi_i \equiv 1$. As the PU property is satisfied by P_1-FEM basis functions, classical FEM shape functions ψ_i can be used by the extended finite element method (XFEM) [125] and the generalized finite element method (GFEM) [182]. The X/GFEM approach

$$u(\mathbf{x}) = \sum_{i=1}^{N} c_i \psi_i(\mathbf{x}) + \sum_{m \in I \subset \{1,\dots,N\}} d_m \psi_m \eta(\mathbf{x}), \quad \mathbf{x} \in \Omega, \tag{6.5}$$

extends the traditional FEM ansatz (4.14) by adding problem dependent enrichment functions $\eta(\mathbf{x})$. The influence of the enrichment is typically restricted to a subset of nodes $I \subset \{1, \dots, N\}$. Correct blending of η with the traditional ansatz at the boundary of I is ensured by the PU.[3] Additionally, u is able to accurately represent η within the enriched region, a feature guaranteed by the PU property of the ψ_i. The computation of the solution follows the same steps as the traditional FEM. The global matrices are assembled first. Afterward, the system is solved for the unknowns c_i and d_i in a straightforward manner. Compared to hp-refinement, the *algebraic refinement* of the X/GFEM represents a more efficient approach for improving the approximation space, as depicted in Figure 6.1. It harnesses a priori knowledge about the characteristic solution behavior expressed in terms of the enrichment functions. Thus, instead of adapting the mesh near a moving discontinuity, a simple static grid in combination with proper enrichment can be sufficient, reducing the overhead introduced by mesh generation and adaptation significantly. Fracture mechanics is an example where enrichment functions can be obtained by asymptotic analysis of the analytical solution of cracks growing in solid material (see Section 7.2.1). In fluid mechanics, a precise analytical description of characteristic non-smooth structures, like shock waves or boundary layers, is typically not available. In such cases, data obtained from experiments or from representative precomputed simulation results (so-called handbook functions) can be used as algebraic refinement functions. From the computer graphics perspective, the enrichments can be seen as textures mapped onto a coarse geometry to simulate an increased geometric fidelity without explicit mesh refinement.

[3] For an optimal approximation property the use of so-called blending elements for certain enrichment functions might be necessary [58, 173].

While the algebraic refinement improves the approximation properties, it can adversely affect the stability of the basis functions. This can prevent the efficient solution of the arising linear system. The particle-partition of unity method, described in the following section, offers a solution to this issue. It additionally avoids the tedious process of mesh generation. The XFEM, in contrast, has the advantage that it can be integrated more easily into existing FEM codes, as it is based on traditional FEM grids.

6.1.3 Particle Partition of Unity Method

The particle-partition of unity method (PPUM) benefits from algebraic refinement but avoids the instability issues of GFEM and XFEM. It uses a so-called flat top PU in combination with local preconditioning [172]. Moreover, it constructs its PU with a meshfree scattered data technique and allows for multilevel solvers that can handle arbitrary enrichment functions [73]. Currently, the PPUM represents the most stable and efficient generalization of the FEM. In contrast to traditional FEM approaches, in PPUM, fields are represented in a meshless manner by a set of points $P = \{\mathbf{x}_i \,|\, i = 1 \dots \hat{N}\}$ distributed within the domain Ω, as shown in Figure 6.4 (a). To obtain a continuous field representation, first a quadtree is constructed in 2D domains, or an octree in 3D domains, respectively. The following discussion assumes the two-dimensional case. The quadtree is obtained by subdividing an initial bounding-box $\mathcal{B}_\Omega \supset \Omega$ of P until each cell

$$\mathcal{C}_i = (c_i^x - h_i^x, c_i^x + h_i^x) \times (c_i^y - h_i^y, c_i^y + h_i^y),$$

with center (c_i^x, c_i^y) and size $(2h_i^x, 2h_i^y)$, contains at most a single point $\mathbf{x}_i \in P$. These cells represent the leafs of the quadtree, as illustrated in Figure 6.4 (b). In the next step, the pairwise disjoint cells $\mathcal{C}_i$ are enlarged by scaling with a factor $\alpha > 1$ resulting in a set of overlapping patches

$$\omega_i := (c_i^x - \alpha h_i^x, c_i^x + \alpha h_i^x) \times (c_i^y - \alpha h_i^y, c_i^y + \alpha h_i^y), \tag{6.6}$$

see Figure 6.4 (c). The set $\{\omega_i\}$ is called the cover C_Ω. A cover patch ω_i is defined for leaf-cells $\mathcal{C}_i$ $(i = 1 \dots N)$ that contain a point $\mathbf{x}_i \in P$ as well as for empty cells that do not contain any point from P.

On each cover patch ω_i the PPUM defines a local field approximation u_i—similar to the DG method. The u_i, however, are coupled with their neighbor patches by using the partition of unity $\sum_{i=1}^{N} \varphi_i \equiv 1$ (described at the end of this section) to achieve a smooth blending in the overlap regions. This results in the global PPUM approximation

$$u^{\mathrm{PU}}(x, y) = \sum_{i=1}^{N} \varphi_i(x, y) u_i(x, y), \tag{6.7}$$

a weighted sum of the u_i. Consistent with the XFEM formulation (6.5), each local patch approximation u_i in general consists of a smooth polynomial part p_i and an

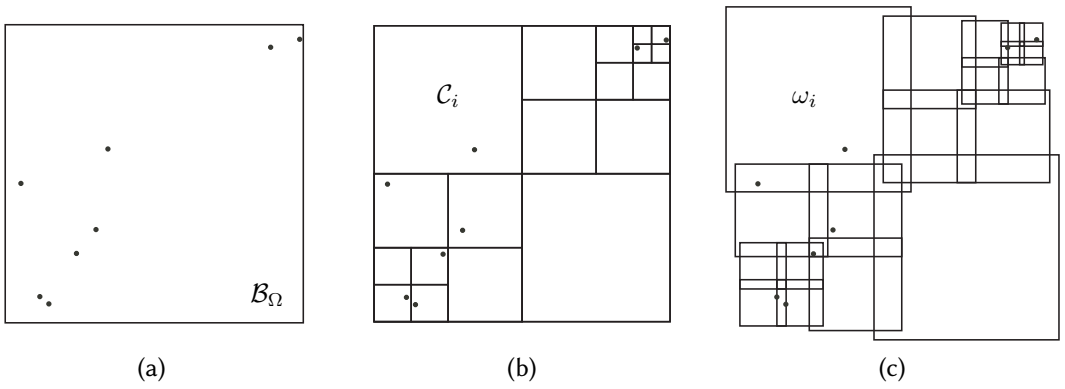


Figure 6.4 — (a) The PPUM is based on a meshless point representation of fields, (b) from which a quadtree with leaf cells $\mathcal{C}_i$ is constructed. (c) By scaling the cells the PPUM cover is derived, consisting of a set of overlapping patches ω_i.

application-dependent enrichment part, i.e.,

$$u_i(x,y) = p_i(x,y) + e_i(x,y). \tag{6.8}$$

To this end, coefficients p_i^s and e_i^t are associated with polynomial basis functions ψ_i^s and enrichments η_i^t respectively. Thus, the local approximation on a cover patch ω_i is given by

$$u_i(x,y) = \sum_s p_i^s \psi_i^s(x,y) + \sum_t e_i^t \eta_i^t(x,y). \tag{6.9}$$

Inserting (6.9) into (6.7) yields the respective global approximation

$$u^{\mathrm{PU}}(x,y) = \sum_{i=1}^{N} \varphi_i(x,y) \left(\sum_s p_i^s \psi_i^s(x,y) + \sum_t e_i^t \eta_i^t(x,y) \right), \tag{6.10}$$

which is used to solve the weak formulation of the corresponding problem. The use of local polynomials $p_i(x,y)$ that are spanned by a local basis ψ_i^s defined on ω_i alleviates p-adaptivity. The enrichment basis functions η_i^t are application dependent and are usually given as global functions η^t on the whole computational domain since they are designed to capture special behavior of the solution at a particular location in Ω. Examples of enrichment functions for crack propagation problems are given in Section 7.2.1.

Constructing the Partition of Unity

A partition of unity (PU) has to be defined on the PPUM cover C_Ω to be able to determine the contribution of the local patch solutions in overlap regions. The PU functions φ_i can be constructed by Shepard's approach [177]. For each cover patch $\omega_i \in C_\Omega$, a weight function $w_i : \Omega \to \mathbb{R}$ with support $\mathrm{supp}(w_i) = \omega_i$ is defined as

$$w_i = \begin{cases} \mathcal{W} \circ T_i(x,y) & (x,y) \in \omega_i \\ 0 & \text{else} \end{cases} \tag{6.11}$$

The involved affine transformation $T_i : \omega_i \to [0,1]^d$ maps the world coordinates to local patch coordinates. These are then used to evaluate the reference d-linear B-spline $\mathcal{W} : [0,1]^d \to \mathbb{R}$. The Shepard functions

$$\varphi_i(x,y) := \frac{w_i(x,y)}{S(x,y)}, \quad \text{with} \quad S(x,y) := \sum_{j=1}^{N} w_j(x,y) \tag{6.12}$$

are then derived by averaging the weight functions. The functions $\{\varphi_i\}$ with $i = 1,\ldots,N$ form a partition of unity on the cover, i.e., they satisfy $0 \le \varphi_i(x,y) \le 1$ and $\sum_{i=1}^{N} \varphi_i \equiv 1$ at all points of the domain.

6.2 Toward Higher-Order Visualization

The traditional approach for visualizing higher-order data is based on resampling and subsequent visualization with interpolation-based techniques. Direct visualization of polynomial field data of high order can provide many advantages over this traditional approach. First, the excessive computation and increase in data size induced by resampling is avoided. Secondly, and even more important, it is easier to obtain pixel-accurate images when directly working with the original polynomial data. Unfortunately, visualization algorithms for higher-order basis functions of arbitrary degrees given on arbitrary cell types have limited coverage in the literature and are still an open research problem. The lack of general higher-order visualization tools is also partly due to the large number of available FEM methods that often use their own proprietary basis functions and element mappings from reference space to physical space. Moreover, interactive display rates are typically hard to achieve because of the expensive evaluation of the polynomial solutions. Existing approaches for higher-order data are discussed in the following. A more extensive overview is given in [160].

An exact rendering of nonlinear functions is not directly supported by standard graphics APIs like OpenGL, since they are designed to work with planar primitives that are intrinsically linear (Section 2.2). As one solution to this problem, resampling is widely used. The simplest approach neglects the polynomial variation within the cells and only keeps the field values at the grid vertices. For polynomial data of high order, this is only appropriate to get a quick overview of the field. Figure 6.5 demonstrates that trilinear approximation within a hexahedron might not only fail to capture the features contained in the higher-order polynomial solution, but also can lead to false interpretations. Simply choosing a higher fixed resampling resolution is typically also no option if accurate results are desired. Accurate techniques have to subdivide the higher-order meshes adaptively to obtain an optimal linear grid that can be rendered with standard visualization tools. Usually, error metrics are used to control the subdivision depth [154, 169]. These methods typically involve an expensive preprocessing step, often resulting in an immense increase of data size.

Resampling is avoided by techniques that directly visualize higher-order data. These techniques project the polynomial solution, often given in reference space, to the

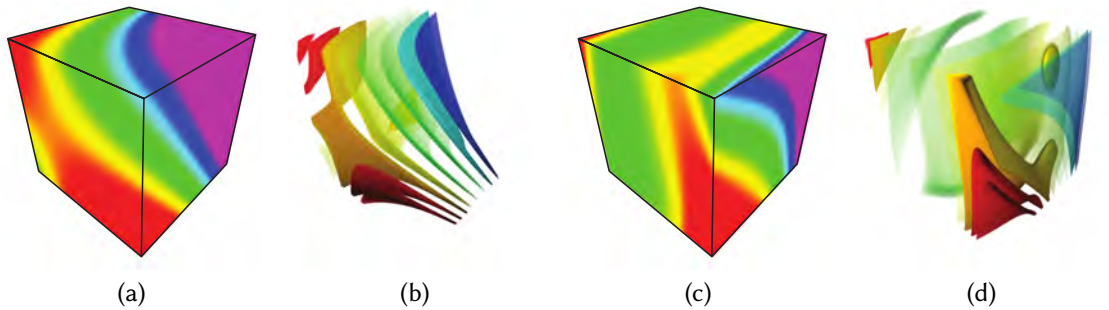


Figure 6.5 — Trilinear approximation of a hexahedral higher-order element with (a) opaque rendering, (b) DVR misses the details of the polynomial solution (c) and (d).

physical world space [75]. One issue is the costly inverse mapping of positions from world space to reference space, as there is, for instance, no analytical solution for general curved elements (see Section 4.3). Williams et al. [213] are able to visualize quadratic tetrahedral elements with flat faces with direct volume rendering accurately. Other cell types can also be handled, but not in high quality. Curved tetrahedra are visualized by the ray casting and isocontouring techniques of Wiley et al. [210, 211]. They accurately fit parametric curves to the curved rays in reference space. Subsequently, the polynomial solution is sampled along these curves. Their approach, however, is limited to quadratic elements. Liu et al. [110] recently presented an interactive volume ray casting technique for data of arbitrary order. Following a similar approach like Wiley, they precompute a set of potentially curved viewing rays for each element in order to avoid the costly mapping from world to reference space during rendering. Clustering techniques are employed in order to reduce the large number of curves. Additionally, techniques for inter- and intra curve interpolation are used to improve image quality during runtime. While interactive display rates are achieved, the precomputation is very time-consuming. Hence, only comparatively small data sets can be handled. Nelson and Kirby's pixel-exact ray casting technique for isosurfaces in spectral hp-elements follows an alternative approach [133]. The 3D reference space solution is projected onto 1D polynomials defined along the viewing rays in physical space. Then, the ray-isosurface intersections are identified by solving a 1D root finding problem for each ray. The technique, however, is far from being interactive. Interactive pixel-accurate visualization on cut-surfaces was presented later by the same authors [132], followed by an open-source release of their software [134]. While not directly related to higher-order simulation data, the isosurface ray caster for arbitrary implicit functions by Knoll et al. is also noteworthy in this context [98]. It uses a fast GPU implementation based on interval arithmetics and affine arithmetics for robust root finding. A hardware accelerated rendering approach for 2D slicing of 3D space-time cubic tetrahedral grids is discussed in [221]. There, the element mappings are linear, allowing a fast inversion and evaluation of the solution in reference space.

Meshless approximation methods are the third class of higher-order visualization techniques. Zhou and Garland [220] implemented a point-based volume visualization system that resamples non-conforming tetrahedral meshes with points by applying Lloyd relaxation in a preprocessing step. Meyer at al. proposed a particle system that approximates isosurfaces in higher-order finite elements in reference space [123]. Typically, there is a trade-off between the required pre-computation time and desired accuracy of the approximation with this class of methods.

There are few feature-based techniques for higher-order data. Pagot et al. presented a method for line-type feature extraction and for isosurface extraction [141, 142].

Chapter 7

Direct Visualization of 2D Higher-Order Fields

The visualization of two-dimensional higher-order fields with popular visualization frameworks like ParaView requires a resampling of the data to a linear representation. Modern GPUs, however, do provide high enough performance for on-the-fly evaluation and pixel-accurate visualization of such data. This chapter discusses the advantages of pixel-accurate direct visualization of higher-order 2D fields, e.g., with transfer functions mapping the field values to color. To this end, an interactive GPU-based visualization framework for particle-partition of unity (PPUM) simulation data has been developed [199]. From a data-centric view, PPUM data represent a combination of cell-based (local coordinate-based) analytic representations similar to higher-order FEM and enrichment functions placed in a scattered manner in global coordinates. As such, the PPUM represents a generalized FEM approach (see Section 6.1.3), generating data that none of the existing higher-order visualization methods can handle (Section 6.2). While the following discussion specifically targets PPUM data, it is also representative for higher-order 2D fields from FEM or DG methods. Their polynomial field solution can be handled as a special case.

Section 7.1 presents the interactive pixel-accurate visualization framework for two-dimensional PPUM fields of high order and their spatial derivatives. Details on its efficient OpenGL shader implementation are covered. Results are presented in Section 7.2 and the advantages of the method in comparison to traditional techniques like resampling are discussed. The evaluation is based on a crack simulation scenario computed with PPUM, involving derived field quantities like the von Mises stress. Moreover, the framework is able to visualize solver performance, allowing for a more directed simulation design by means of visual debugging. Section 7.3 concludes the chapter and gives an outlook on future work.[1]

[1] Parts of this chapter have previously been published in Üffinger et al. [194].

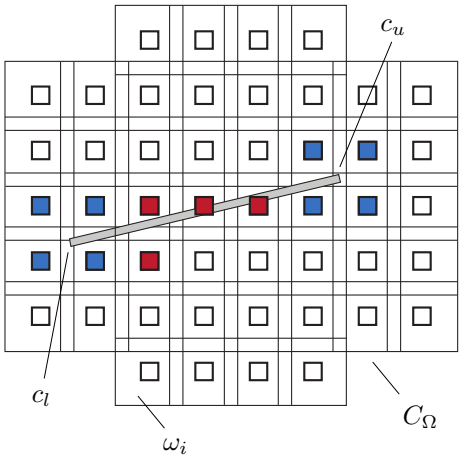


Figure 7.1 — Schematic view of a PPUM crack simulation with cover C_Ω. The centers of the overlapping patches $\omega_i \in C_\Omega$ (aligned on a uniform gird) are marked by squares whose color indicates the type of crack enrichment: additive enrichments (blue) and multiplicative enrichment (red) are used near the crack line with crack tips c_l and c_u. White color indicates a pure polynomial approximation.

7.1 Pixel-Accurate Visualization of PPUM Data

This section presents a pixel-accurate visualization framework that exemplifies the challenges involved in interactive visualization of 2D field solutions computed by PPUM simulation methods. The system utilizes the OpenGL rendering pipeline and the flexibility of its shading language GLSL to achieve accurate evaluation and visualization of the higher-order data. As described in Section 6.1.3, the analytical global PPUM field solution is given as $u^{\mathrm{PU}}(x, y)$, with world space coordinates $\mathbf{x}_w = (x, y) \in \Omega$ being located within the PPUM cover C_Ω (see Equation (6.7) and (6.10)). The global PPUM solution represents a weighted sum of local per patch solutions u_i, each consisting of a polynomial part p_i and an optional enrichment part e_i.[2] The enrichments are often specified in an analytical manner. One example are the additive enrichment functions shown in Figure 7.3, which are used in crack propagation scenarios. These functions capture characteristic singularities in the material deformation field that cannot be captured appropriately with resampled representations. They are used to enrich the PPUM patches near cracks, as illustrated in Figure 7.1. Hence, pixel-exact PPUM visualization techniques do not only have to evaluate the polynomial part p_i but also the true analytical representation of the enrichment functions e_i. The characteristic enrichment functions are specific to a particular physical problem and its PPUM solution. To account for this, our rendering pipeline provides a GLSL function interface that allows to easily replace the implementation of enrichment functions. An automatic replacement of the GLSL implementation will hence become possible in future applications where the enrichment functions are selected dynamically during simulation.

An overview of the PPUM rendering pipeline is depicted in Figure 7.2. The higher-order field solution is defined on a number of overlapping rectangular PPUM patches that cover the domain Ω. In order to evaluate (6.10), the system renders the geometry of

[2] The PPUM scheme restricts the influence of a local solution u_i (6.8) to a corresponding patch ω_i that overlaps with neighboring patches (see Figure 7.1).

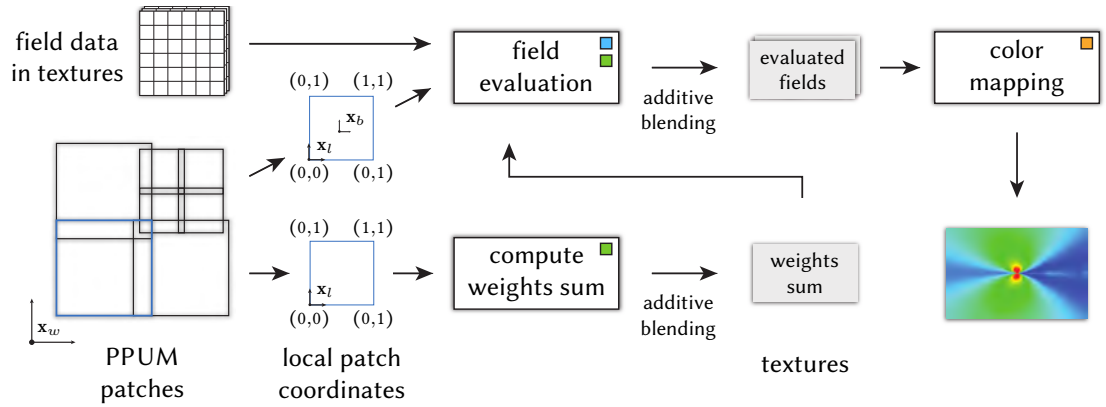


Figure 7.2 — PPUM rendering pipeline. Colored boxes mark problem specific implementations of the GLSL function interface: PU weight function and its gradient (green), enrichment functions (blue), derived field functions (orange).

the individual overlapping patches and accumulates the corresponding field solutions on a per-pixel basis in image space, similar to splatting techniques. The local field solution at the pixel locations are therefore evaluated in a fragment shader. The necessary field data, like the coefficients of the polynomial and the enrichment part of the local solutions, are provided in data textures. While the splatting approach primarily accounts for the scattered data aspect of the PPUM data, it can also be applied to traditional FEM and DG data with non-overlapping cells. Section 7.1.1 describes the evaluation core that handles the evaluation of the polynomial part of the solution of a specific patch ω_i, computes the contribution of the patch enrichment functions by calling the exchangeable enrichment function implementation, and weights everything according to the partition of unity function of the patch. Finally, after the contributions of all patches have been summed up on a per-pixel level, the field values stored in the screen-sized field textures are mapped to color (Section 7.1.2).

7.1.1 Field Evaluation

In the following, the field evaluation algorithm is described at the example of a single scalar field solution. The extension to multiple scalar fields, vector fields, and their first-order analytical derivatives is discussed in Section 7.1.3.

The global field solution $u^{\mathrm{PU}}(x, y)$ given in (6.7) and (6.10) consists of multiple terms. Besides the local per patch solution function $u_i(x, y)$, there is the PU part φ_i that determines the contributions of the $u_i(x, y)$ in regions where multiple neighboring patches overlap with ω_i. Consequently, multiple patches can contribute to the final field value of a pixel. The contribution of each patch at a pixel (determined by the PU) can be separated from each other if the sum of all weight functions $S = \sum w_i$ (see Equation (6.12)) is computed at each pixel first.

To compute the weight function sum S on the whole domain Ω, the individual overlapping patches ω_i are rendered separately. For each generated fragment, the weight function w_i is evaluated at interpolated local patch coordinates $\mathbf{x}_l \in [0,1]^2$. Fast hardware accelerated OpenGL blending is employed to sum up the contributions of overlapping patches in the floating point texture *weights sum* attached to the render target. The flat top PU of the PPUM implies that the weight sum S is equal to $w_i(\mathbf{x}_l)$ in regions covered by a single patch ω_i only.

Now, the contribution $\varphi_i u_i$ of each patch ω_i to a pixel can be computed separately by a fragment shader. To this end, the involved weight function sum S at the corresponding fragment is fetched from the texture computed in the previous stage. The contributions of multiple ω_i to a pixel are again summed up with additive OpenGL blending. To be able to evaluate the correct solution function u_i, the individual patches ω_i are rendered with additional attribute values attached to each vertex. This includes a unique patch identification number that allows to access the patch data from within the shader. For this purpose, the coefficients of the polynomial and enrichment functions are stored in textures. Additionally, element-local coordinates $\mathbf{x}_l \in [0,1]^2$ are attached to the patch geometry. For each pixel covered by the patch geometry, the rasterization engine automatically generates a fragment and calculates its interpolated local coordinates $\mathbf{x}_l$ and corresponding world coordinates $\mathbf{x}_w \in \Omega$. The field contribution u_i (6.9) is then evaluated at the interpolated position.

The system provides a main evaluation routine that includes the evaluation of the polynomial part of the solution in patch-local barycentric coordinates $\mathbf{x}_b = 2 \cdot (\mathbf{x}_l - 0.5)$. The polynomials are given in a monomial basis representation, which comes with the advantage that only the polynomial coefficients need to be stored. The structure and the degree p of the corresponding basis functions $\psi_i = x_b^k y_b^l$ with $k + l \leq p$ can be easily reconstructed during runtime, for example, by using a Morton order. Besides compact storage, the monomial representation additionally enables a simple iterative evaluation of the polynomials on the GPU (see Section 8.1.2 for more details).

In the case of patches with algebraic enrichment, the multiplicative and additive enrichment functions have to be evaluated too when computing $\varphi_i u_i$. As the additive and multiplicative enrichment functions are problem specific, we designed a simple GLSL function interface, which is used by the evaluation core routine.[3] Data required by the actual implementation, like the orientation of the crack tip functions of our example problem, are provided through a data texture. As already indicated, the enrichment functions are typically defined with respect to the global world space system. The function interface allows the problem specific parts of the PPUM function evaluation system (marked as colored boxed in Figure 7.2) to be replaced easily. With the GLSL compilation approach, this can be done during runtime without the need to restart the application.

[3] If the core, e.g., needs to evaluate the N_a additive enrichment functions at $\mathbf{x}_w$, it calls the function void EvalAddEnrich(vec2 $\mathbf{x}_w$, int patchId, in sampler2D enrichData, out float values[N_a]).

7.1.2 Color Mapping

The mapping of the field values to color is performed in a separate render pass. First, the system calls a predefined function that selects a field component or maps the computed field values to a single scalar value, like the gradient field magnitude. Alternatively, a replaceable mapping function implemented for a specific problem is called. This is done in the case of the von Mises field in Section 7.2.2. Finally, the scalar value is transferred to the color domain by using a 1D transfer function.

7.1.3 Vector Fields and Spatial Derivatives

The evaluation of vector fields is a straightforward extension. As each component of the vector field comes with a full set of coefficient vectors, the components can be handled independently like scalar fields. The only difference is that the evaluation core needs to evaluate two scalar functions and the textures storing the result of the field evaluation in image space have to provide twice the storage space.

Often, the visualization of derived fields that build upon spatial field derivatives is of major interest. To account for this, the core system provides functionality to compute analytic first derivatives of the field solution. This includes the gradient of scalar fields and the Jacobian of vector fields. To this end, the respective gradient function implementations need to be derived for the problem specific enrichment functions. In case of the polynomial solution, the derivative can easily be computed, e.g., for a specific monomial term $x_b^k y_b^l$ the derivative in x_b direction is simply given as $k x_b^{k-1} y_b^l$. This enables the core to accurately compute the scalar gradient field with virtually no overhead. This is a major advantage over sampled field data which require gradient estimation techniques (Section 3.4). The gradient field has to be evaluated with respect to world space, but the polynomial solution is defined in the local barycentric space of the patch. Hence, the chain rule needs to be applied during gradient computation. In regions with overlapping patches this requires some extra effort. Let us assume that the scalar field contribution of a single patch ω_i is given as $\varphi_i u_i$ with $\varphi_i = \frac{w_i}{\sum_l w_l}$ being the PU contribution and u_i the local field value of the patch. Its gradient

$$\nabla(\varphi u_i) = \nabla\varphi_i u_i + \varphi_i \nabla u_i \tag{7.1}$$

then does not only involve ∇u_i, but also

$$\nabla\varphi_i = \nabla\frac{w_i}{\sum w_k} = \frac{\nabla w_i \sum w_k - w_i \sum \nabla w_k}{(\sum w_k)^2}, \tag{7.2}$$

which includes the gradient of the PU weighting function ∇w_i. To be able to evaluate this, the sum of the weight function gradients $\sum \nabla w_k$ has to be additionally computed by the weights sum shader and stored in the weights sum texture. The contributions of the individual patches can then be simply summed up and stored in the field textures, similarly to the scalar field evaluation approach.

Multiple scalar and vector fields and their gradients can be computed simultaneously by attaching multiple RGBA floating point textures as render targets to the field evaluation shader. These fields can then be used as building blocks to construct more complex derived fields in the color mapping stage.

7.2 Results and Evaluation

The higher-order visualization framework is evaluated with several simulation results of fracture mechanics problems, computed with the PPUM in two dimensions (Section 7.2.1). In particular, the static loading of a pre-cracked steel panel and the propagation of a crack is investigated. The equations of elasticity are approximated on a two-dimensional domain with internal traction-free boundaries. From a numerical point of view, the main issue in these simulations is the accurate approximation of the displacement field near the crack tip. The main question, from an application point of view, is if the material fails (near the crack tip) due to the current loading conditions. A widely used criterion for the failure of material is the so-called von Mises stress, a pointwise scalar measure of tensorial stress data obtained from the Jacobian of the displacement field. Linear splines were employed as weight functions by the simulation used in the examples in Section 7.2.2–7.2.4.[4] Finally, in Section 7.2.5, the performance of the interactive system is investigated.

7.2.1 Fracture Enrichments

Fracture mechanics simulations compute a displacement field **u**, represented as a 2D vector field that describes the geometric deformation of a solid material. Stresses acting on the material can lead to the development of a crack $C \subset \Omega$. This introduces a discontinuity into the field **u** along the crack line. Singularities additionally appear at both crack tips c_l and c_u (see Figure 7.1 for a schematic sketch of a crack line). In the PPUM, enrichment functions are used to model this behavior. For patches ω_i with

$$\omega_i \cap C \neq \varnothing \quad \text{and} \quad \{c_l, c_u\} \cap \omega_i = \varnothing, \tag{7.3}$$

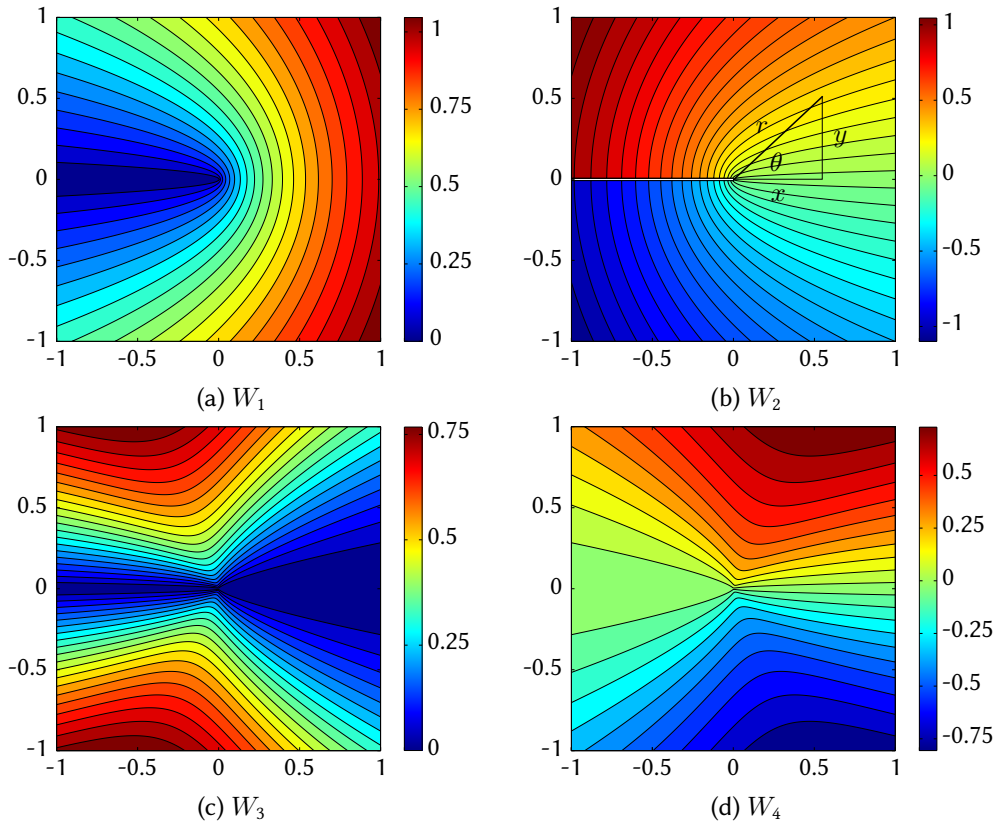
(the red patches in Figure 7.1), the local polynomials $p_i(x,y) = \sum_s p_i^s \psi_i^s(x,y)$ are enriched by the additional basis functions

$$H_{\pm}^C(x,y)\psi_i^s(x,y), \tag{7.4}$$

with the Haar function $H_{\pm}^C$ modeling the discontinuous jump appearing in the displacement field along the crack C. In those regions, the respective local approximation u_i is given by

$$u_i(x,y) = \sum_s p_i^s \psi_i^s(x,y) + \sum_s q_i^s H_{\pm}^C(x,y)\psi_i^s(x,y), \tag{7.5}$$

[4] Besides linear weight functions (6.11), more general weight functions are supported by the visualization framework, e.g., quadratic B-Splines, which are already implemented in the weight function interface.



(a) W_1 (b) W_2

(c) W_3 (d) W_4

Figure 7.3 — Additive crack tip enrichment functions represented with the analytical Westergaard functions W (7.6). Traditional techniques would require strong h- and p-refinement near the singularities at the crack tip $\mathbf{c}_u = (0, 0)$. Moreover, the discontinuity of the horizontal crack line, which can be seen in (b), would need to be resolved.

with two sets of polynomial coefficients p_i^s and q_i^s. This type of enrichment is denoted as *multiplicative enrichment*. Patches ω_i in the vicinity of crack tips, for which (7.3) is not valid (the blue patches in Figure 7.1), are enriched by the Westergaard stress functions

$$W := \{\sqrt{r}\cos\frac{\theta}{2}, \sqrt{r}\sin\frac{\theta}{2}, \sqrt{r}\sin\theta\sin\frac{\theta}{2}, \sqrt{r}\sin\theta\cos\frac{\theta}{2}\}, \tag{7.6}$$

given in local polar coordinates with origin at the tip. The polar coordinates (θ, r) are obtained as $(\text{atan2}(y, x), \sqrt{x^2 + y^2})$ from Cartesian coordinates with $\theta \in (-\pi, \pi]$. Figure 7.3 illustrates the four Westergaard stress functions W_i that are used to model the displacement behavior near the crack tip. These functions are derived from an asymptotic expansion of the solution and capture its dominant singularity. For these patches, the local approximation u_i is given by

$$u_i(x, y) = \sum_s p_i^s \psi_i^s(x, y) + \sum_t w_i^t \eta_i^t(x, y), \tag{7.7}$$

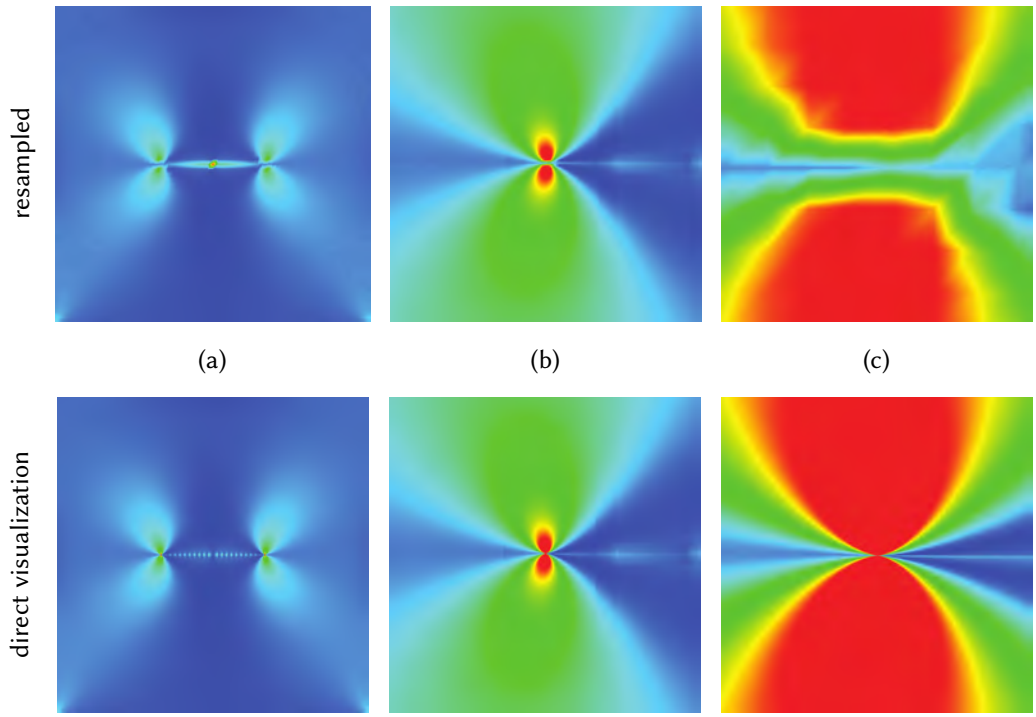


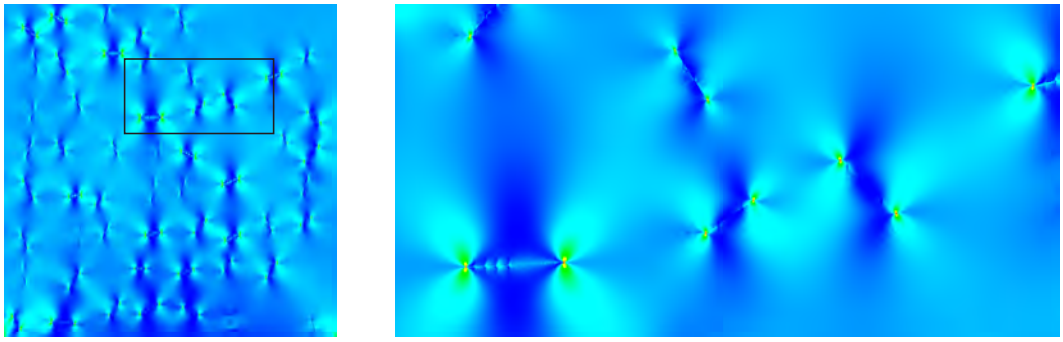
Figure 7.4 — Comparison of traditional resampling (top) with our method (bottom) using the horizontal center crack with its typical stress distribution at both crack tips. (a) Resampling at patch resolution 64^2 misses features and shows incorrect features, e.g., at the center, (b) ROI of the left crack tip resampled with 2048^2 and (c) close-up. The direct visualization is pixel-accurate and avoids misinterpretations, e.g., at the singularity.

with the polynomial coefficients p_i^s and the four additional coefficients w_i^t associated with the four enrichment basis functions $\eta_i^t = W_i$. This type of enrichment is referred to as *additive enrichment*.

In summary, to evaluate the global PPUM approximation (6.10), the weight functions $w_i(\mathbf{x}_l)$ (6.11) and the basis polynomials $\psi_i(\mathbf{x}_l)^s$ can be evaluated in local patch coordinates $\mathbf{x}_l$. The multiplicative enrichment functions, i.e., the Haar function $H_{\pm}^C(\mathbf{x}_w)$ (7.4) are evaluated in global coordinates $\mathbf{x}_w$ and the Westergaard functions $W(\theta, r)$ (7.6) are evaluated in the local polar system (θ, r) centered at the crack tips and oriented along the crack lines.

7.2.2 Horizontal Center Crack

This example considers a panel that is fixed at its lower horizontal boundary and loaded in vertical direction on the upper boundary. Additionally, a crack is placed at the center



(a) (b)

Figure 7.5 — (a) Von Mises stress in a steel plate precracked at multiple locations with fixed bottom and a vertical drag force applied at the top with (b) close-up. The PPUM discretization uses a cover consisting of 16384 patches of order 2, 4006 additive and 138 multiplicative enriched patches (3.97 ms @ 1024×1024).

of the panel. Under these loading conditions the crack is expected to open in vertical direction and the characteristic singularities at the two crack tips will be clearly visible in the von Mises stress. This behavior can be observed from the visualization shown in Figure 7.4. The PPUM simulation in this example used a quadtree cover resembling a Cartesian grid, similar to the cover shown in Figure 7.1, but with a horizontal crack. Additive enrichments were applied only in the vicinity of the two crack tips to capture the singular behavior of the solution. A coarse uniform subdivision on quadtree level 6 with 64^2 patches was used, to define the PPUM cover and compare our visualization results with the traditional resampling approach, which is likely to yield misleading results because the sampling points cannot capture the leading singularity (see Figure 7.4 (a), top). Moreover, even an extreme fine resampling cannot resolve the singular behavior at the crack tip (Figure 7.4 (c)). Direct pixel-exact visualization, in contrast, clearly captures the singular point (bottom). To achieve comparable results with resampling, an at least tenfold higher resolution would be required at this zoom level. But, resampling the whole domain with such a fine resolution would lead to a data set size of more than 2 GB for this simple example. Pixel-accurate visualization also helps to investigate the regions where multiple patches overlap. The discontinuities that might appear in these regions are not artifacts introduced by the visualization but part of the PPUM data—similar to discontinuities in DG data. While the displacement field is continuous in this example, the derivatives used in the von Mises field are discontinuous in the blending regions, as first-order weight functions are used. Resampling might miss these regions, potentially hindering visual debugging of the simulation. The visualization framework presented here can also handle more complex PPUM data, for example, with enrichments distributed in different regions of the domain. Figure 7.5 shows an extension of the center-crack simulation involving 64 arbitrarily oriented cracks.

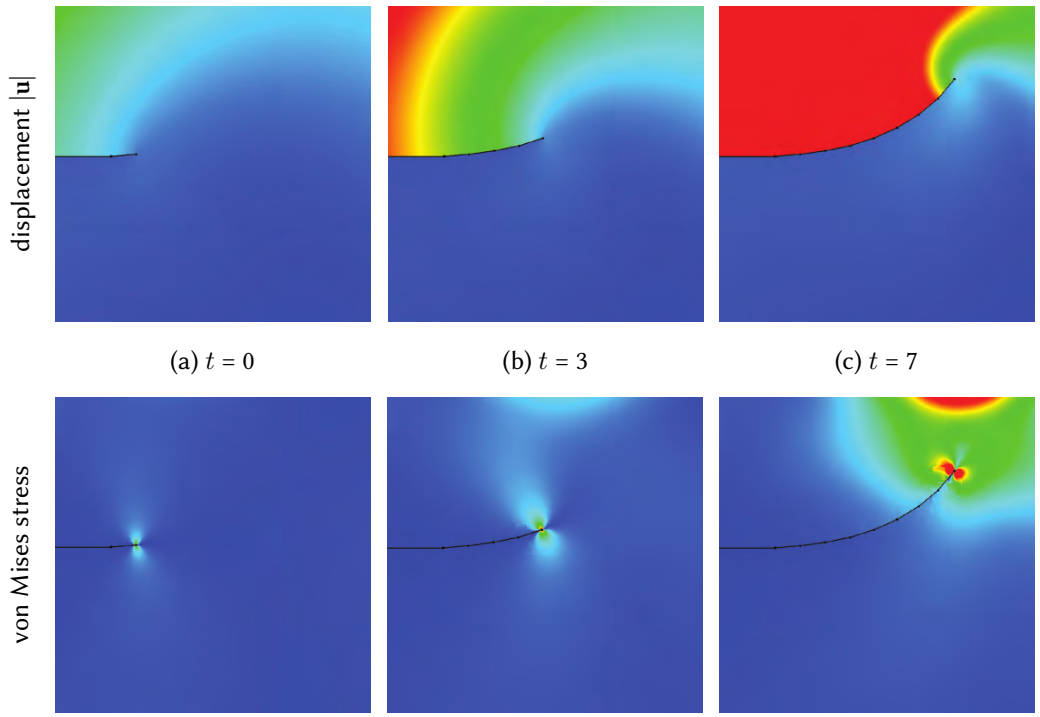


Figure 7.6 — Crack propagation visualized by displacement magnitude and von Mises stress. The black line depicts the growing crack (low values: blue, high values: red).

7.2.3 Crack Propagation

The second example investigates a time-dependent scenario of crack propagation in a steel plate. The simulation used a quasi-static approach and predicted the direction in which the crack will grow via the maximum hoop stress criterion. To this end, the stress intensity factors are extracted from the computed solution via the contour integral method. The visualizations depicted in Figure 7.6 clearly show the crack behavior expected from the employed loading conditions. The singularity at the crack tip moves through the simulation domain and the stress levels grow rapidly as the tip comes close to the domain boundary. This simple time-dependent scenario demonstrates what higher-order approaches effectively accomplish—on the simulation and the visualization side. While a traditional simulation method would be forced to resolve the cracks with an extremely fine grid resolution, e.g., by using local h-refinement, a very coarse spatial sampling is sufficient for the PPUM. Moreover, with PPUM, re-meshing can be avoided during the propagation of the crack. This is also a benefit for interactive visualization because the geometry of the cover only has to be loaded once. Subsequently, during rendering, only the solution functions have to be updated if a new time step is loaded.

7.2.4 Hp-Adaptive Simulation and Visual Debugging

The visualization framework also supports adaptive PPUM data. In the following, the results of three PPUM simulations are investigated (Figure 7.7). Two simulations use algebraic refinement in combination with h-refinement—the first with fixed polynomial order $p = 1$ (a) and the second with $p = 2$ (b) on all patches. The third simulation additionally harnesses p-adaptivity (c). The overall quality of the results is comparable with respect to accuracy. As shown in the figure, simulation debugging is supported, with graphical overlays communicating solver behavior, like the polynomial degree or the enrichment type. This can reveal the complex interplay between h-, p-, and algebraic refinement, and help debugging the refinement strategy of the solver. In the example, the refinement patterns clearly indicate the overall performance advantages of higher-order methods, especially in connection with the chosen enrichment scheme. These schemes allow for much larger patches at the same accuracy.

With our pixel-accurate visualization approach, very fine details of the simulation result can be detected. This additionally helps analyzing the PPUM and its properties. In Figure 7.4, for instance, the visual artifacts observable on the crack line (b) are due to the fact that the additive enrichment functions employed around the two tips meet in the center of the domain where they cannot match. This leads to an (insubstantial) oscillation in the derivatives and the stress field.

7.2.5 Performance

The direct PPUM visualization framework is highly interactive for all examples, providing up to several hundred frames per second. Nevertheless, a detailed performance analysis is given to investigate the scaling behavior of the system. All experiments

Table 7.1 — Render times in ms per frame of the center crack example with one quarter enriched patches at different polynomial orders p. Experiments (1)–(4) with varying p (c states the number of polynomial coefficients), (5) OpenGL immediate mode rendering, (6) without enrichments, (7) and (8) visualize vector field magnitude only instead of von Mise field (Intel Core i7-2600, NVIDIA GeForce GTX 670 @ 1024 × 1024).

number of patches		4	16	64	256	1024	4096	16384
(1)	p2, c6	2.17	2.37	2.50	2.66	2.78	3.06	3.95
(2)	p4, c15	2.62	2.72	2.88	2.98	3.14	3.46	4.64
(3)	p8, c45	6.18	6.72	7.14	7.42	7.82	8.66	12.10
(4)	p16, c153	26.10	29.07	30.93	31.19	29.48	37.79	53.70
(5)	p2, immediate mode	2.18	2.38	2.49	2.64	2.88	4.88	10.48
(6)	p8, w/o enrichment	4.80	5.28	5.62	6.08	6.18	6.84	9.48
(7)	p4, vf. magnitude	2.18	2.38	2.5	2.64	2.8	3.06	3.96
(8)	p16, vf. magnitude	15.12	16.84	18.06	18.62	19.7	21.94	31.04

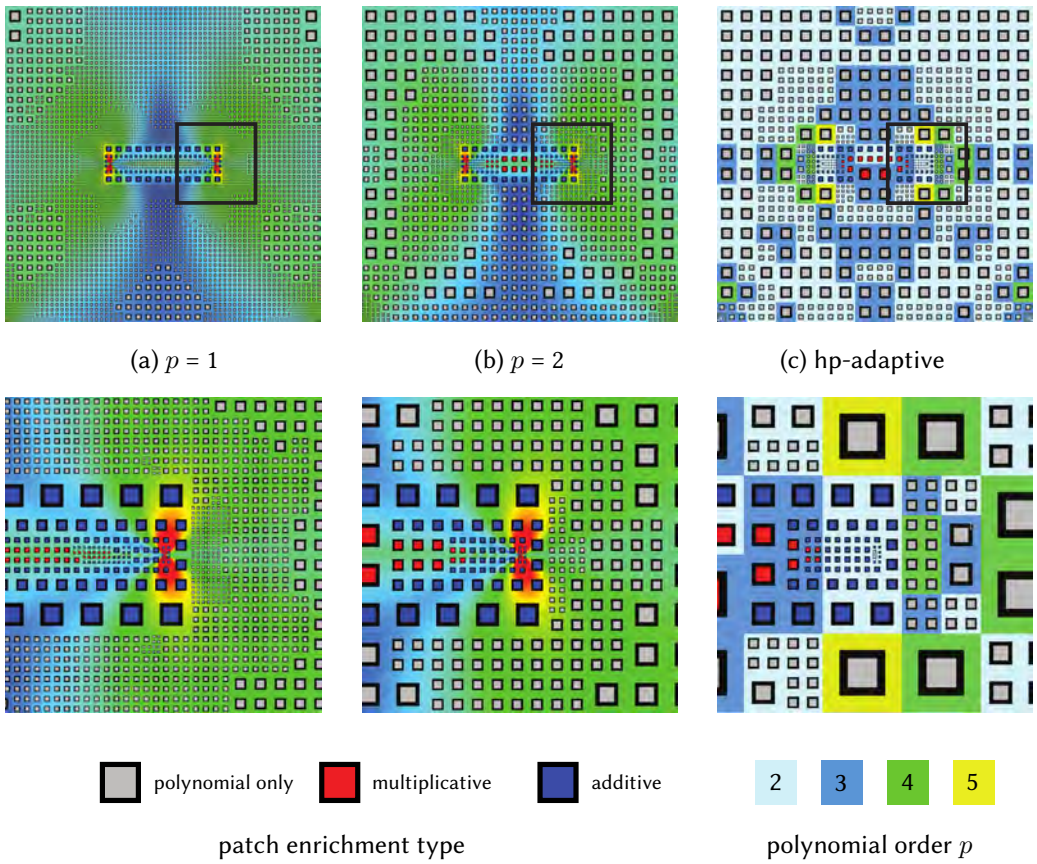


Figure 7.7 — Visual debugging of adaptive PPUM simulations with close-ups in the bottom row: (a) and (b) fixed polynomial degree p, (c) hp-adaptive simulation. Using $p = 1$ leads to strong h-refinement, whereas the hp-adaptive simulation shows the coarsest patches. The enrichment type is indicated by colored boxes with black border. For the hp-adaptive simulation the polynomial order is additionally mapped to color.

were performed on a workstation with an Intel Core i7-2600 CPU running at 3.40GHz, 16GB Ram, and a Nvidia GeForce GTX 670 GPU. The center crack example was chosen because its cover resembles a Cartesian grid at different quadtree refinement levels. One fourth of the patches have additive enrichments. If not stated otherwise, the von Mises field was visualized with a constant viewport of size 1024×1024 pixels, which exactly covered the PPUM domain.

Table 7.1 lists the results. At the lower polynomial degrees ($p \leq 4$), the values do not differ very much, and only slightly increase when a large number of patches are rendered. For $p > 4$, the evaluation of the PPUM solution u^{PU} becomes the dominating factor. In this range, the render times are about proportional to the number of polynomial coefficients that have to be evaluated. The render time grows sublinearly with respect to the number of patches, as the viewport resolution is fixed.

Extrapolation of the observed behavior indicates that the system is able to handle PPUM covers consisting of a much larger number of patches easily. It is noteworthy that buffering the geometry data on the GPU (the default approach) is required in order to achieve high frame rates for data sets with many patches. If buffering is deactivated and immediate mode OpenGL rendering is used (fifth experiment), the geometry transfer bottleneck becomes clearly visible. Disabling the PPUM enrichments leads to a slight increase in performance (sixth experiment). For the last two experiments the displacement field magnitude was visualized. In contrast to the von Mises field, the field derivatives do not have to be computed in these cases. This leads to a higher performance, in particular for larger polynomial degrees. All these measurements indicate that the system is able to handle more complex derived fields or weight functions of higher order.

7.3 Conclusion and Outlook

A framework for the efficient and accurate visualization of 2D fields from particle-partition of unity simulations has been presented in this chapter. The framework combines techniques from both scattered and cell-based higher-order data visualization to provide insight into this upcoming type of data. Accurate and efficient visualization of these data is of particular importance. In contrast to low-order techniques, the involved highly flexible simulation basis requires careful and thorough analysis. Such analysis can support efficient simulation case design, but can also support the research of this promising field of simulation techniques. GLSL runtime shaders have proven to be an elegant and viable approach to handle the complex PPUM field representation, which consists of polynomials enriched with problem-specific functions. As the described system is able to handle generalized FEM data, it is in principle also capable of handling cell-based polynomial data, like from 2D discontinuous Galerkin simulations. If necessary, routines for drawing non-rectangular cell shapes with attached local parameterizations have to be provided. Additionally, in the case of complex geometry mappings, a mechanism for handling the transformation to the reference element geometry is required. If the mapping cannot be inverted analytically, the evaluation step can become rather expensive. In such cases, it might be worth considering to project the polynomial solution to a monomial representation given in cell-barycentric coordinates in world space (see Section 8.1.2). The user, however, has to be aware that this typically increases the polynomial order and that potential projection errors have to be controlled.

The quadtree structure of the PPUM solution provides a starting point for future extensions of the system. Hierarchical visualization of the field data on the different PPUM resolution levels, for example, could enable progressive rendering. The quadtree structure could also serve as a spatial acceleration data structure to facilitate an efficient distributed rendering of large PPUM fields on compute clusters. An interactive exploration tool for the PPUM hierarchy would also help in analyzing the solver behavior.

Another direction is related to the nature of the crack problem. Instead of mapping the deformation field or a derived quantity to color, the simulation domain could also be deformed according to the vector field, leading to visualizations that are more intuitive. Kaufmann et al. [90] present a technique that is able to deform and cut geometric surfaces based on displacement textures. However, the pixel-accurate visualization of the PPUM field on a deformed surfaces still remains a big challenge. Due to the nonlinear geometry deformation, the iterative Newton-Raphson schemes would be required to obtain undeformed patch coordinates $\mathbf{x}_l$ for field evaluation. This, however, would most likely be hindered by severe convergence problems near the discontinuities and singularities that are modeled by the enrichment functions.

The proposed framework is optimized for rendering planar field domains making an extension to 3D fields difficult. Cutting planes, however, could be supported by providing a mechanism for rendering planar polygonal cell shapes, obtained as the intersections of 3D cells with the plane. With the local 3D coordinates being available within these shapes, the evaluation of the 3D field solution on the plane becomes straightforward. Techniques for the volumetric visualization of 3D higher-order fields are covered in the next chapter.

Chapter 8

Interactive Volume Visualization of Higher-Order Fields

The higher-order techniques presented in the previous chapter are optimized for 2D fields and cannot be directly applied to 3D fields. While the 2D framework can easily achieve interactive display rates, the number of field samples that have to be evaluated in 3D is much larger. Moreover, as the degrees of freedom of the cell solutions increases in 3D, each field evaluation involves a larger number of polynomial basis functions. Hence, sophisticated rendering techniques are required to still achieve high quality results at interactive display rates. Most available visualization tools for 3D fields of high order are limited to certain cell types and polynomial degrees, or use resampling approaches resulting in trade-offs in interactivity and quality. In particular, there existed no direct volume rendering (DVR) technique for the hp-adaptive DG data introduced in Chapter 6. The visualization framework presented in Section 8.2 overcomes these limitations. It allows interactive exploration of non-conforming unstructured grids in which each cell has its own higher-order polynomial solution. In contrast to the PPUM approach, the local solutions are not coupled through blending functions, leading to discontinuities at cell boundaries. The system employs GPU-based ray casting for direct volume rendering of complex grids that feature non-convex curvilinear cells with varying polynomial degree. Interactivity is achieved with a frequency-based adaptive sampling strategy that accounts for high field variations along rays. Additionally, the ray casting is distributed across a GPU cluster (Section 8.4). The performance and utility of the system is evaluated for different aeroacoustic simulations (Section 8.3) and techniques for visual debugging are discussed as well (Section 8.5).[1]

[1] Parts of this chapter have previously been published in Üffinger et al. [194].

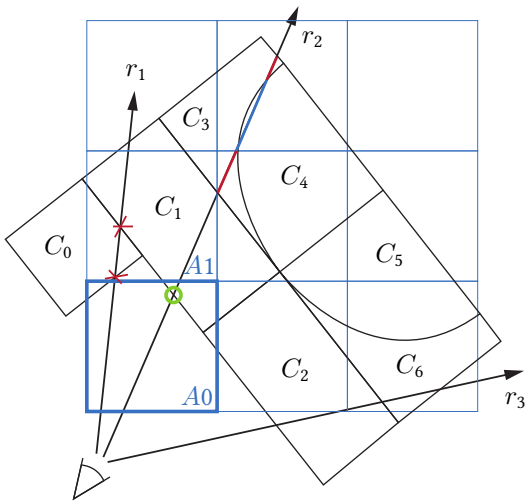


Figure 8.1 — Unstructured grid with curved cells and the atom grid acceleration data structure (blue grid) used for entry cell detection (green) and empty space skipping.

8.1 Building Blocks

The following introduces two important building blocks for the interactive and accurate volume visualization of our DG data. Section 8.1.1 discusses why ray casting is used at the core of our visualization framework. Section 8.1.2 then argues why the polynomial field solution should be converted to a monomial representation prior to visualization—an approach that is also employed by the PPUM framework presented in Chapter 7.

8.1.1 Ray Casting of Hp-Adaptive Grids

An introduction to direct volume rendering (DVR) was given in Section 5.3. While there are many techniques for low-order field data [117], only few techniques exist for higher-order data [110, 133] (see Section 6.2). Figure 6.2 illustrates the complexity of our DG data. The grid contains cells of different type with cell sizes varying over several orders of magnitude. Non-convex regions are possible and curved cubic faces are additionally used in order to improve the geometric approximation of the sphere obstacle. Non-conforming elements are supported as well, but only in the case of planar cell faces. Object-order DVR techniques, like cell-projection, are not able to handle such complex grids because a correct visibility sorting of the non-convex curved cells is not possible. Additionally, even for unstructured grids that contain convex cells only, visibility cycles might prevent a correct visibility sorting [99]. This is why we adopt the ray casting approach for our volume rendering framework. While the presented ray casting system focuses on DVR—it evaluates the discrete volume rendering integral (5.9) with a front-to-back compositing scheme (5.13)—it can easily be extended to support isosurface visualization or field visualizations on codimension 1 manifolds, like cutting planes.

With ray casting, the unstructured grid can be traversed efficiently along viewing rays by exploiting neighborhood information [62]. At first, the point where the ray

enters the grid has to be determined. Search data structures are often employed to accelerate this process (see Figure 8.1). As soon as the entry cell is known, the exit face, where the ray leaves the cell, can be computed. Then, the grid traversal algorithm can proceed with the next cell along the ray. This also works with curved element faces if appropriate ray-patch intersection algorithms are available. Figure 8.1 shows an example. A curved cell is traversed by the ray r_2 for a short segment (C_3, red). Afterward the ray leaves the cell, traverses another cell (C_4, blue), and then re-enters the first cell (C_3, red). As required by DVR, the contributions of the three segments to the final image can easily be blended together in visibility order.

8.1.2 A Monomial Field Representation for Visualization

As discussed in Section 6.1.1, the DG cell solutions $\hat{u}_i$ employ orthonormal basis functions and reside in the normalized element space depicted in Figure 6.3 (b). Hence, for visualization, the solution has to be transformed to world space, i.e., the geometry mapping $\Phi_{n,w}$ has to be inverted. In the case of the space-time DG method, this can be done analytically, as $\Phi_{n,w}$ is an affine transformation. For numerical reasons, it is beneficial to transform the normalized element space solution not only to a global world space system but to a barycentric coordinate system local to the cell, as illustrated in Figure 6.3 (c). The corresponding affine mapping is denoted with $\Phi_{n,b}$. The physical space polynomial of unchanged degree can easily be obtained with $u_b(\mathbf{x}_b) := \hat{u}(\Phi_{n,w}^{-1}(\mathbf{x}_b + \mathbf{t}_b))$. Here, $\mathbf{t}_b$ denotes the translation from the origin of the global world space system to the cell center. In our case, this transformation is performed offline by projecting the solution to world space. This reduces the number of operations that have to be performed during runtime.

In an interactive visualization system, the evaluation of the higher-order polynomials has to be as efficient as possible. Efficient caching and a fast iterative evaluation on the GPU, however, are only possible if a compact polynomial representation is employed. The orthonormal simulation basis $\{\tilde{\psi}_i\}$ is not optimal in this respect. Instead, we propose to use a monomial representation

$$u_b^m(\mathbf{x}_b) = \sum_{i=1}^{N(p)} c_i \, x^{k_i} y^{l_i} z^{m_i}, \tag{8.1}$$

where p is the maximum order and $N(p)$ is the corresponding number of basis functions. The monomial exponents k_i, l_i, m_i are given in an increasing degree order according to the scheme in Listing 8.1. The monomial representation is obtained from the orthonormal basis of the simulation by expansion and subsequent gathering of the monomial terms. In contrast to the orthonormal basis (see Section 6.1.1), monomial basis functions (6.2) do not depend on the geometry of the element, leading to a drastic reduction of the number of coefficients that are required to represent the polynomial. Thus, the basis is compact insofar as that only the coefficients c_i need to be stored explicitly. To further highlight the advantage of the monomial basis, the size of its

Table 8.1 — The number of coefficients for general elements of degree p is much smaller in the monomial basis $N(p)$ than the orthonormal simulation basis $\tilde{N}(p)$ (3D).

degree p	3	4	5	6	7
$N(p) = \frac{(p+1)(p+2)(p+3)}{6}$	20	35	56	84	120
$\tilde{N}(p) = \frac{N(p)(N(p)+1)}{2}$	210	630	1596	3570	7260
overhead factor	10.5	18	28.5	42.5	60.5

```
i ← 0
for r ← 1 to p + 1 do
 for s ← 1 to r do
  for t ← 1 to r − s + 1 do
   (k_i, l_i, m_i) ← (r − s − t + 1, s − 1, t − 1)
   i ← i + 1
  done
```

Listing 8.1 — Computation of the exponent tuples (k_i, l_i, m_i) for the monomial basis $\{b_i^{(p)}\} = \{x^{k_i} y^{l_i} z^{m_i}\}$ of degree p in ascending degree order with $i = 1, \ldots, N(p)$.

representation is compared to the size of the orthonormal simulation basis $\{\tilde{\psi}_j\}$. The structure of the orthonormal basis depends on the coefficients $\langle b_i^{(p)}(\mathbf{x}), \tilde{\psi}_j(\mathbf{x})\rangle$ in (6.3). A formula for the number of these coefficients can be directly derived from the Gram-Schmidt algorithm with an arithmetic series. Table 8.1 shows that with increasing degree p the number of total coefficients $\tilde{N}(p)$ of the orthonormal polynomial quickly becomes very large in comparison to the number $N(p)$ required for the monomial representation. It is noteworthy that $\tilde{N}(p)$ represents the worst-case scenario of a general (curved) cell where all coefficients are non-zero.

The monomial basis is not suitable for simulation since it does not yield sparsely populated matrices, which allow efficient iterative solvers to be used on the resulting system of equations. Moreover, it introduces numerical instabilities due to badly conditioned matrices. For visualization, in contrast, the solution function u_b^m only needs to be evaluated. To achieve the highest possible performance, the evaluation is done with single precision floating-point accuracy on the GPU. Single precision is sufficient for our data sets. A double precision evaluation, which we took as reference, revealed a very low maximum relative pointwise error of $1.3 \cdot 10^{-6}$ for the single precision evaluation of u_b^m in barycentric coordinates. In addition to compact storage and efficient evaluation on GPUs, the monomial representation is also beneficial if both the field and its gradient are required. This is the case for DVR, where the gradients are required for shading operations. With the monomial representation, the gradients ∇u_b^m come at virtually no extract cost.

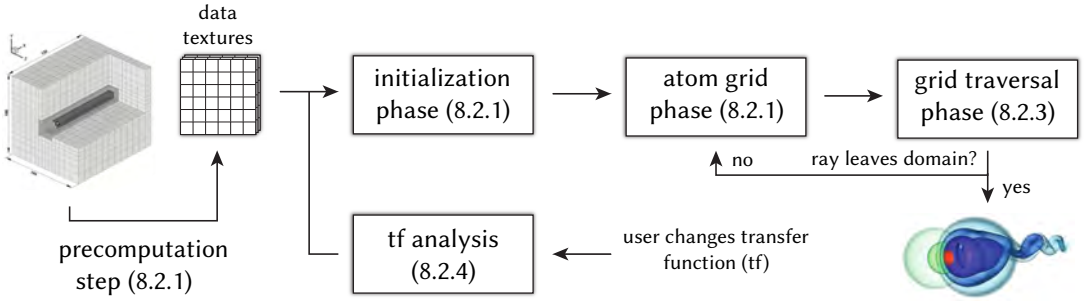


Figure 8.2 — Overview diagram of the higher-order ray casting system.

8.2 Higher-Order Ray Casting Framework

This section describes the higher-oder rendering system including the ray casting kernel implemented in CUDA. The framework is evaluated in Section 8.3 and parallelization on a compute cluster is discussed afterward in Section 8.4.

8.2.1 System Overview

Figure 8.2 shows an overview diagram of the system. Firstly, after loading the higher-order data set, three preprocessing steps are performed. An acceleration data structure, the so-called *atom grid*, is constructed in order to speed up the detection of the cells where the rays enter the grid. Secondly, bounding volumes are generated for curved element faces. Thirdly, a gradient analysis of the polynomial solution is performed, which is needed for adapting the sampling step sizes during ray casting. These steps are executed only once for each data set. The topology and geometry of the unstructured grid and the polynomial field data are stored in multiple textures to benefit from caching mechanisms. Figure 8.3 illustrates the data layout and inter-texture dependencies. For each cell, two links are stored. The first link refers to the list of cell faces stored in the (*faces*) texture. The second link points to the polynomial solution of the cell in the (*coefficients*) texture. For each polynomial of order p, the number of coefficients $N(p)$ is stored followed by the sequence of the $N(p)$ coefficients c_i. During ray traversal, the neighbor cell can be directly determined with the help of the backward links in the *face data* texture. Multiple indirections are required to access the complete information of the cell and its neighborhood. This is common for unstructured grids and introduces a higher latency compared to structured grids. Hence, if possible, the memory accesses should be interleaved with arithmetic instructions in order to hide the latency. While the atom grid is stored in a 3D texture, a 1D texture is sufficient for the transfer function. Each time the user changes the transfer function (tf), its frequencies are analyzed—a step that is required by the adaptive sampling mechanism discussed in Section 8.2.4. For each cell, an updated step size is computed and uploaded to the GPU, altogether taking less than one second. Then, the ray casting kernel is executed. For each pixel of the resulting image, one CUDA thread is executed that casts a ray into the scene

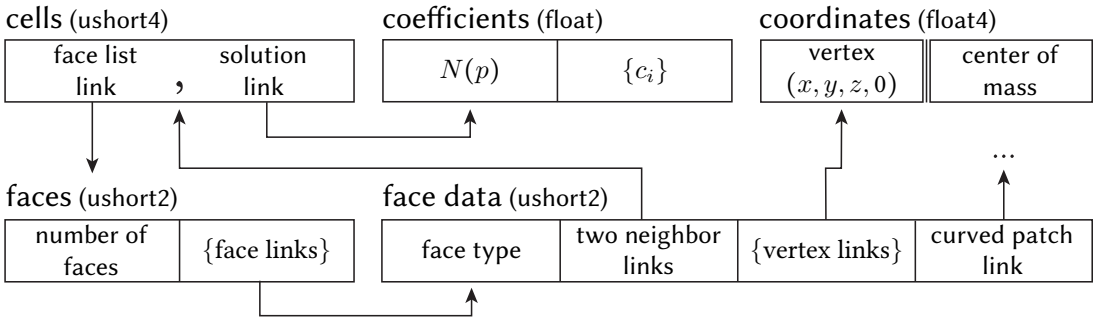


Figure 8.3 — Five 2D textures are used to store the hp-adaptive field data, each represented by a rectangle with annotated name and format, showing the layout of one data element. Links in ushort2 format refer to arbitrary locations in other textures.

starting at the camera location, as shown in Figure 5.3 (b). Many rays are traced in parallel, with their number only being limited by the resources of the chosen GPU hardware and the register and shared memory usage of the kernel.

The ray casting kernel consists of three major phases. In the initialization phase, the ray is constructed in world space and the intersection of the ray with the bounding box of the data set is calculated. The ray thread stops immediately if there is no intersection. Otherwise, the *atom grid phase* and the *grid traversal phase* are executed in an iterative manner, until an early ray termination criterion is fulfilled or the ray leaves the bounding box. Finally, the pixel color is stored in the image buffer.

8.2.2 Atom Grid Phase

The atom grid—a uniform grid—helps to detect the entry cell efficiently. It also eases the handling of non-convex grids, like grids with holes, and the partitioning of the domain for distributed rendering. The atom grid resides above the actual simulation grid, as depicted in Figure 8.1. Each atom grid cell links to the cells of the unstructured grid it covers. The edge length of the smallest cell defines the desirable resolution of the atom grid, which is coarsened globally until it fits into graphics memory. For determining the entry cell for a given ray, regardless of whether this is the initial entry cell or whether the grid has been left before, the ray is checked against all cell faces of the current atom grid cell. If there is an intersection, a new cell entry position has been found. Otherwise, the scheme proceeds to the next atom grid cell according to the 3D voxel traversal algorithm [3]. The ray entry search is illustrated in Figure 8.1. Ray r_2 finds its entry cell C_1 in atom $A0$. The intersection of ray r_1 with C_0 and C_1 however are invalid in $A0$. The entry cell C_0 of r_1 is found after traversing to $A1$. The grid traversal phase starts as soon as an entry cell and the corresponding entry point are found.

8.2.3 Grid Traversal Phase

In the grid traversal phase, at first, the exit face of the currently active cell and the corresponding intersection with the ray are determined. Therefore, the ray is intersected with all faces of the cell and the intersection nearest to the entry point is chosen. Then, the monomial representation u_b^m of the active cell is sampled along the ray segment in the barycentric coordinate system. Afterward, the next cell is identified and the algorithm proceeds iteratively. If the ray leaves a cell through a face without an adjacent neighbor cell, the atom grid phase is entered again.

The ray caster can handle hybrid grids that consist of different types of cells, including tetrahedra, hexahedra, prisms and pyramids. Non-conforming cells, where a cell can have more than one neighbor directly accessible through one face of the cell, are also supported. The ray caster, however, expects that faces are shared by two neighbors at most. Hence, non-conforming faces have to be dissected accordingly.

The ray casting system also supports higher-order elements with curved faces. Our data, for example, contains cubic quad and triangle faces that are given in a parametric surface representation. While ray-patch intersection routines are integrated into the kernel only for such patches, an extension to other patch types is straightforward. In general, the intersection of the ray with a cubic patch cannot be calculated analytically. Commonly, an iterative Newton-Raphson solver is employed (see Section 3.6). For this purpose, the ray is represented by two intersecting planes, resulting in a system of two non-linear equations that can be solved for their roots [69]. For fast convergence, a good initial guess of the intersection point is required. Ray tracing methods typically use bounding volume hierarchies to isolate the roots prior to applying the Newton-Raphson scheme. We use object-aligned parallelepipeds for the quad faces and tripipeds for the cubic triangles [12]. For the construction of the bounding box hierarchy, the curved patches are converted to a Bézier representation. Fortunately, our curved faces are only slightly curved. A single bounding volume is usually sufficient to achieve convergence in less than 4 Newton iterations. Patches that deviate more strongly from a planar face can also be handled, typically resulting in a deeper bounding box hierarchy and correspondingly higher intersection costs. If the angle between the ray and the patch becomes too small, the ray may intersect the patch at multiple points. In such cases, the Newton iteration is started at two opposite positions on the patch. Then, the correct intersection point can be isolated [12]. The CUDA kernel expects patches in a monomial representation stored in a 2D texture. The quad faces are cubic tensor-product patches, each requiring 16 three-dimensional coefficient tuples. Each triangle requires 10 tuples.

For the evaluation of the monomial field representation u_b^m, the ray sampling points are transformed to the barycentric coordinate system of the respective cell. The local coordinate is then used to evaluate the polynomials iteratively according to (8.1). The simultaneous evaluation of the field gradient does only require a few extra operations, as $\partial(x^k y^l z^m)/\partial x = k x^{k-1} y^l z^m$. Polynomials of arbitrary degree can be handled easily,

as the coefficients c_i are ordered uniquely in memory according to increasing polynomial degree (see Listing 8.1). Hence, the same set of monomial exponents $\{(k_i, l_i, m_i)\}$ can be used for all cells. Up to a fixed degree p, the exponents are precomputed and stored in fast shared memory. The solution coefficients c_i are also preloaded into fast shared memory prior to sampling. However, as the rays of one CUDA thread block are largely incoherent, each ray thread needs to load the polynomial coefficients of its cell, limiting the number of coefficients that can be cached. This number is determined before executing the kernel. The coefficients that do not fit into shared memory can then be fetched from texture memory during execution.

8.2.4 Adaptive Sampling

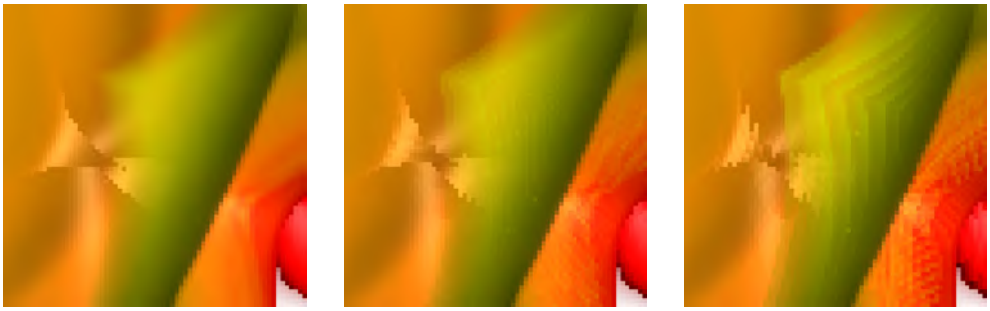
With a conservative choice of a user-defined minimum sampling step size for the whole data set, large parts of the hp-adaptive data get oversampled, leading to a very large increase in runtime. On the other hand, it cannot be guaranteed that with the chosen rate all features are sampled adequately (see Section 5.3). Calculating the required sampling rate is nontrivial, as the function to sample is not known a priori. DVR usually has to deal with a composited color function $k_c(\mathbf{x}) = g \circ u(\mathbf{x})$, which determines the color k_c of a field sample $\mathbf{x}$ by applying a user defined transfer function g to the scalar field sample $u(\mathbf{x})$ (see (5.7)). However, it is not practical to determine the exact sampling frequency each time a sample is taken. Thus, in our system, a semi-conservative sampling strategy on a per-cell basis is applied, sampling each cell with its specific adapted sampling rate. The opacity correction formula (5.15) is used for adapting the opacity values obtained from the transfer function accordingly.

According to Bergner et al. [17], it is possible to analyze the composited signal $k_c = g \circ u$ by looking at both components separately. A good approximation of the local band limiting frequency

$$\nu_{k_c} = \nu_g \max_{\mathbf{x}} |\nabla u(\mathbf{x})| \tag{8.2}$$

can be calculated by taking the maximum frequency ν_g of the transfer function times the local gradient magnitude of the underlying scalar field u. We evaluate ν_{k_c} conservatively on a per-cell basis in a preprocessing step. To obtain the range of field values $[u_{\min}, u_{\max}]$ and the maximum gradient magnitude $|\nabla f|_{\max}$, the polynomial solution of each cell is evaluated on a fine regular grid with a resolution that depends on the degree of the polynomial p, e.g., using $10 \cdot p$ samples along each coordinate axis.

The frequency analysis of the transfer function has to be executed each time the user modifies the color mapping of the scalar field. In our case, the user is able to draw RGBA transfer functions that are stored in an array of size 256. Fast Fourier transform (FFT) is employed to calculate the frequency power spectra of the associated color functions $(R, G, B) \times A$. As cells cover often only a small part $[f_{\min}, f_{\max}]$ of the transfer function range, we are exclusively interested in the frequencies within this narrow window. To account for this, the part of interest is cut out of the transfer function and resampled onto 256 samples prior to applying the FFT. In general, the windowed part of the



(a) $\tau = 1$ (b) $\tau = 5$ (c) $\tau = 10$

Figure 8.4 — Trading rendering quality against responsiveness (crop of Figure 8.5): (a) high quality rendering exhibiting a discontinuity of the DG solution, 6.3 s, (b) 1.2 s, (c) interactive rendering, 0.6 s (Nvidia 285 GTX).

transfer function is nonperiodic. In order to avoid introducing artificial high frequency components at the borders, the transfer function is multiplied by a Hanning window $w(x) = 0.5 - 0.5\cos(2\pi x/d)$, where d is the width of the transfer function range. In frequency space, this results in a convolution of g with the spectrum of the window w, which approximately equals a smoothing with a symmetric Gaussian-like kernel. Because of that, the direct current component of the signal should be removed before applying the windowing function. Another issue is introduced by the hand drawn non-smooth transfer function. To get reasonable results from the frequency analysis, the functions are smoothed with a Gaussian kernel before transforming them to frequency space. The frequency ν_g is then obtained as the highest frequency component of all color channels. For each cell ν_{k_c} is determined and its step size $\delta\lambda = 0.5/\nu_{k_c}$ is calculated. Multiplying $\delta\lambda$ with a step size factor τ allows for controlling the overall rendering quality, as demonstrated in Figure 8.4. Using a factor larger than 1, for example, increases responsiveness during camera interaction.

8.3 Results and Evaluation

The presented ray casting framework allows scientists to visualize data of their higher-order simulations interactively and accurately. Using adaptive sampling with $\tau = 1$ results in high-quality visualizations with render times in the range of 0.2 up to 10 seconds on a stand-alone workstation (Table 8.2). If necessary, the system reduces the rendering quality during camera interaction (by choosing $\tau > 1$) in order to retain interactive frame rates (see Figure 8.4).

The system is evaluated with three CFD data sets. The shock channel (Figure 8.5) is a numerical simulation where a shock with Mach number $Ma = 3$ hits an obstacle positioned in the middle of a channel. As the shock moves over the obstacle, a lifted ballistic wave forms together with reflections of the shock wave on the channel

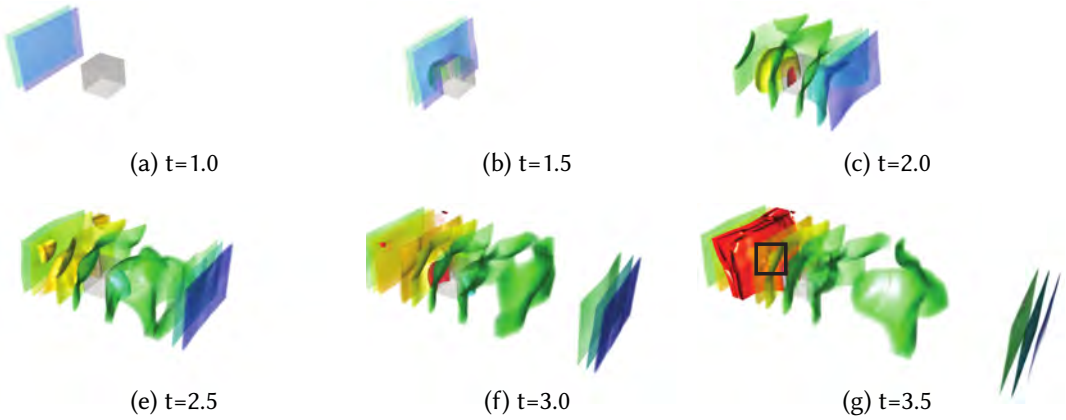


Figure 8.5 — Shock channel sequence of flow around a box. Figure 8.4 shows a close-up of the region marked in (f). The transfer function contains several distinct opacity peaks to highlight thin isovalue intervals, colored according to a rainbow color map.

walls. The handling of shocks as well as a sufficient resolution of the effects can be a challenging task for low-order numerical schemes. This example nicely demonstrates how DG simulations are able to compensate for coarse mesh resolutions by using polynomials of high order. Here, a regular grid of resolution $20 \times 2 \times 3$—the box obstacle in the figure corresponds to one cell—was used together with a polynomial degree of seven. Due to the coarse grid and the high order, the DG solution exhibits some strong discontinuities at the cell boundaries. These can be seen in the close-up views in Figure 8.4.

The sphere test case is a hydrodynamical simulation solving the compressible Navier-Stokes equations. The domain contains a sphere obstacle and is discretized with a highly complex grid using hp-refinement (see Figure 6.2). It contains 581929 degrees of freedom from 35535 cells of different types and polynomial degrees varying between 1 and 5. Initially, a uniform flow with Reynolds number $Re = 300$ and $Ma = 0.3$ is set up. The simulation results in the expected vortex street roll-up, as shown in Figure 8.6. The data set is also used to evaluate the handling of curved elements—1290 curved cubic triangles are located in the direct vicinity of the obstacle. Ray casting curved faces only leads to small improvements, as the curved elements are comparatively small in this data set. For instance, the red high density region touching the front of the sphere better reflects the curved surface of the obstacle (Figure 8.6 (c)). Finally, we generated an artificial data set with arbitrary polynomial degree in order to measure the scaling of the performance when increasing the polynomial degree.

Detailed performance measurements were performed for the channel and the sphere data set for three different GPU architectures from Nvidia. This includes a GPU with the Tesla architecture (285 GTX, 2009), the Fermi architecture (465 GTX, 2010), and the current Kepler architecture (670 GTX, 2012). Single-precision floating point accuracy was used in the kernels. Although three different workstations served as hosts, details

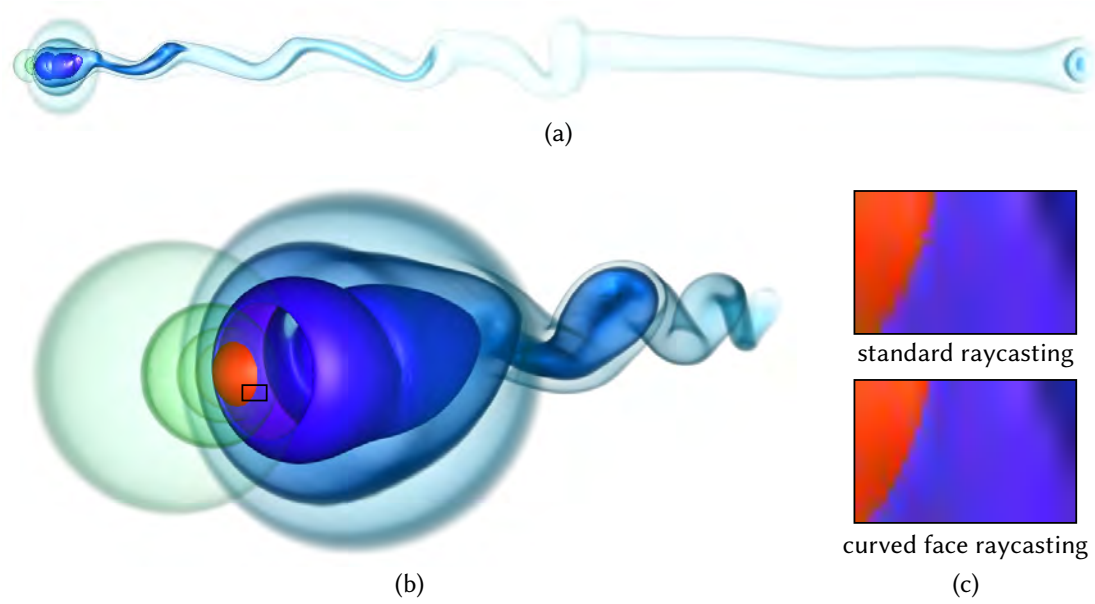


Figure 8.6 — (a) Illuminated DVR of the sphere data set showing the typical vortex street roll-up, with (b) close up. (c) The difference between approximate and accurate ray casting of curved faces is only marginal, as curved faces only exist near the sphere.

on CPU and memory configurations are omitted. The CPU performance and memory bandwidth only have an insignificant impact, as the whole data is located in GPU memory during runtime. Timings are given in Table 8.2. Using the current Kepler GPU, images are computed around four times faster compared to the older Tesla architecture. To demonstrate the importance of adaptive sampling, we also compared our approach to higher-order ray casting with a fixed step size. We used the globally smallest adaptive step size $\delta\lambda_{min}$ as fixed step size to ensure results with at least the same quality. Showing a speed-up of up to 20 with the channel data set, the benefit of adaptive ray casting is much larger for the newer architectures. An explanation for this behavior might be that newer GPUs with their larger caches are less sensitive to incoherence during computation, which, for example, is induced by the varying step sizes. Adaptive step sizes are even more important in the case of the sphere data set, which is characterized by its strongly refined grid. Here, the speed-up is an order of magnitude larger, with fixed step size ray casting taking several minutes to compute an image. Render times increase if curved faces are handled accurately (sphere data set). The impact, however, is comparatively small, as only a small subset of the elements is curved. Again, newer architectures are capable to handle the additional complexity more easily. They exhibit a smaller relative performance decrease. Measurements of the scaling performance with the third data reveal a linear relationship between the number of polynomial coefficients and the render time (Table 8.3).

Table 8.2 — Performance of adaptive and fixed step ray casting in seconds per frame using different GPU architectures from Nvidia. The smallest adaptive step size $\delta\lambda_{min}$ was used as the fixed step size to ensure comparable quality (1280×910 viewport).

	285GTX, Tesla		465GTX, Fermi		670GTX, Kepler		
	adaptive	fixed	adaptive	fixed	adaptive	fixed	$\delta\lambda_{min}$
channel (t = 1.0)	0.56	5.82	0.18	3.31	0.15	2.98	0.0031
channel (t = 2.5)	2.59	11.37	0.78	7.22	0.69	6.44	0.0016
channel (t = 3.5)	3.40	17.49	1.05	8.94	0.92	8.0	0.001
sphere	1.02	>120	0.31	>120	0.25	>120	0.0005
sphere curved	1.26	>120	0.35	>120	0.27	>120	0.0005
sphere zoom	5.91	>120	2.37	>120	2.0	>120	0.0005
sphere zoom c.	7.31	>120	2.47	>120	2.05	>120	0.0005

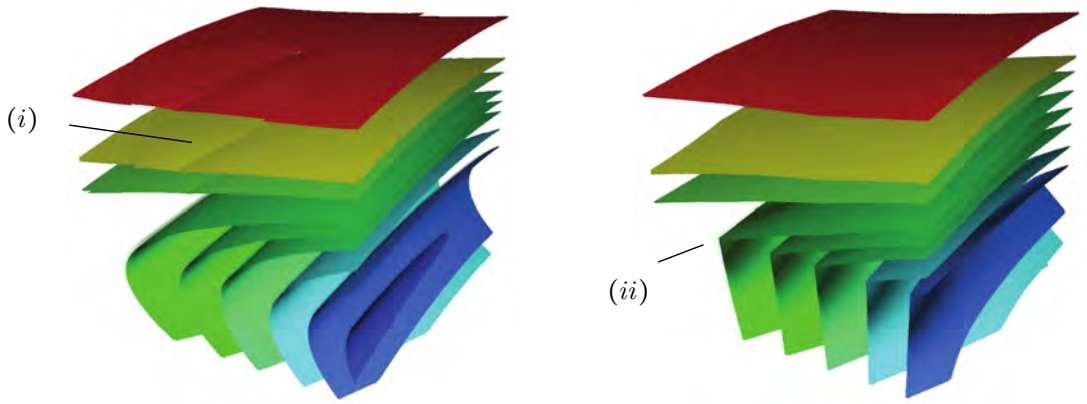
Table 8.3 — Relation of render time to solution degree p exhibiting a roughly linear scaling in the number of monomial basis functions $N(p)$.

p / $N(p)$	2 / 10	3 / 20	4 / 35	5 / 56	6 / 84
t in ms	205	307	476	730	1067

In order to demonstrate the accuracy of the higher-order ray casting system, we resampled the higher-order data onto a regular grid and visualized it with a standard GPU-based ray caster. Figure 8.7 illustrates that even a very fine resampling resolution is by far not high enough to capture all features in the data. In the example, a small region of the sphere data set with an edge length of 0.3 in comparison to a size of 120 for the whole data set was resampled with 4^3 samples. With the chosen resolution, the resampled representation of the complete domain would require more than 14 GBytes of storage for the scalar and the gradient field.

8.4 Distributed Rendering

The previous section demonstrated that higher-order ray casting is very demanding with respect to its computational costs. The frame rates can be increased by distributing the workload across a compute cluster. Existing techniques for low-order data typically use an object-space [8, 116, 129], or image-space partitioning [201] of the problem, or a combination of those two approaches. Load balancing techniques are required in addition to be able to distribute the workload equally among the nodes of the cluster. Achieving optimal utilization, however, is difficult, often hindered by adaptive data and complex transfer functions. Finally, the partial image results computed by the nodes need to be composited. In large cluster environments consisting of a large number



(a) higher-order ray casting (b) ray casting of resampled field

Figure 8.7 — DVR of resampled higher-order fields can easily miss features, like the DG discontinuities at cell boundaries (i), or show misleading structures (ii).

of render nodes parallel compositing schemes are employed to avoid bottlenecks. Our parallel volume rendering system employs kd-trees for object-space partitioning, similar to [129]. To this end, the grid is subdivided into as many partitions as there are GPUs, as illustrated in Figure 8.8. The workload is then distributed by assigning each partition to one backend render node. In the example, GPU 0 ray-casts segment $[\mathbf{x}_0, \mathbf{x}_1]$, GPU 2 ray-casts segment $[\mathbf{x}_1, \mathbf{x}_2]$, and GPU 1 ray-casts segment $[\mathbf{x}_2, \mathbf{x}_3]$ of the ray. In the case of our data, unlike for medical volume data, the performance for data sets that can be visualized in high quality close to interactivity is limited by pure rendering speed only. GPU memory capacity does not play a big role. Hence, we limit ourselves to task distribution and do not consider data distribution, i.e., the whole data set has to fit into the memory of a each single GPU. This avoids data transfers at runtime, potentially reducing an adverse impact on load balancing. Subdividing the data set on cell level would be problematic as this might lead to non-convex partitions, which would make correct compositing impossible. Instead, we use the overlaying uniform atom grid structure that we initially introduced for determining entry cells (see Figure 8.1). The kd-tree partitions the atom grid to convex, non-overlapping groups of atoms called bricks (Figure 8.8). A brick is rendered on the GPU of a backend node in a single pass. The backend nodes also determine the blending order of their corresponding brick by considering the viewpoint. This information is attached to the image before sending it to the dedicated display node, which is called frontend node in the following. For each frame, the frontend node combines the partial image results with standard blending operations, resulting in the final image. The frontend node additionally handles user interaction.

An estimation of the relative rendering cost of the bricks is required for dynamic load balancing during runtime and for obtaining an appropriate initial partitioning. For this purpose, we store one additional value for each atom A that roughly estimates its

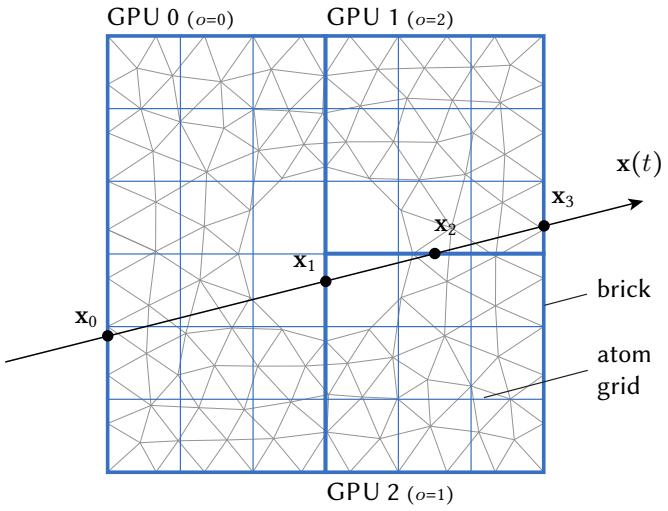


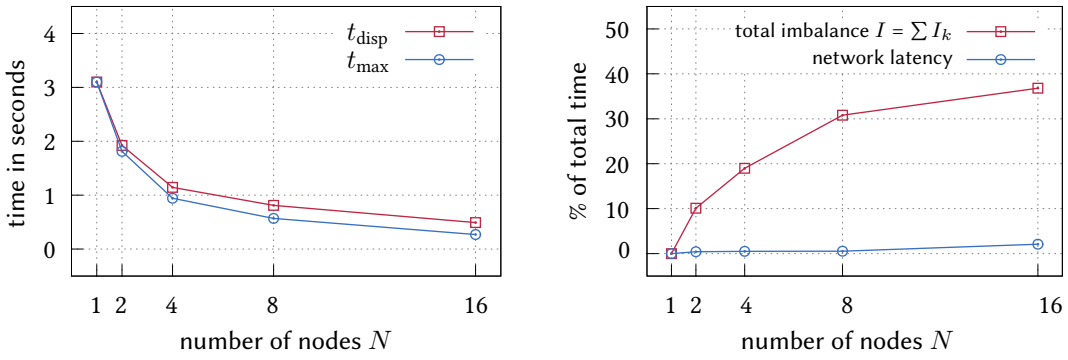
Figure 8.8 — Distributed ray casting partitions the domain into convex bricks aligned with the atom grid data structure. Each render node ray-casts one brick and determines the blending order o of its partial image result.

rendering complexity as the sum of the estimated costs of all cells q_i contained in the atom:

$$w_A = \sum_{i=1}^{N} \mathrm{C}(q_i) \cdot (G(q_i) + \kappa N(\mathrm{degree}(q_i))). \tag{8.3}$$

The estimate w_A accounts for the geometric complexity G of the cells q_i. Cells with curved faces have higher complexity and contribute with a larger value. The evaluation cost of the polynomial solution is incorporated as well. It depends linearly on the number of basis functions N (see Table 8.3). Finally, the contribution of each cell is weighted with respect to the fraction of the cell that is covered by the atom $C(q_i)$. Besides this static complexity measure, the load balancing algorithm additionally considers the rendering time of each brick. The timings measured by the individual backend nodes are therefore sent to all nodes and are then distributed among the atoms relative to their rendering complexity w. The user can manually tweak the performance by adjusting the weighting of the complexity components and the influence of the complexity measure in the rendering time distribution. The estimated costs per atom are used to determine which atoms to add or to remove from a brick and thus influence the workload of a GPU. The partitioning of the atom grid is done using the kd-tree, which is rebalanced every frame. It is traversed top-down and sub-trees are balanced by moving the split plane toward the region with the larger overall estimated render time, stopping when the weights on both sides are as close as possible. The movement of the split plane is slowed down by penalizing large relocations of the split plane to avoid cases in which huge oscillations occur per frame due to an inappropriate complexity estimation.

To evaluate scalability, sixteen cluster nodes were used, each equipped with a NVIDIA GeForce 285 GTX and connected via Gigabit Ethernet. We performed our tests for a sub volume of the sphere data set located near the obstacle, and used an atom grid resolution of $288 \times 312 \times 312$. A series of $N_f = 285$ frames was rendered while moving the camera around the data set and zooming in and out at various speed. The



(a) (b)

Figure 8.9 — (a) Distributed rendering performance on a cluster with $N = 1$ to 16 nodes with a 285 GTX. The total time is measured before (t_{max}) and after compositing (t_{disp}) of the partial image results. (b) Better scaling is hindered due to increasing total imbalance I with N (red). Network latency has a rather small adverse impact (blue).

averaged timing results over all frames are depicted in Figure 8.9 (a). It can be seen that the system overall exhibits good scaling behavior. A speedup of three, for example, is achieved when four render nodes are used. Better scaling for 16 or more nodes, however, is hindered by several effects. As can be seen in the left graph, the total rendering time t_{disp} is close to the maximum ray casting time on the backend nodes t_{max}. This indicates that network traffic only has a small impact on scaling, though it naturally increases with the amount of rendering nodes involved. This is also revealed by the network latency plot in Figure 8.9 (b), which shows the relative difference between the maximum backend render time and the overall time to display.

To analyze the effectiveness of load balancing, we define a measure of render time imbalance for each backend node k

$$I_k = \frac{1}{N_f} \sum_{i=1}^{N_f} \frac{t_i^{\text{max}} - t_i^k}{t_i^{\text{max}}}, \tag{8.4}$$

with $k \in \{1 \dots N\}$. For node k, this measure relates the render times t_i^k of all of its N_f frames to the corresponding maximum backend render times $t_i^{\text{max}} = \max_j t_i^j$. Summing up all I_k results in the total render time imbalance $I = \sum_{i=1}^{N} I_i$ in percent of the total render time. It is plotted in the second graph. The results depict significant load imbalance between the backend nodes, resulting in a lower occupancy. The reason for this is that a uniform grid is used for load balancing. This hinders an efficient load balancing of the adaptive sphere data set, which is very inhomogeneous with respect to the rendering complexity. A small number of atoms cover a significant share of the overall complexity. Hence, as atoms cannot be split, a large indivisible 2D array of atoms is redistributed when the split plane of the kd-tree is moved for one unit. Due

to this, the data set cannot be distributed evenly according to its complexity, leading to significant variations in the rendering times of the backend nodes. It can also be observed that the limited granularity leads to bigger performance impacts for larger N due to the increasing distribution imbalance.

Using a finer task granularity could reduce the imbalance and improve dynamic load balancing [32]. In our case, this could be achieved by using an atom grid of higher resolution, but at the cost of an increased size of this data structure and additional preprocessing. As an alternative, one could subdivide the ray casting of a brick on a dedicated backend node into several tasks. The host processes could then reassign sub tasks to other nodes if the costs of the brick are much higher than expected. However, several factors hinder such an optimization. First, the network traffic and latency is expected to increase because of the larger number of partial image results and the communication overhead. Moreover, subdivision is constrained to partitions that guarantee that the result images can be composited in view direction order. Finally, it should not be neglected that each relocation comes with the overhead of entry cell detection.

8.5 Visual Debugging

While adaptivity is necessary to handle complex fluid phenomena, it introduces additional complexity into the development of simulation schemes like the DG method. Interactive visualization of solver properties can alleviate the debugging process. It can help in detecting problematic regions where the adaptation algorithm fails [37]. One example which was already discussed in the PPUM context are visualizations of the polynomial order and the grid refinement, like it is shown on a cross section in Figure 6.2. Volume rendering techniques could be used to map these properties to color in the whole 3D domain. A property specific to the DG simulation method are the discontinuities at the cell boundaries, which should decrease as the simulation converges. In the following, a simple visualization is presented that supports the investigation of the discontinuities in the spatial context of the adaptive grid.[2] It visualizes discontinuity magnitude by rendering the cell boundaries and mapping the difference between the adjacent field approximations to color. An example is shown in Figure 8.10 using the channel data set and a rainbow transfer function to map zero discontinuity to black and the highest discontinuities to red color. This visualization reveals that the formation of the shock wave introduces larger discontinuities, which then follow downstream along with the shock wave front. These visualization are not restricted to planar geometries, the discontinuity can also be visualized on an arbitrary subset of the grid faces. The restriction of the field evaluation to 2D manifolds results in very high display rates comparable to the direct PPUM visualization technique described in

[2] The discontinuity visualization technique was developed by my colleague Ralph Botchen, who has provided some groundwork on visualizing DG simulation data.

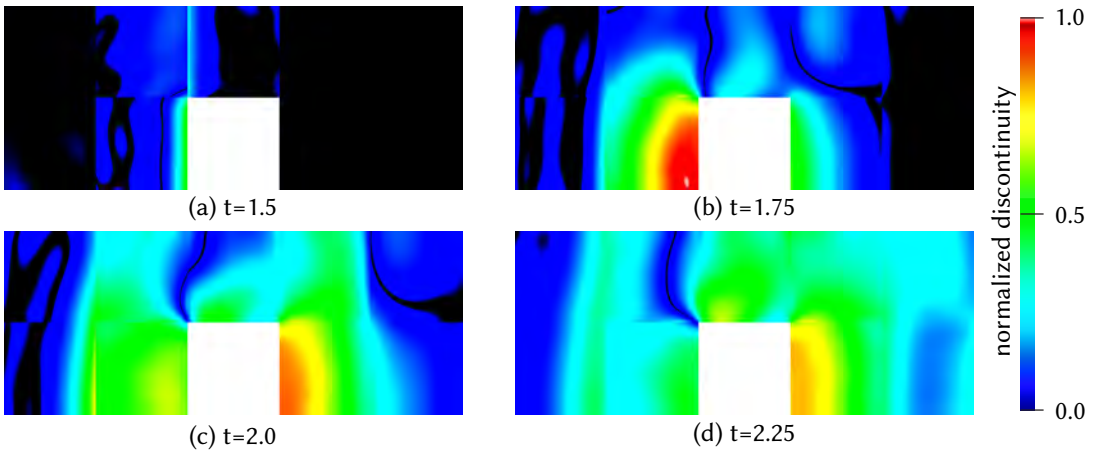


Figure 8.10 — Discontinuity visualization of the shock channel density field on a planar subset of nine quadratic cell faces near the box obstacle (white). With the fluid coming from the left, the discontinuities reflect how the shock front approaches the obstacle (a)–(b) and later advances downstream (c)–(d) (compare with Figure 8.5).

the previous chapter. Our accurate visualization of the polynomials further enables new interesting ways for exploring the field solution and its constitutive basis function. The simulation scientists that developed the DG method, for example, computed a solution of high order, projected it to lower orders, and then compared the differences with our tool in order to investigate the properties of their solution. They also found it useful to be able to visualize individual modes of the polynomials, like the contribution of a subset of basis functions. With resampling, they were not able to perform this analysis because small features of the polynomials were missing even at very high oversampling resolutions.

8.6 Conclusion and Outlook

The higher-order volume rendering system demonstrates the advantage of direct visualization of three-dimensional higher-order data, compared to the traditional approach of resampling. First, the time required to generate a visualization of the DG data is drastically reduced. Second, high-quality rendering results are obtained with an adaptive sampling technique fitted to the hp-adaptive data. Moreover, interactive display rates are achieved by exploiting GPU acceleration on single machines and in cluster environments. The presented ray casting technique is able to handle very high polynomial orders and complex h-adaptive grids. However, there is one requirement concerning the geometry mapping of the FEM data. For our DG simulation, the reference space mapping $\Phi_{n,w}$ is rather simple. Effectively, we were able to project the solution to physical world space coordinates in a preprocessing step, an advantage that our system harnesses. For general FEM data with more complex mappings, extra costs

are introduced during runtime. Liu et al. [110] propose an approximation approach which is able to handle the inversion efficiently during runtime allowing for interactive DVR of small FEM data set (see related work in Section 6.2). Our approach is suitable for more generic (non-DG) FEM data of higher order, as long as the reference space solution can be approximated by a world space projection with error estimates being available. We demonstrated that our monomial world space representation is adequate for interactive visualization up to high orders of seven or eight. While projecting polynomials of even higher degree to monomials could deteriorate the quality, very few simulations use such high orders. For future work, the performance for large cluster environments could be improved by providing a more adaptive structure that better adapts to the inhomogeneous properties of the data set. Additionally, the framework could be easily extended for interactive visualization of the higher-order solution on cutting planes or arbitrary 2D manifolds, providing a ray casting-based alternative to the techniques presented in [132]. High display rates could be achieved with such an approach.

Chapter 9

Toward Simulation-Consistent Interpolation Near Walls

While the simulation side often uses appropriate models to obtain accurate solutions, it is still common practice to transfer the resulting data to the visualization side in terms of discrete values only. For analysis, a continuous field representation is then often reestablished by simply applying tensor-product linear interpolation (Section 3.3), potentially ignoring the underlying simulation models. The higher-order visualization methods presented in the previous chapters account for this shortcoming. They enable a direct and accurate visualization of analytical higher-order field representations, like polynomial solutions, consistent with the simulation that produced them. This chapter presents first steps toward a more accurate visualization of simulations that employ so-called subgrid-scale models. Some CFD simulation methods utilize turbulence models in order to solve the Navier-Stokes equations on comparatively coarse grids, e.g., with the RANS approach. However, as the boundary layer is typically very thin and would require a comparatively high resolution, often high Reynolds number models are used in addition, harnessing analytic *wall functions*, like the *law of the wall*, to model the characteristic near-wall velocity distribution. A brief introduction to near-wall flow behavior and turbulence modeling was given in Section 4.4. There are relatively few visualization techniques that specifically target the analysis of near-wall flow. Petz et al. [150], for example, presented a method that flattens boundary geometry to alleviate the analysis near curved walls. Further, there are techniques that investigate flow separation [93], topology [103], and generation of vortices [209, 158]. These methods, however, do not account for wall functions. The following advocates the development of visualization techniques that are more consistent with simulation models. After describing the law of the wall, a new interpolation scheme consistent with wall functions is presented.

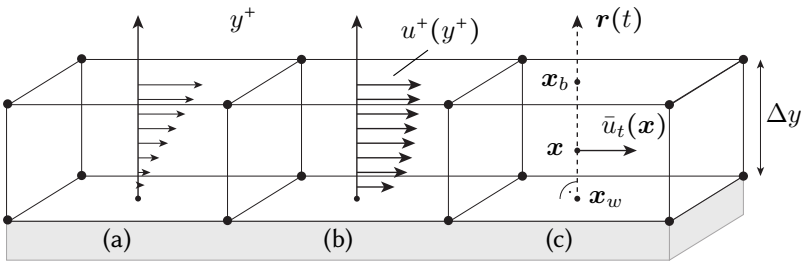


Figure 9.1 — Boundary cell layer with wall (gray) and bilinear face layer (faces opposite to wall). Trilinear interpolation (a) within the boundary layer is a poor approximation of (b), the velocity profile according to the law of the wall. (c) Wall function evaluation.

9.1 The Law of the Wall

The law of the wall models the Reynolds averaged wall-tangential velocity component $\bar{u}_t$ of a fluid flow near no-slip boundaries at high Reynolds numbers [167]. The relationship between $\bar{u}_t$ and the wall distance y is typically given by the respective dimensionless variables

$$u^+ = \frac{\bar{u}_t}{u_\tau} \qquad (9.1) \qquad \text{and} \qquad y^+ = \frac{\varrho u_\tau y}{\mu} \qquad (9.2)$$

with friction velocity $u_\tau = \sqrt{|\tau_w|/\varrho}$, wall shear stress τ_w, density ϱ, and dynamic viscosity μ. A linear relationship $u^+ = y^+$ can be observed in the direct proximity of the wall within $y^+ \leq 5$, where viscous effects dominate. This region is called the *viscous sublayer*. A transitional buffer layer ($5 \leq y^+ \leq 30$) connects the linear relationship with the logarithmic relationship

$$u^+ = \frac{1}{\kappa} \ln(y^+) + C \qquad (9.3)$$

observed in the turbulent *log law layer* ($30 \leq y^+ \leq 300$). The logarithmic relationship includes the von Kármán constant $\kappa = 0.41$ and a constant C that reflects the dimensionless thickness of the viscous sublayer, e.g., $C = 5.5$ for a smooth planar wall. Popular turbulence models, like the $k - \epsilon$ model, are based on the turbulent kinetic energy k, which describes the kinetic energy of velocity fluctuations. Many solvers that use such models apply the relationship [106]

$$u_\tau = C_\mu^{1/4} \sqrt{k}, \qquad (9.4)$$

with constant $C_\mu = 0.09$ together with (9.3) to compute the friction velocity and then derive the wall shear stress from the field values k, $\bar{u}_t$, etc. at the first grid nodes off the no-slip boundary. Consistently, the simulation uses the wall function only within the layer of cells that is in contact with the no-slip boundary. In simulations using unstructured grids, near-wall regions are preferably discretized with several layers of

hexahedra, as illustrated in Figure 9.1. If a wall function is employed, the first off-wall nodes are required to lie within the log layer, i.e., where $30 \leq y^+ \leq 300$.

9.2 Interpolation Scheme

Near walls, the domain is typically discretized with one or more layers of hexahedra, as depicted in Figure 9.1. Consistent with the simulation, we evaluate the wall function only within the layer of boundary cells adjacent to no-slip boundaries, which we call the *boundary cell layer*. The figure illustrates that the velocity profile obtained with trilinear interpolation (a) is a poor approximation of the logarithmic profile observed near the wall in experiments (b). We define the *bilinear face layer* to consist of all off-wall faces of the boundary cell layer. It connects the boundary cell layer with the interior of the domain, where the traditional interpolation scheme is employed (see Figure 9.2 (b)). In the following, we describe how the law of the wall (in particular Equation (9.3)) is evaluated at a point $\boldsymbol{x}$ within the boundary cell layer, i.e., how it is incorporated in the original interpolation scheme (cf. Figure 9.1 (c)). First, the dimensionless distance from the wall $y^+(\boldsymbol{x})$ is obtained. To this end, the point $\boldsymbol{x}_w$ on the wall nearest to $\boldsymbol{x}$ is determined, and a ray $\boldsymbol{r}(t) = \boldsymbol{x}_w + t(\boldsymbol{x} - \boldsymbol{x}_w)$ is intersected with the bilinear face layer, resulting in the point $\boldsymbol{x}_b$. The traditional 2D interpolation scheme is then used within the "bilinear" cell face that contains $\boldsymbol{x}_b$ to interpolate the required field values at $\boldsymbol{x}_b$, including $\boldsymbol{u}$, y^+, and k. Common solvers, like CFX [5], provide all necessary quantities, such as k, μ, ϱ, and $y+$ at the first off-wall nodes. Note that in our application the traditional 3D interpolation is trilinear and hence the respective 2D interpolation is bilinear. However, other interpolation schemes would likewise fit into our approach. Evaluating (9.4) with the interpolated turbulent kinetic energy $k(\boldsymbol{x}_b)$ yields the friction velocity u_τ. With this, and the normal distance $y(\boldsymbol{x})$, the dimensionless distance $y^+(\boldsymbol{x})$ is computed according to (9.2). Then, the log law (9.3) can be evaluated to obtain $u^+(\boldsymbol{x})$. Special treatment is required if $\boldsymbol{x}$ lies within the viscous sublayer, where $y^+(\boldsymbol{x}) < y_v^+$. We chose a constant threshold of $y_v^+ = 11.06$, consistent with the solver [6], and employ linear interpolation in these cases:

$$u_v^+(\boldsymbol{x}) = \frac{y^+(\boldsymbol{x})}{y_v^+} u^+(y_v^+) \tag{9.5}$$

in order to bridge the buffer layer gap and ensure C^0 continuity at the boundary distance y_v^+. Transforming u^+ with (9.1) results in the tangential velocity $\bar{u}_t(\boldsymbol{x})$. While the simulation enforces zero velocity at the no-slip walls, the first off-wall nodes can exhibit a velocity component $\bar{u}_\perp(\boldsymbol{x})$ orthogonal to the wall. Since the law of the wall assumes planar walls and wall-tangential flow, the (simulation) model does not apply in such cases. Employing our approach only for $\bar{u}_t(\boldsymbol{x})$ and interpolating $\bar{u}_\perp(\boldsymbol{x})$ with the traditional 1D scheme, in our case linear, however, provided poor results: streamlines did not detach correctly at the step in Figure 9.2 (c). Interpolating $\bar{u}_\perp(\boldsymbol{x})$

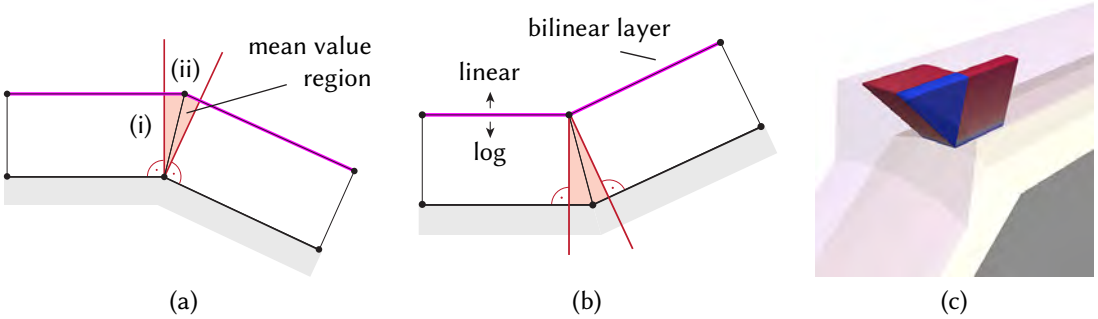


Figure 9.2 — Convex (a) and concave (b) boundaries with mean value regions (red) and the bilinear face layer (pink). (c) Three-dimensional example showing two mean value regions induced by two convex edges (the blue to red color mapping exhibits the logarithmic velocity profile) and one by a vertex (blue).

according to $\bar{u}_t(\boldsymbol{x})$ provided much better results. Hence, our interpolation approach is $\bar{\boldsymbol{u}}(\boldsymbol{x}) = \bar{u}_t(\boldsymbol{x}) \cdot \bar{\boldsymbol{u}}(\boldsymbol{x}_b)/\bar{u}_t(\boldsymbol{x}_b)$. A detailed examination is left as future work.

The interpolation scheme as described so far can only be applied if the simulation mesh provides a well-defined wall normal at $\boldsymbol{x}_w$ since the wall function models wall-tangential flow. It cannot handle cases where $\boldsymbol{x}_w$ lies on an edge or on a vertex of the wall faces, and if these faces are not coplanar there. Such a case is shown in Figure 9.2 (a), where the wall exhibits a convex configuration at one boundary vertex. Another so far unsupported case are multiple points (candidates for $\boldsymbol{x}_w$) on the wall where the normal points to the sample point under consideration, see Figure 9.2 (b). Both cases result in unsupported ambiguous regions (red in Figure 9.2), within which we use mean value interpolation to ensure C^0 continuity of our overall interpolation approach. Mean value interpolation obtains an interpolation within a volume from values on a triangular mesh that encloses the volume [85]. These "red" regions are henceforth denoted as *mean value regions*. Each "nonplanar" vertex or edge on the no-slip wall induces one mean value region, as illustrated with the curved channel example in Figure 9.2 (c). The surface representation of each mean value region (the closed triangle mesh) is constructed in a preprocessing step, using the CGAL library [31] by cutting the involved cells with the planes indicated by the red lines in Figure 9.2 and subsequent merging of the "red" parts. The faces of the resulting polyhedron that were obtained by cutting with the aforementioned planes have to fit our wall function consistent interpolation scheme and thus have to be, due to its nonlinearity, subdivided and the values at the resulting vertices set by our new interpolation scheme. Triangles of the polyhedron lying within the bilinear layer are also subdivided, using the traditional scheme to compute the field values. Our overall interpolation approach is evaluated at runtime within a specific mean value region by accessing its precomputed surface mesh together with the values at its vertices and applying mean value interpolation.

Currently, our prototype only supports convex mean values regions. Concave mean value regions, like in Figure 9.2 (b), are not supported, yet. The main reason for this are the difficulties in obtaining the mean value regions if concave boundary edges or vertices are involved. While it is straightforward to obtain the mean value regions for convex edge or vertex wall regions, it turned out to be intricate to obtain them for concave or concave/convex cases. This is because those mean value regions can intersect, be oblique, consist of several parts, or even intersect at remote regions in distorted grids. While our approach so far supports any unstructured grid consisting of hexahedra as long as the boundary is convex, we support concave wall regions in our current implementation only if the involved cells are rectangular with the following workaround: we avoid the determination of the mean value regions involving concave wall edges or vertices and instead treat the whole cell that exhibits a concave edge or vertex as a mean value region.[1] This workaround is better than traditional trilinear interpolation in these regions but does not employ the wall function to its full extent. Finding such a formulation for generic configurations is, from our experience, nontrivial and future research in this direction is required.

9.3 Results and Evaluation

The log law interpolation approach is evaluated at the example of a channel flow with a step computed with a RANS solver in CFX (Figure 9.3). The case investigates the stationary solution of an incompressible fluid at high Reynolds numbers near no-slip walls. The domain was discretized with a regular grid, with the flow entering on the left and leaving the domain on the right. Symmetry boundary conditions were used at the remaining sides. The $k-\epsilon$ turbulence model was employed with a wall function. Details are given in [6]. Using local field evaluations and streamlines, our interpolation scheme (log) is compared to traditional trilinear interpolation on the same grid and a reference simulation of four times higher resolution (hi-res). In the figure, the bilinear face layer is represented by the transparent pink surface, demarking the interior from the region where wall functions are applied. Figure 9.3 (b) compares velocity profiles near the step. Trilinear interpolation shows an unnatural cutoff within the wall cells, compared to the profile obtained from the simulation of four times higher resolution. Our approach (log), in contrast, reproduces the logarithmic flow behavior near the wall, even with the chosen coarse grid resolution. We also computed streamlines traversing the boundary region, all with equal integration time. The close-ups in Figure 9.3 (a) show that trilinear interpolation strongly underestimates the velocity near the wall. The wall-nearest line, for example, already ends in close-up (i). The underestimation of trilinear interpolation is also visible at the corresponding line in the hi-res field. It also ends early, before our lines, see close-up (iii). Note that the vertical deviation to the "hi-res lines" is also caused by deviations between the hi-res and low-res simulations.

[1] Due to our restriction on rectangular grids in concave wall regions there is exactly one cell that is adjacent to a concave edge or vertex.

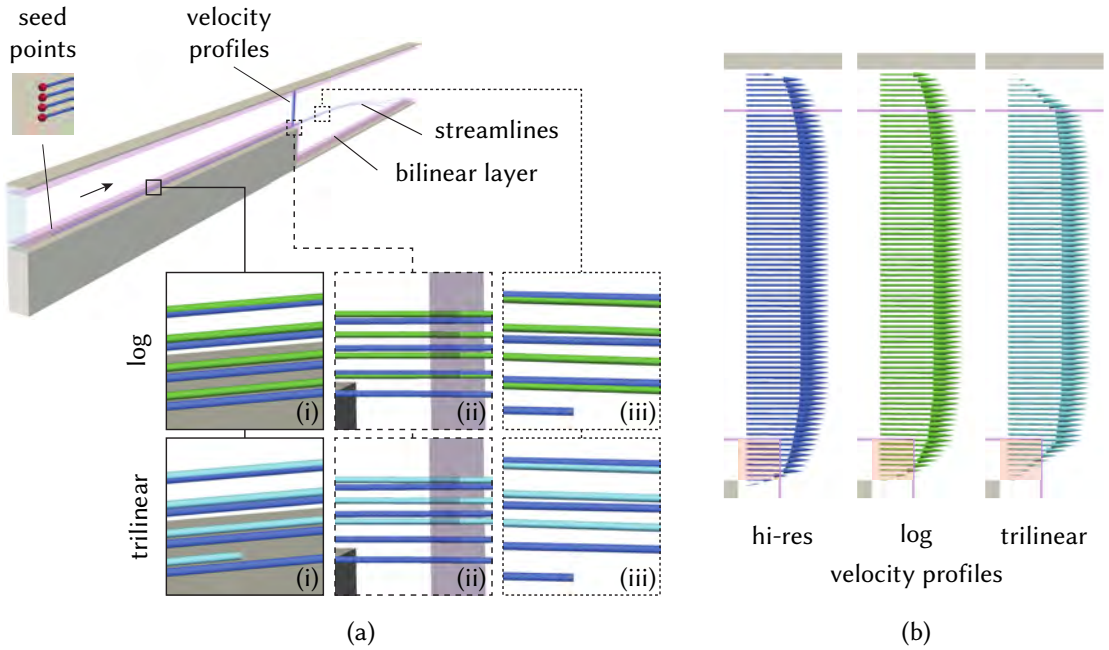


Figure 9.3 — Channel flow with a step. (a) Overview with the bilinear layer (pink), separating the log region near the wall from the trilinear region in the interior. Closeups (i)–(iii) compare streamlines using trilinear (light blue) and log interpolation (green) with streamlines from 4× hi-res field (blue). (b) Comparison of velocity profiles, showing both, mean value region (red) and log region (top).

Region (ii) demonstrates that our mean value interpolation approach does not introduce additional errors.

Our single-threaded CPU implementation of the interpolation schemes take 4 ms (trilinear), 23 ms (log law), and 1594 ms (mean value), for 1000 evaluation points, on an Intel Xeon X5550. While the overhead of log interpolation is acceptable, the mean value interpolation takes much longer. It directly depends on the chosen resolution of the mean value mesh. Here, we chose a mean value region of the first data set consisting of 1162 vertices and 2052 triangles. To improve the timings, one could resample the mean value regions in a preprocessing step, using a fine rectilinear grid, and then use trilinear interpolation at runtime. The same could be done with the log regions, using refinement when approaching the wall. However, while resampling offers a more generic approach for improved visualization of logarithmic wall behavior with standard visualization software, it can only provide an approximation of the wall function.

9.4 Conclusion

The presented interpolation technique accounts for wall functions modeling the law of the wall, which is harnessed by many CFD solvers. We demonstrated that our approach

leads to results that are more consistent with the solver model than traditional trilinear interpolation near walls and that this allows for more consistent analysis. While the $k-\epsilon$ turbulence model was used in our examples, the technique is in principle applicable to all simulations based on wall functions. For $k-\omega$ or Reynolds stress models this is often the case. Our work demonstrates the potential of making visualization techniques more consistent with the respective simulation models. It is clear that this work is only a first step toward wall function consistent interpolation and we hope to trigger future work and awareness on the topic.

Part III

Integral Curves for Feature-Based Flow Visualization

Chapter 10

Introduction to Feature-Based Visualization

The analysis of large time-dependent vector fields is a challenging task, even if the data is given on a 2D domain. Direct visualization of the field values, for example, with the techniques presented in Part II, is often not sufficient for gaining a profound understanding of complex turbulent flow structures (Section 4.4). In the case of 3D fields, visual clutter and occlusion issues can additionally hinder an efficient analysis. Feature extraction techniques provide a means to alleviate these issues. They extract a more compact, condensed representation from the raw data in an automatic or semi-automatic manner. These representations typically try to capture the prominent characteristics of the data.

This chapter bridges the gap between the direct visualization methods of Part II and the vector field topology concepts introduced in Part IV. First, the term *feature* is defined, and several examples of flow features are given (Section 10.1). Then, an interactive visualization technique based on feature concepts is presented in Chapter 11. The approach uses feature surfaces and flow features on them to guide the seeding of integral lines in 3D flow. Vortices and other features can for example be captured with this interactive technique. Interactive visualization of a large number of integral curves is achieved with a novel GPGPU algorithm utilizing OpenGL geometry shaders (Chapter 12). Using a large number of integral lines to visualize features can still lead to significant visual clutter, even with our feature-guided seeding approach. Our new visualization methods for finite-time Lyapunov exponent (FTLE) fields account for this shortcoming (Chapter 13). The FTLE aggregates the information of multiple integral curves to measure diverging particle motion. The regions of strongest separation convey essential information about the global structure of unsteady flow. They can be extracted as ridge features from the FTLE field. Section 13.1 introduces the basic FTLE definition and presents a framework for FTLE computation in time-dependent flow. This is followed by two novel interactive visualization techniques for the analysis

of 2D FTLE fields (Section 13.3). Finally, methods for improving the quality of FTLE fields are covered (Section 13.4). All this paves the way for our new topology concept for time-dependent 3D vector fields (Part IV). This concept is based on FTLE ridges that are used to obtain distinguished hyperbolic flow structures, which are then used to seed streak manifolds that represent a main component of time-dependent vector field topology.

10.1 Flow Features

Flow features can be defined as regions of the domain with a characteristic flow behavior that is important in the context of a particular problem. Flow features include laminar and turbulent flow behavior, vortices, shock waves, boundary layers, regions of flow separation and attachment, and others. While sophisticated algorithms are required to capture these structures with integral curves and surfaces seeded in the whole domain (see Section 5.4), feature-based algorithms attempt to extract and track features directly. Abstraction from the raw data is the main goal of these techniques. Often, the visualization of a few prominent flow features is sufficient to understand the main characteristics of a flow, like in the example shown in Figure 4.4. Moreover, for very large data, the reduction of the raw data to its essential structure might even be necessary in the first place to enable interactive exploration.

Many feature-based algorithms compute derived fields involving the *Jacobian*, i.e., the gradient of the velocity field

$$J(\mathbf{u}) = \nabla\mathbf{u} = \begin{pmatrix} \frac{\partial u_x}{\partial x} & \frac{\partial u_x}{\partial y} & \frac{\partial u_x}{\partial z} \\ \frac{\partial u_y}{\partial x} & \frac{\partial u_y}{\partial y} & \frac{\partial u_y}{\partial z} \\ \frac{\partial u_z}{\partial x} & \frac{\partial u_z}{\partial y} & \frac{\partial u_z}{\partial z} \end{pmatrix}. \tag{10.1}$$

The Jacobian provides a first-order approximation of the local flow behavior with respect to a Taylor expansion of $\mathbf{u}$. Some approaches also require the availability of the gradient ∇s and the *Hessian* of a related scalar field s. The Hessian

$$H(s) = J(\nabla s) = \begin{pmatrix} \frac{\partial^2 f}{\partial x x} & \frac{\partial^2 f}{\partial x y} & \frac{\partial^2 f}{\partial x z} \\ \frac{\partial^2 f}{\partial y x} & \frac{\partial^2 f}{\partial y y} & \frac{\partial^2 f}{\partial y z} \\ \frac{\partial^2 f}{\partial z x} & \frac{\partial^2 f}{\partial z y} & \frac{\partial^2 f}{\partial z z} \end{pmatrix} \tag{10.2}$$

is composed of the second partial derivatives and represents the "curvature" with respect to Cartesian coordinates. Eigen analysis is one example of a common operation performed on these matrices. It is noteworthy that reconstructing derivatives of high order from discretized fields might be problematic if the selected feature algorithms are sensitive to noise.

A multitude of feature-based approaches have been devised, and they cannot all be covered in this work. An overview, including methods for extraction and tracking

of features in time-dependent fields, is given in [153]. While *local methods* employ local flow information for feature extraction, like the velocity gradient, *global methods* harness global flow properties, e.g., obtained by means of integral curves. Only *Galilean-invariant* approaches result in the same set of features regardless of the chosen frame of reference. This is a desirable property in particular in the case of time-varying phenomena, where the addition of a constant velocity to the whole field could lead to different results. Because of this, Galilean-invariant methods usually rely on relative quantities like the Jacobian of the velocity field instead of using the velocity directly.

Separation and Attachment Lines

Separation and attachment lines are distinguished feature lines in vector fields that other trajectories approach to or diverge from asymptotically in forward or backward time, respectively. They are important because they indicate flow separation and attachment on boundary surfaces (Section 4.4). Kenwright proposed a criterion for their extraction [92, 93]. Peikert and Roth later introduced the "parallel vectors" operator to the visualization community [147]. It provides a framework that allows for the extraction of a wide range of line-type features in 3D domains, including separation, attachment, and vortex core lines, as well as 1D ridges and valleys.

Height Ridges and Valleys

Local maxima and minima play an important role in the analysis of functions $f : \mathbb{R}^n \to \mathbb{R}$. They are classified by the signs of the second derivatives of f and require a vanishing first derivative. If the second derivative also vanishes, a change in sign of the first derivative is a more robust indicator. Eberly introduced *height ridges* and *valleys* as generalizations of local maxima and minima embedded in n-dimensional space [40]. A height ridge of dimension d is defined as a manifold that exhibits a local maximum in $n - d$ directions, with n being the dimension of the domain and $0 < d < n$. In 3D scalar fields this generalization enables 1-dimensional *ridge lines* and 2-dimensional *ridge surfaces*. One example for 1D ridges are vortex core lines which can be extracted as ridges of λ_2 fields [162], or as valleys of pressure fields [124]. Analogous to the special case of univariate functions, testing the existence of a d-dimensional ridge involves the gradient and the Hessian of the scalar field s. In the following, the eigenvectors ϵ_i of the Hessian $H(s)$ correspond to eigenvalues λ_i, sorted in descending order $\lambda_1 > \ldots > \lambda_n$. For a d-dimensional ridge the d "largest" eigenvectors are oriented along with the manifold. Hence, the remaining $n - d$ eigenvectors ϵ_k ($k = d + 1, \ldots, n$) are perpendicular to the manifold by concept. This is stated by Eberly's ridge criterion

$$\epsilon_k \cdot \nabla s = 0 \quad k = d + 1, \ldots, n, \tag{10.3}$$

which additionally requires $\lambda_k < 0$ for the corresponding eigenvalues. Valleys, the generalized counterpart to minima, are obtained as height ridges by inverting the sign of the scalar field.

One-dimensional ridges embedded in an n-dimensional domain can be extracted with the "parallel vectors" approach [147]. While this method does not require an explicit computation of the eigenvectors it cannot be used for ridges of higher dimension. The marching ridges algorithm is a common choice for such cases [61]. Similar to the marching cubes algorithm for isosurface extraction (Section 5.1), a 2-dimensional ridge is constructed locally on a per-cell basis. To this end, the eigenvectors have to be oriented consistently at the vertices of the cell to be able to check the cell edges for zero crossings of equation (10.3), as well as for the negative eigenvalue criterion. Then, a marching cubes lookup table is used to obtain a triangulation of the ridge surface. In contrast to isosurfaces, ridges are generally non-orientable and feature a more complex geometry and topology [170]. This makes their extraction more complex, often requiring a very fine sampling resolution to avoid gaps and obtain high-quality manifolds [156]. In the context of this thesis, the global structure of time-dependent vector fields is derived from ridges in finite-time Lyapunov exponent fields (Chapter 13) using the marching ridges scheme.

Vortex Extraction

Many different feature criteria have been proposed in the area of *vortex extraction* [82]. This is partly due to the lack of a rigorous vortex definition. Vortices are characterized by instantaneous streamlines swirling around their core. Moreover, vortices are often accompanied by a local pressure minimum. However, in general the existence of a pressure minimum is not a robust indicator. Unsteady irrotational straining, for example, can induce a minimum in regions without a vortex. Moreover, due to viscous flow effects, pressure minima can be eliminated in vortical regions. Vortical structures are typically subject to movement in time-varying flow, requiring a Galilean-invariant approach for their extraction. The local physically motivated λ_2 vortex criterion offers such an Galilean-invariant approach [82]. It is motivated by the existence of a pressure minimum, but accounts for the aforementioned misleading cases. Therefore, an expression of the Hessian $H(p)$ of the pressure field p is derived from the velocity field of the incompressible Navier-Stokes equations (4.10a) by applying the gradient operator. Then, several terms that might hinder robust vortex detection are removed, resulting in $H(p) = P = S^2 + \Omega^2$, which involves the symmetric part S and the asymmetric part Ω of $\nabla\mathbf{u}$. Vortices are detected as regions where the middle eigenvalue λ_2 of P is smaller than zero. It is noteworthy that the criterion ignores the gradient-based restriction (10.3) used by valley extraction. Often, geometrical vortex representations are extracted from the λ_2 field and visualized (Figure 11.4), like isosurfaces or valley lines (vortex core lines) of the λ_2 field [164].

Chapter 11

Surface-Guided Integral Lines

Visualizing 3D flow fields with a large number of integral lines intrinsically suffers from problems of clutter and occlusion. A common practice to alleviate these issues is to restrict the visualization to surfaces that capture important field characteristics, however, at the cost of loosing information due to the projection of the vector field. In this chapter, we introduce surface-guided integral lines to complement dense 2D vector field visualization on curved surfaces (Section 11.1). A pixel-accurate representation of the surfaces in image space (Section 11.2) is used for image-based seeding strategies (Section 11.3). This allows to achieve an importance-driven distribution of seed points on the surface that adapts to local field properties. Interactive exploration is enabled by using efficient data structures and algorithms on GPUs. Section 11.4 demonstrates our approach with the example of streamline visualizations, using our geometry shader algorithm for fast streamline computation (Chapter 12). Besides streamlines, path lines and streak lines could also be used together with the described approach.[1]

11.1 Motivation and Related Work

Feature surfaces can be derived directly from the flow field, like stream surfaces or isosurfaces of velocity magnitude. On the other hand, they can also be constructed from more complex derived scalar quantities, like the λ_2 field or from associated field variables such as density, temperature, or pressure. Such surfaces typically represent important features the user is interested in. Figure 11.1, for example, shows the visualization of a two-phase-mixture simulation [163], using the separation layer as the natural selection of a feature surface. Here, a LIC representation is used to visualize the projected flow field on the surface. Restricting the visualization to the surface has the advantage that the 3D occlusion problem is reduced to the self-occlusion of the selected surface, and the challenge of visualizing the 3D vector field is reduced

[1] Parts of this chapter have previously been published in Üffinger et al. [195]. The presented seeding technique builds upon the groundwork laid in my diploma thesis [193].

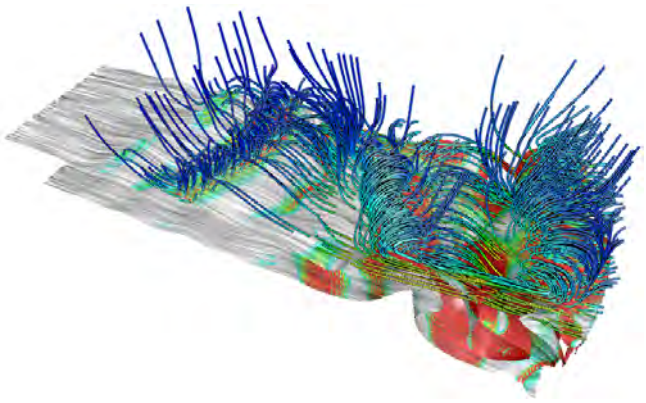


Figure 11.1 — A feature surface representing the separation layer in a two-phase flow. Vortices are captured with streamlines seeded in surface areas with high vorticity (red).

to a 2D domain. On the other hand, the approach suffers from information loss. The data domain is reduced to the chosen surface and—since the surface is not a stream surface—the original 3D flow data needs to be projected onto the 2D surface, making it difficult to conceive the true three-dimensional behavior of the flow field.

To provide more detailed insight into the flow field in the vicinity of the surface, we propose to combine 2D surface-based techniques with 3D integral line visualization. The seeding of the integral lines is guided by the surface itself. This includes the restriction of the seed points to the surface to alleviate the 3D seeding problem. Additionally, to emphasize the interesting flow features even more, the seeding space can be further narrowed to regions featuring certain flow properties. In the aforementioned example, the local property of high vorticity magnitude was chosen as seeding criterion. This allows to capture the vortical flow behavior close to the surface (see Figure 11.1). Interactivity is the key factor in the exploration of unknown data fields. Therefore, we present a solution that accomplishes the visualization of the feature surface as well as the integral line seeding and integration completely on the GPU. This avoids bottlenecks caused by additional data transfer to and from the graphics processor.

Our surface seeding approach is related to multi field visualization [59]. It allows to analyze correlations between a vector field and one or more associated field quantities. For our visualization, we adopt an idea of Max et al. [120], who used short geometric lines of fixed length (hairs) to visualize the flow in the direct vicinity of a surface. While their seed points are distributed over the whole surface, our seeding additionally adapts to local field properties. A related approach for distributing streamlines uniformly in screen-space was proposed by Li and Shen [108].

11.2 Image Space Representation of Feature Surfaces

The presented technique visualizes 3D flow on and near feature surfaces which are used to guide the seeding process described in the next section. By working on rasterized surface data projected onto the image plane, the algorithms do not depend on a specific surface parameterization, i.e., all renderable surfaces can be used as input. In the

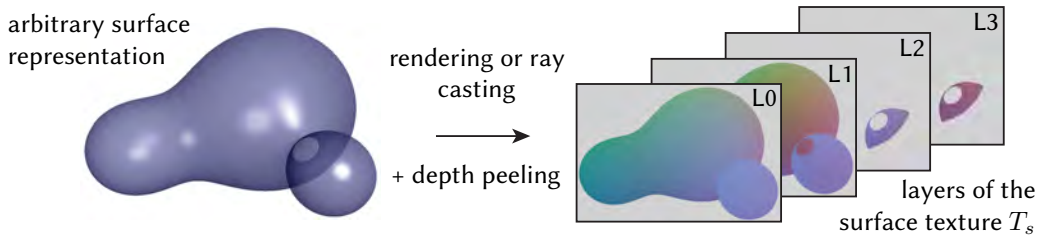


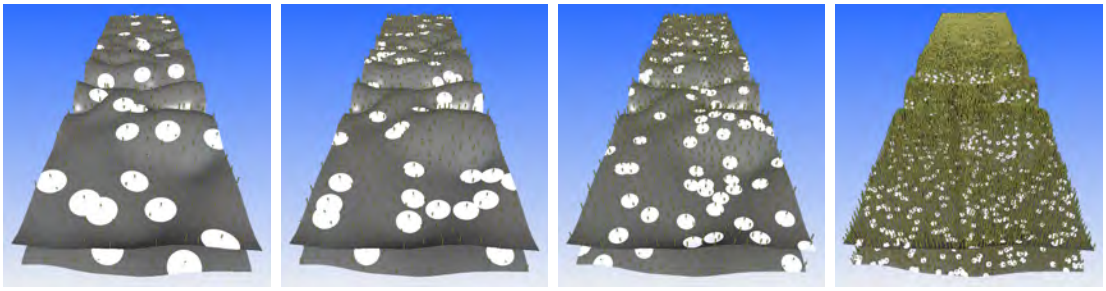
Figure 11.2 — The surface-guided GPU streamline seeding technique works on projected surface data that is extracted and stored in image space.

following, we concentrate on feature surfaces defined by isosurfaces of 3D scalar fields. A GPU-based ray casting algorithm for structured grids [180] is used to enable an interactive extraction of the image space surface representation. Using a depth-peeling approach [131], multiple depth layers of the surface are obtained, as illustrated in Figure 11.2. As an alternative, the ray casting technique for unstructured grids described in Chapter 8 could also be used. The extracted *surface texture* representation stores the depth layers in a 2D texture array T_s, containing positions and normals of ray-surface intersections as well as optional field values associated with these intersections. The extracted surface layers allow for a combined transparent visualization of the surface and the streamline geometry by rendering the opaque lines first and blending the shaded surface layers on top. In the case of a static surface, the extraction has to be executed only once for each isovalue or after each view change.

11.3 Image Space Seeding Algorithm on the GPU

The image space seeding strategy works on the extracted surface texture T_s introduced in the previous section. The basic idea is to extract a subset of the surface texels and use the surface information stored in T_s to generate seed points. The choice of appropriate texels is importance-driven with respect to the local properties of the feature surface as well as the flow field and other available quantities. Based on such properties, a binary predicate P can be specified by the user which is evaluated for every surface texel contained in T_s. This partitions the surface texels into a set of candidate texels fulfilling the criterion and the complementary set, which is of no further interest. The candidate texels are then compactified using the histogram-pyramid GPU algorithm of Ziegler et al. [222], resulting in the list of seed point candidates.

The number of candidates is further thinned out in order to avoid that streamlines start too close to each other. Simply choosing random points from the candidate list, however, does not deliver useful results, as distant texels cover a larger surface area due to the perspective projection. Hence, the probability of missing important flow features in distant regions of the surface would be higher than in parts of the surface near to



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

Figure 11.3 — Approximate uniform seeding on a feature surface with increasing seed point density from (a) to (d). White circles illustrate a random subset of markings rendered in T_m. Smaller footprints result in a denser distribution.

the viewer. To solve this issue, we employ an iterative image space seeding strategy which computes an approximate uniform object-space distribution of the remaining seed points on the GPU. The basic idea is simple. For every newly selected seed point a footprint is rendered into the corresponding layer of the *marking texture* T_m, a texture array of the same dimension as T_s, that keeps track of the surface already occupied by selected seed points. Due to the constant object space size of the footprints, it is ensured that distant seed points occupy fewer pixels than seed points located near to the viewer. This is shown in Figure 11.3. The markings stored in T_m can then be used as a predicate P to restrict the seed point search to uncovered parts of the surface.

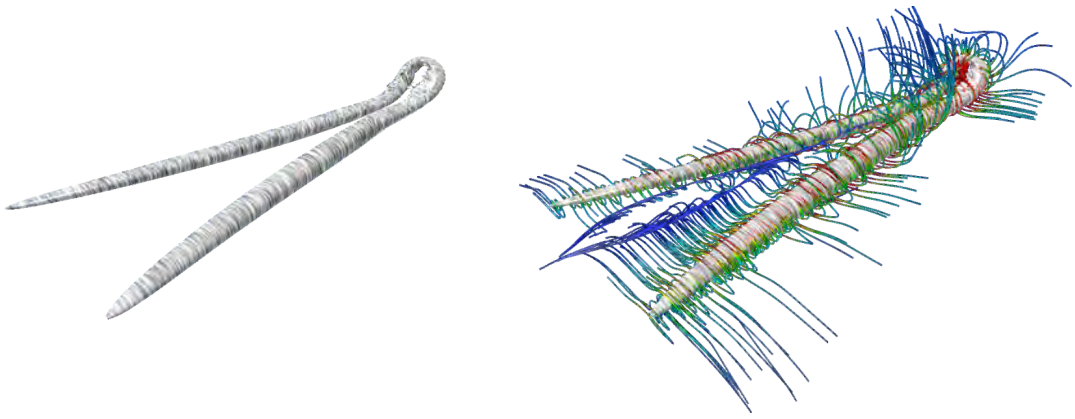
The iterative seeding strategy works as follows (cf. Listing 11.1). In each iteration, N seed points are selected at random from the yet uncovered candidate texels. Then, for each selected point a circular marking centered at its object space position p_i and oriented co-planar to the surface is rendered into T_m. This is achieved by drawing a textured quad with its normals co-aligned to the surface normal at p_i. With perspective projections, markings rendered in the foreground of the scene occupy more pixels in image space than markings that are further away from the viewer. Our iterative seeding approach accounts for this. The probability of selecting more distant candidate texels automatically increases after a few iterations.

The process is repeated until the number of remaining candidate texels falls below a specified threshold. The density of the resulting seed point distribution can be controlled with the marking radius R, as illustrated in Figure 11.3.

The markings guarantee that seed points chosen in one pass of the algorithm lie at least the distance R apart from the seed points added in previous iterations of the algorithm. Thus, an approximate uniform distribution on the visible parts of the surface would be achieved, if one seed point was added per iteration. For performance reasons multiple seed points have to be added simultaneously. As N seed points are selected randomly, it cannot be guaranteed that the minimum distance R is always met. This is why the achieved quality decreases if N is chosen too large. We use a simple

```
seedList ← ∅
while |cand| > thresh do
  cand ← extractCandidatePoints(P, T_s, T_m)
  seeds ← selectRandomPoints(N, cand)
  updateMarkingTexture(seeds, R, T_m)
  seedList ← seedList ∪ seeds
done
```

Listing 11.1 — Pseudo code of the iterative image-based GPU seeding algorithm with seed predicate filter P, new seed points per iteration N, and seed density radius R.



(a) (b)

Figure 11.4 — (a) Dense flow visualization in the local system of a hairpin vortex on a λ_2 isosurface (37 fps). (b) Surface-guided streamlines additionally provide context information about the flow (19 fps, 1200 × 1000 viewport).

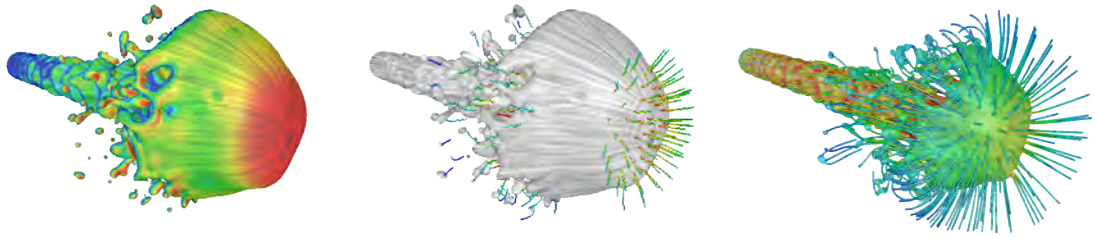
heuristic to determine the number N. Before starting the seed point distribution, a few markings chosen randomly are rendered and the number of created fragments is queried. Then, the number of seed points necessary to cover a small fraction, e.g., 5% of the surface texels, is calculated. By using this number for all iterations a relatively uniform distribution is obtained on the surface. If the viewpoint changes, the previously computed seed points are reused to retain a coherent visualizations, i.e., their markings are rendered first, and afterward, new seed points are added only in those regions of T_m that are still uncovered.

11.4 Results and Evaluation

This section evaluates the suitability of the surface-guided seeding approach for interactive exploration of real-world flow fields. All results were rendered on a Nvidia GeForce 8800 GTX. The first example is part of a DNS simulation on a uniform grid of

resolution 135×225×129 that explores the transition from laminar to turbulent flow. Figure 11.4 (a) shows a LIC visualization of the flow field on a λ_2-isosurface enclosing a vortex core. The visualization of the projected flow field alone does neither capture the flow structure in the proximity of the surface nor the interaction between the two legs of the vortex core. Our method, in contrast, allows to enhance the visualization by seeding streamlines on the surface that are traced forward and backward (Figure 11.4 (b)). These lines visualize how the flow approaches the vortex core, allowing the viewer to get a better understanding of how the flow is affected by the vortex. Generating such visualizations requires no other user interaction than choosing an appropriate seed density and line length. With standard seeding techniques, like seed rakes, it would be the user's task to find an appropriate seed location. The next example uses local properties of the surface and of the flow field to steer the distribution of seed points. Figure 11.5 shows a time step of a two-phase DNS simulation of a liquid jet injected into an air-filled volume with resolution 384×160×160. The investigated breakup phenomena play an important role in many technical applications, like combustion engine design. Here, the surface represents the boundary layer between the two fluid phases. To study the flow in the separation layer LIC is visualized on the surface. The error introduced by projecting the three-dimensional vector field on the surface is color-coded on the surface (Figure 11.5 (a)). The flow runs nearly tangential to the surface in the blue regions. A red coloring, in contrast, indicates strong cross-surface flow, i.e., large projection errors. In such regions, a non-colored LIC visualization does not provide a good impression of the 3D flow. To improve the visualization, streamlines are started in regions where the projection error exceeds a given threshold t_c (Figure 11.5 (b)). To this end, a simple filter predicate $P := (\mathbf{N} \cdot \bar{\mathbf{u}} > t_c)$ is constructed, where $\mathbf{N}$ is the surface normal and $\bar{\mathbf{u}}$ is the normalized flow vector at the respective surface point. Finally, the length of the streamlines can be controlled by the field quantities as well. In the case of Figure 11.5 (c), short lines are generated in regions where LIC works well, whereas longer lines are generated at the front of the jet, which exhibits higher projection error.

The advantages of our GPU technique are its assistance in interactive flow exploration and its unified handling of arbitrary renderable surface representations. For the two structured grid examples, the ray casting and LIC computation took between 30 and 45 ms combined. The image-based seeding algorithm has the advantage of not depending on the grid type. It only adds another 20 to 30 ms to the total computation time. This enables interactive rates even on older Nvidia 8800 GTX hardware. The dense uniform seeding strategy helps in getting a fast impression of the field. Subsequent exploration is supported by user-definable predicates to restrict the seeding to regions of interest. While local field quantities were used in the examples, like vorticity magnitude or the projection error, seeding criteria can also be derived from global field information. One example is a global measure of flow separation, the so-called FTLE, which will be introduced in Chapter 13. There, it will be demonstrated that seeding path lines in regions of strong separation (with high FTLE values) can help in getting



(a) (b) (c)

Figure 11.5 — Rendering of a feature surface and the nearby flow. (a) LIC with color-coded projection error. (b) Seeding is restricted to regions with high error (red), i.e., strong cross-surface flow, 17 fps. (c) Seed points are distributed uniformly and line length is controlled by cross-surface flow magnitude. The velocity magnitude of the non-projected flow is mapped to color, 15 fps (900×600).

a better understanding of the global flow structure (Section 13.3.1). It is noteworthy that the technique presented in this chapter could also be used to guide such a seeding by constructing appropriate seeding predicates based on the FTLE. In Part IV the FTLE leads to topology-based methods for flow visualization. The construction of the resulting vector field topology is based on distinguished field locations as well. These locations are used for seeding integral manifolds that separate the flow into regions of different behavior.

Chapter 12

GPU-Based Computation of Integral Curves

Integral manifolds (Section 5.4) are an important building block for many flow visualization techniques. An example is the finite-time Lyapunov exponent (FTLE) field (Chapter 13), which is derived from a dense field of integral curves. Before the advent of GPGPU programming languages like CUDA, the programmable GPU graphics pipeline was harnessed to accelerate integral line computation. Early techniques used fragment shaders to advect particles along 3D flow fields on structured grids, enabling an interactive visualization for huge numbers of particles [101, 144]. While these methods can also generate streamlines, storage overhead can become a problem when considering streamlines of variable length. To account for this, we present a streamline integration technique for structured grids that uses OpenGL geometry shaders (GS) [195]. Our approach is able to handle early termination of streamlines efficiently (Section 12.1). Nowadays, the computation of integral curves is typically carried out with CUDA, OpenCL, or the recently introduced compute shaders. Our GS approach, however, is still eligible because the geometry shader directly generates line geometry that can be rendered efficiently. As an example, the GS approach has been employed successfully for the interactive image space seeding technique described in the previous chapter. Moreover, we provide a detailed performance analysis in Section 12.2. The analysis reveals that certain GS parameters can have a large impact on GS throughput in general. Section 12.3 finally describes the differences to the CUDA-based integrators used for FTLE computation, where a direct display of the line geometry is usually not necessary.[1]

[1] Parts of this chapter have previously been published in Üffinger et al. [195]. While the basic algorithm sketched in Section 12.1 had already been presented in the context of my diploma thesis [193] an improved DirectX 10 implementation and an extended evaluation were added in [195].

12.1 Streamline Integration with the Geometry Shader

In contrast to the one-to-one stream processing semantics of the vertex and fragment shader stages, the geometry shader (GS) is a programmable part of the GPU graphics pipeline that allows to produce a varying number of output elements depending on calculations done in the shader (Section 2.2). This property is used for efficiently terminating the integration of individual lines to avoid subsequent computation and memory overhead for those lines.

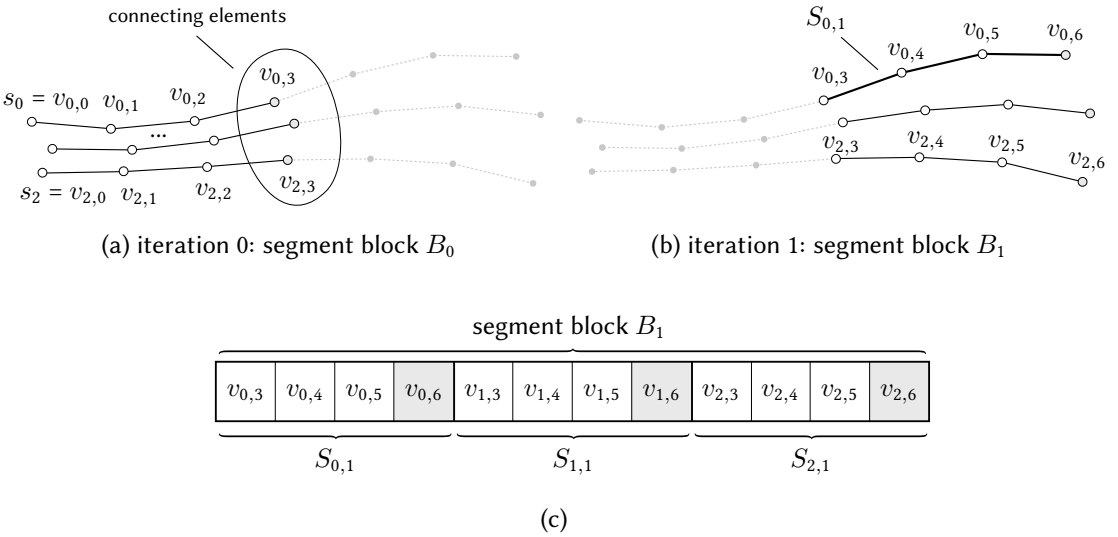
The basic idea is to send a set of seed points to the GPU and to perform the advection directly in the geometry shader. Subsequently, the GS hands down the generated vertex positions in the pipeline as line strips. Unfortunately, the number of vertices that can be written by one invocation of the GS is limited. To still allow generating streamlines of arbitrary length, we developed a multi-pass algorithm (Figure 12.1). It uses the transform-feedback (TF) mechanism of OpenGL to write the generated line data to vertex buffers resident in texture memory. Thereby, generated line data can be input directly into subsequent passes of the shader. Rasterization is completely disabled in the process. For a more detailed description some definitions are required (compare with Figure 12.1 (c)):

- *Vertex elements* $v_{l,m}$ are the basic components that are computed and output by the GS, with the line id l and the vertex id m representing the relative position to the *seed element* $s_0 = v_{l,0}$.
- A *streamline segment* $S_{l,k}$ contains all vertex elements of streamline l generated in the k-th iteration (or k-th pass) of the algorithm.
- A *segment block* B_k contains all streamline segments $S_{l,k}$ generated in parallel in the k-th iteration.

Two parameters control the generation process:

- The *block size* N determines the number of simultaneously generated lines.
- The *segment length* L controls the number of integration steps per streamline segment. The upper limit for L is given by the storage requirements of a single vertex and the output limitations of the GS.

For the generation of the first segment block B_0, N seed elements s_i—stored in a vertex buffer—are input to the GS as point primitives. For each s_i an instance of the GS is executed to generate exactly one streamline segment $S_{i,0}$. Starting with the seed element, each instance emits $L + 1$ new vertex elements. As the order of the primitives output by the geometry shader is determined by the order of the input primitives, the streamline segments $S_{i,0}$ are written consecutively into the TF-buffer, resulting in the regular block structure shown in Figure 12.1 (c). The three line segments are



(a) iteration 0: segment block B_0

(b) iteration 1: segment block B_1

(c)

Figure 12.1 — The multi-pass geometry shader algorithm advances multiple streamlines in a segment-wise manner. Transform feedback allows for an efficient communication between subsequent iterations (a) and (b) of the pipeline. (c) Regular structure of one segment block in GPU memory, here B_0 ($N = L = 3$).

generated in parallel. They are all of size $L + 1$. The *connecting vertex elements* of the N line segments $S_{l,k-1}$ are directly used as seed elements for the next pass to obtain continuous streamlines. Since vertex buffers do not support simultaneous reading and writing, two distinct buffers have to be used in a ping-pong fashion. After their generation, the individual streamline segments can be interpreted and rendered efficiently as independent line strips.

There are many cases were the integration of streamlines should be stopped for efficiency reasons. Our GS approach has the advantage that a varying number of vertices can be output during runtime. Still, the termination cases have to be handled correctly because the algorithm depends on the fixed block structure of the output data: If an instance of the GS detects a positive termination condition the remainder of the line segment is filled with dummy elements marking the termination. If a GS instance, in contrast, receives a termination element as input, it does not output any vertex data for the corresponding line. In such a case, the total number of vertex elements written to the TF-buffer is reduced by L. Finally, with the number of written elements obtained from an OpenGL query, the number N of active lines segments can be computed and adapted accordingly in the next pass. This approach is efficient with respect to memory usage as well as computational overhead. At most L dummy vertex elements are generated for each terminated line. In subsequent passes, no more memory or computation is required for terminated lines, as the GS output is automatically compactified by the GPU.

12.2 Evaluation of Integration Performance

While the OpenGL geometry shader implementation was used for the interactive visualization system described in Chapter 11 for performance measurements we reimplemented the algorithm in DirectX 10. The performance has been evaluated on a workstation equipped with an Intel Q6600 2.4 GHz quad-core processor, and an Nvidia GTX 280 graphics card. The integration performance is compared to the fragment shader technique proposed by Krüger et al. [101] and an optimized CPU implementation, which uses OpenMP and SSE3 to harness the four cores of the test system. A standard RK4 integration scheme was used and line termination was disabled.

The characteristics of the vector field as well as the location of the chosen seed points can have a strong impact on the measured integration performance. To account for this, we constructed two artificial vector fields, avoiding possibly misleading caching benefits introduced by batches of streamlines staying close together for longer periods. *Setup A* aims at reducing intra-thread cache hits of individual lines by using a helical field of size $128 \times 192 \times 192$ that ensures that lines travel through the whole domain. *Setup B* studies the case of optimal cache usage by using a flow field size of $8 \times 8 \times 8$, which is expected to fit completely into the texture cache. The performance results of *Setup A* and *Setup B* can be interpreted as an approximate lower and upper limit for real-world performance. Illustrations of these fields are given in [195]. For *Setup C*, the CFD flow shown in Figure 11.1 of resolution $384 \times 192 \times 96$ is used. For all setups, a random uniform distribution of seed points is applied.

Up to a certain degree, performance is expected to increase with the number of streamlines N that are computed simultaneously. In the GS case, performance is additionally affected by the segment length L. Figure 12.2 shows the results of *Setup A*. The left diagram compares the three test candidates. The newly proposed GS technique outperforms the fragment shader approach. This finding is also supported by the results presented in Table 12.1. The optimized CPU implementation clearly falls back behind the GPU algorithms. The diagram in Figure 12.2 (b) gives insight into how the geometry shader performance depends on the number of vertices L generated per invocation. The abrupt increases and drops suggest a preference of the Nvidia hardware for certain batch sizes, with optimal performance achieved around a segment length of 40 to 50

Setup	A	B	C
CPU (SSE, OpenMP)	6	11	8
GPU (fragment shader)	41	255	46
GPU (geometry shader)	79	256	82

Table 12.1 — Streamline integration performance in 10^6 RK4 integration steps per second. An optimized CPU implementation (Intel Q6600) is compared with a GPU fragment shader algorithm [101] and our geometry shader approach (Nvidia GTX 280).

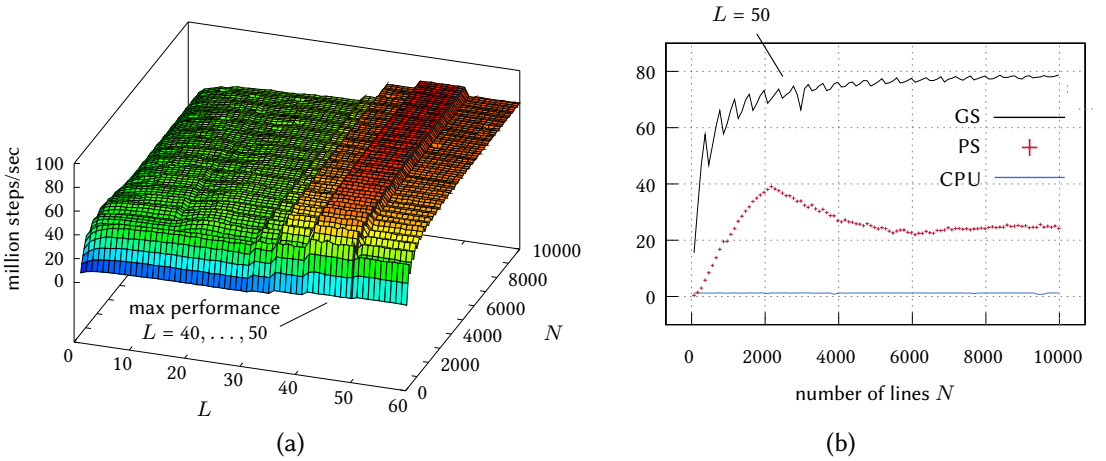


Figure 12.2 — Integration performance in million steps per second (Setup A). (a) GS performance depends on the number of lines N and the segment length L. (b) Optimal performance at $L = 50$ compared to PS and CPU integrator (Nvidia GTX 280).

vertices. A similar behavior can be observed with the CUDA programming language, where the use of shared memory and the thread block size have to be optimized by the user. For the measurements shown in Table 12.1 optimal parameters for the respective hardware have been used.

12.3 Path Line Integration with CUDA and Clusters

While the multi-pass GS approach is optimized with respect to the occupied size of lines of varying length, our CUDA integrators are optimized for FTLE computation with streamlines and path lines of fixed size. Thus, the whole integral lines can be computed in a single execution of the kernel. For path line computation, the consecutive time steps of unsteady fields are streamed to the GPU if the whole field does not fit into graphics memory. Despite the streaming overhead, path line computation is still significantly faster on the GPU than on the CPU. Besides the advantage in overall performance, GPUs additionally benefit strongly from the texture caching mechanisms implemented in hardware. While our algorithms only use single workstations and GPUs, the workload could be easily distributed on multiple machines, as long as at least two time steps of the vector field fit into the GPU memory. Very large field data can be distributed on multiple machines, similar to distributed DVR techniques. However, the problem of integral curve computation cannot be decomposed that easily. With particle tracing being a global operation, it is difficult to predict at which point in time the computation of a line has to be handed over to a neighboring compute node. Hence, distributed computation of integral manifolds requires sophisticated load balancing algorithms to achieve a high CPU and GPU utilization [138].

Chapter 13

Finite-Time Lyapunov Exponent Fields

The *finite-time Lyapunov exponent* (FTLE) is a measure of separation in vector fields. Whereas the FTLE has been applied for a long time to predictability analysis, there is a recent trend in the visualization community to use the FTLE for revealing the overall structure, or topology, of time-dependent vector fields. For this purpose, the generalized local maxima, or ridges, of the FTLE field are extracted. FTLE ridges indicate strong separation, and therefore separate the flow into different regions with coherent behavior. This property makes FTLE ridges suitable as a building block for time-dependent topology concepts, as discussed in Part IV.

This chapter focuses on computational aspects and interactive analysis of FTLE fields. Section 13.1 first introduces the classical "infinite-time" Lyapunov exponent (LE) from dynamical systems theory. This leads to the FTLE definition for analytical 2D and 3D vector fields, followed by a framework for FTLE computation (Section 13.2). Within this framework, the advantages and disadvantages of existing FTLE algorithms are discussed. Section 13.3 covers novel interactive visualization techniques for 2D FTLE fields that help in gaining a better understanding of the complex FTLE structure. Often, the complexity of FTLE fields also leads to sampling issues. This is additionally fostered by other shortcomings of the traditional FTLE algorithms. Section 13.4 discusses these quality issues and introduces a new algorithm for FTLE computation, accounting for nonlinear flow map behavior. An overview of all FTLE-related methods developed in the context of this thesis is given in Table 13.1.[1]

13.1 Definition

The Lyapunov exponent (LE) is used in predictability analysis to study the rate of separation of infinitesimally close trajectories in a temporally unconstrained dynamical

[1] Parts of this chapter have previously been published in Üffinger et al. [197].

FTLE visualization	FTLE quality	Vector field topology
Trajectory augmented visualization (Sec. 13.3.1)	Nonlinear FTLE computation (Section 13.4.1)	LCS property of FTLE ridges (Chapter 15)
Interactive visual analysis (Section 13.3.2)	Refinement and supersampling (Section 13.4.2)	Streak-based vector field topology (Chapter 16)

Table 13.1 — Overview of our FTLE-related techniques.

system. Given a system with linear behavior, small perturbations $\delta(\mathbf{x}, t_0)$ that have originated at a position $\mathbf{x}$ and time t_0 in phase space are subject to exponential growth

$$\|\delta(\mathbf{x}, t_0 + T)\| = e^{\lambda T}\|\delta(\mathbf{x}, t_0)\|, \tag{13.1}$$

after evolution for time T. Here, λ is called the *Lyapunov exponent* (LE), denoting the rate of separation. With the actual separation depending on the orientation of the initial perturbation, this definition allows for a whole spectrum of Lyapunov exponents—n exponents for an n-dimensional system like a vector field. The largest of these exponents, $\lambda_1(\mathbf{x})$, is of major interest since it represents the upper bound of perturbation growth, allowing for a classification of the system behavior. Chaotic behavior, for example, is indicated by positive $\lambda_1(\mathbf{x})$, i.e., adjacent trajectories diverge exponentially. This is why an increase in $\lambda_1(\mathbf{x})$ goes along with a decrease in the time span up to which the system can be predicted deterministically. The largest LE is defined as

$$\lambda_{1,t_0}(\mathbf{x}) = \lim_{T\to\infty} \lim_{\|\delta(\mathbf{x},t_0)\|\to 0} \frac{1}{|T|} \ln \frac{\|\delta(\mathbf{x}, t_0 + T)\|}{\|\delta(\mathbf{x}, t_0)\|}, \tag{13.2}$$

with the initial perturbation at time t_0 oriented such that λ becomes maximal. Since the LE was originally introduced for predictability analysis, it has to reflect properties along trajectories [16]. This, and the validity of the involved linear approximation for arbitrary dynamical systems, is ensured by the asymptotic limit $\|\delta(\mathbf{x}, t_0)\| \to 0$.

Since separation of trajectories is investigated as time approaches infinity, the classical LE is a global measure. For this reason, the LE is not suitable for analyzing aperiodic time-dependent dynamical systems. Many non-closed systems that are bounded in space cannot be handled either, because of the chance of trajectories leaving the domain after a finite advection time T.

To account for this limitation, the finite-time Lyapunov exponent (FTLE)

$$\sigma_{t_0}^{T}(\mathbf{x}) = \lim_{\|\delta(\mathbf{x},t_0)\|\to 0} \frac{1}{|T|} \ln \frac{\|\delta(\mathbf{x}, t_0 + T)\|}{\|\delta(\mathbf{x}, t_0)\|} \tag{13.3}$$

was introduced and investigated [72, 135, 219]. In contrast to the asymptotic analysis $T \to \infty$, the FTLE examines a finite time interval $[t_0, t_0 + T]$, i.e., it represents the separation, or predictability, of trajectories after time T. The advection time T serves

as a scale parameter, i.e., it enables the user to reduce the complexity of the resulting FTLE field. While ridges in *forward FTLE* fields ($T > 0$) indicate a strong separation of the flow, in *backward FTLE* ($T < 0$) (also called *reverse FTLE*) the contrary is the case, i.e., ridges represent attracting manifolds. An important reason for the success of the FTLE in visualization is that vector field data are often temporally bounded and hence only amenable to analysis by temporally finite concepts [60, 63, 78, 152, 156, 157].

13.2 Methods and Framework for FTLE Computation

Analytical FTLE evaluation according to (13.3) is usually not possible, in particular if the dynamical system is described by discretized vector fields, like from CFD. For such data, different numerical methods have been proposed that approximate (13.3). A straightforward approach for computing the FTLE at a position $\mathbf{x}$ is to start two or more nearby trajectories (path lines in the case of unsteady fields), advect them for time T, and measure their separation. To obtain accurate FTLE values, the initial perturbation of these trajectories needs to be oriented appropriately to capture the maximum separation. Additionally, the perturbation has to be chosen sufficiently small to allow for the linearization in the asymptotic limit in (13.3).

Even for small initial perturbations, trajectories might still diverge strongly, and hence, the computed FTLE value might not reflect the separation behavior of the field observed along a *reference trajectory* started at $\mathbf{x}$ (Figure 13.1 (a)). Renormalization techniques [135] alleviate this issue by using differently oriented pairs of trajectories, selecting the one with the strongest separation, and restarting it during integration. The reference trajectory $\mathbf{x}_{t_0}^T$ is therefore, as shown in Figure 13.1 (a), seeded at $\mathbf{x}$ and the other trajectories on a circle with radius h. The latter trajectories are kept close to the reference trajectory by renormalization, i.e., by projecting it toward the reference trajectory from time to time if the distance becomes too large during integration. The overall separation is accumulated over the renormalization steps, yielding the R-FTLE value σ_{r,t_0}^T at point $\mathbf{x}$. If the initial perturbation h is chosen sufficiently small renormalization may not be necessary.

Localized FTLE σ_{l,t_0}^T (L-FTLE) [89] belongs to a second class of methods that only use a single trajectory to approximate $\sigma_{t_0}^T(\mathbf{x})$. These methods accumulate separation information encoded in the Jacobian $\nabla\mathbf{u}$, which they evaluate along the reference trajectory started at $\mathbf{x}$, as shown in Figure 13.1 (b). This represents a truly local approach in accordance with the asymptotic limit in Equation 13.3.

Nowadays, the F-FTLE, an approach made popular by Haller [78], is the most common choice for approximating $\sigma_{t_0}^T$. It is based on the *flow map* $\phi_{t_0}^T$, which maps seed points $\mathbf{x}_{t_0}^0$ of trajectories started at t_0 to their respective end points $\phi_{t_0}^T(\mathbf{x}_{t_0}^0) = \mathbf{x}_{t_0}^T$ after advection for time T (Figure 13.1 (c)). The largest FTLE is then obtained as

$$\sigma_{f,t_0}^T = \frac{1}{|T|} \ln \sqrt{\lambda_{\max}\left((\nabla \phi_{t_0}^T)^\top \nabla \phi_{t_0}^T \right)}, \tag{13.4}$$

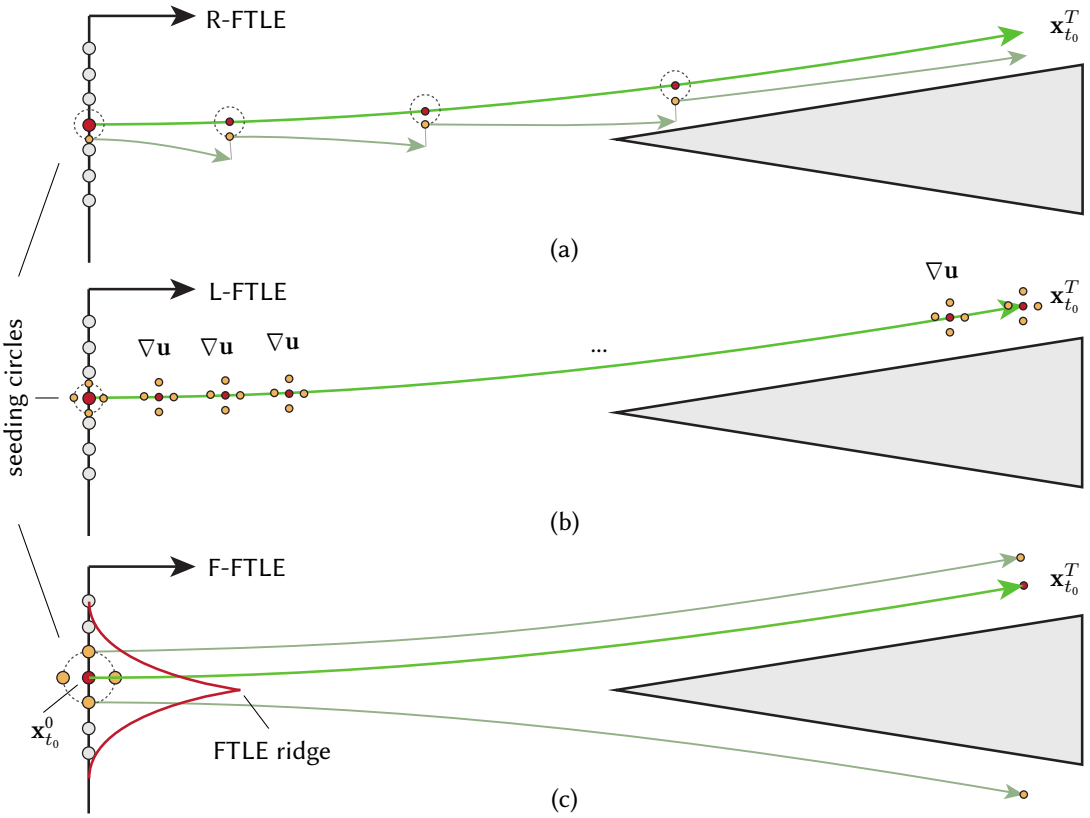


Figure 13.1 — Comparison of FTLE computation methods. (a) R-FTLE and (b) L-FTLE are prone to miss separation due to local measurement of flow stretching. (c) F-FTLE, in contrast, captures the separation induced by the obstacle with no-slip boundaries (gray), resulting in a generalized local maximum, or ridge, of the FTLE profile (red).

with $\lambda_{\max}(\cdot)$ being the major eigenvalue of the right Cauchy-Green deformation tensor $(\nabla\phi^T_{t_0})^\top\nabla\phi^T_{t_0}$, corresponding to the eigenvector pointing into the direction of maximum separation. This represents a first-order approximation of $\phi^T_{t_0}$ based on the first term $\nabla\phi^T_{t_0}$ of the Taylor series. In practice, $\phi^T_{t_0}$ is discretized on a regular grid with cell size h leading to a linearization error depending on h.

Since the F-FTLE is based on the gradient of neighboring flow map samples, the detection of separation is continuous, i.e., no separation can be missed due to "holes" in the sampling.[2] This ensures that FTLE ridges, which are of great interest for topological flow analysis, are not missed. Figure 13.1 illustrates this effect. Whereas the R-FTLE and the L-FTLE—due to their local approach—do not reflect the separation induced by the wedge-shaped no-slip obstacle, the F-FTLE is able to capture the ridge in its profile (there is a peak in the red F-FTLE curve). On the other hand, L-FTLE provides, up to

[2] Small-scale separation might not be resolved if the flow map sampling is chosen too coarse.

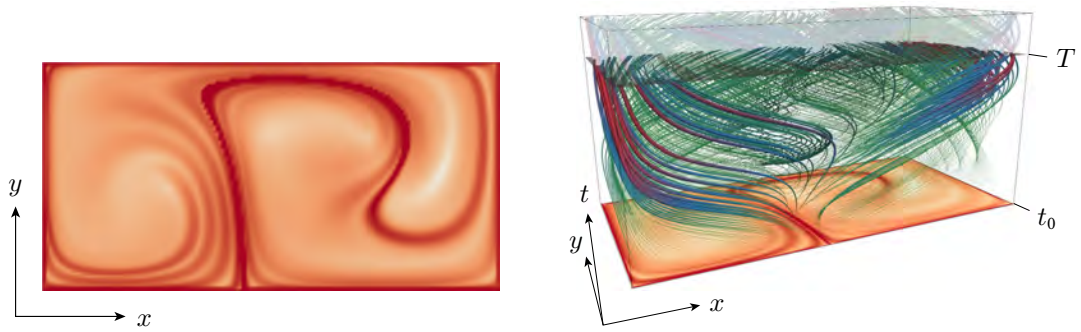
numerics, accurate FTLE values, and R-FTLE converges toward those as $h \to 0$. While this allows for increasing accuracy locally without the need to increment the sample number, these methods are more prone to aliasing issues.

All three methods share a common principle. Nearby trajectories are started on seeding circles centered around points of the reference trajectory—the dashed circles in Figure 13.1. Within this seeding circle framework, R-FTLE corresponds to a single sample on the circle and repeated seeding along the reference trajectory, L-FTLE corresponds to the limit case $h \to 0$ and continuous seeding, and F-FTLE corresponds to four axis-aligned samples[3] on the seeding circle with h identical to the flow map resolution (Figure 13.4). The seeding circle framework serves as the basis for new FTLE techniques (Chapter 13.4) that address the reduction of the F-FTLE linearization error in (13.4).

Computing FTLE fields is a costly task since at least one particle trajectory has to be computed for each sample of the field. For direct visualization of 2D FTLE fields, often FTLE raster images of resolution 256^2 or larger are computed, with the number of integration steps for each sample depending on the advection time T. Complexity increases for 3D FTLE fields accordingly. All of our FTLE techniques (Table 13.1) are based on F-FTLE, using GPU-accelerated implementations (Chapter 12). Some techniques additionally harness optimizations based on adaptive refinement schemes (Section 13.4.2).

13.3 Visualization

FTLE fields are traditionally visualized by displaying 2D images of FTLE values sampled on regular grids, as shown in Figure 13.2 (a). In 3D domains, volume rendering techniques can be used or the FTLE can be visualized on embedded manifolds. The interpretation of FTLE is usually guided by the ridges of the field—the dark red lines in the example—which partition the domain into regions of coherent behavior. Hence, ridges are also often extracted explicitly, resulting in geometric manifolds of codimension 1 that can be rendered. While the FTLE reduces field information, contained in a potentially large time-interval $[t_0, t_0 + T]$, into a single image, the visualization of FTLE alone is often not sufficient to gain a comprehensive understanding of the global field dynamics. To account for this, we present a technique that seeds path lines near ridges to convey further information about the constituting flow patterns (Section 13.3.1). Another layer of complexity is introduced by the FTLE parameters t_0 and T, which have to be chosen by the user. We propose to use techniques from visual analytics in order to facilitate the exploration of this parameter space (Section 13.3.2).



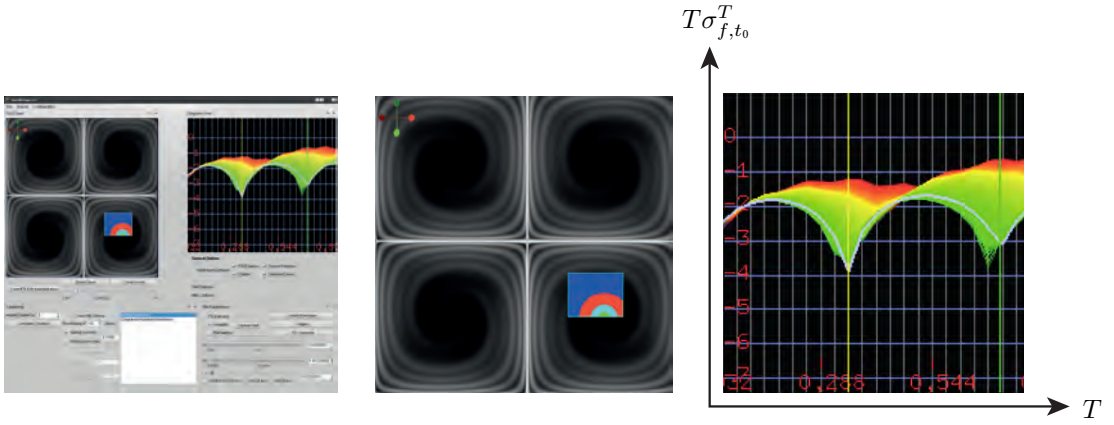
(a) (b)

Figure 13.2 — (a) FTLE field of the 2D double-gyre data set [175]. (b) Augmented space-time FTLE visualization: a subset of the constituting path lines seeded near the salient features, i.e., the FTLE ridges (dark red), are visualized. This reduces clutter in contrast to visualizations including all path lines. The advection time T can be adjusted with the horizontal cutting plane. The MSF is mapped to path line color and thickness.

13.3.1 Trajectory Augmented Visualization

Direct visualizations of the FTLE field or its ridges enables a comprehensive qualitative analysis of the instationary flow field and the dynamics of Lagrangian coherent structures (LCS), see Part (IV). However, such an approach does not provide quantitative information about the flow. The explicit vector field behavior cannot be deduced from the FTLE field shown in Figure 13.2 (a). One possibility to improve the visualization is the augmentation of ridge geometry with LIC patterns [10]. However, in general, this is only meaningful for stationary vector fields. Instead, we propose an augmentation that visualizes the LCS-inducing flow behavior by means of complete trajectories in instationary 2D flow [47]. Using a space-time visualization, time t is treated as the third dimension (Figure 13.2 (b)). Occlusion and visual clutter is minimized by steering the placement of trajectories with the FTLE. Only those lines are visualized that are most relevant for understanding the global structure of the field. The geometric line representation is additionally used to encode the maximum separation factor (MSF), which is defined as the spectral norm of $\nabla\sigma_{t_0}^{T}$. Related to the FTLE, the MSF misses the logarithm and the normalization with respect to the advection time T to prevent interpretation issues when being directly mapped to geometrical properties like the thickness of the visualized lines. By augmenting the LCS visualization with path lines, the explicit flow behavior that led to the LCS of a fixed advection time T can be identified and analyzed easily by looking at a single image. The advection time T is a critical parameter when visualizing LCS with FTLE ridges. So far, no satisfactory solution for its choice has been given, as discussed in Chapter 15. With our visualization, however, the variation of the line thickness according to the MSF can give the viewer an impression on how the FTLE develops with increasing T. Our prototype allows for

[3] This assumes that central differences are used for computing $\nabla\phi_{t_0}^{T}$.



(a) (b) (c)

Figure 13.3 — (a) Visual analytics environment for the interactive exploration of FTLE fields. Example with two linked views: (b) FTLE view with color overlay highlighting four clusters within a region of interest, (c) FTLE chart of trajectories color-coded according to the spatial location of their FTLE seed points $x^0_{t_0}$.

interactive exploration of the FTLE parameters in the sense that a specific advection time T can be chosen, resulting in an update of the visualization after a short time. Note that our surface-guided integral lines approach (Chapter 11) could be used to steer the path line seeding in the case of 3D FTLE fields.

13.3.2 Visual Analytics for FTLE Fields

Interactive analysis of FTLE fields is important to enable an efficient exploration of the FTLE parameter space, which is spanned by the starting point of the analysis t_0 and the advection time T. To this end, a visual analytics framework was developed under our supervision in the context of a student thesis [166]. The resulting application offers more sophisticated analysis tools than the technique presented in the previous section, including automatic clustering algorithms. Figure 13.3 (a) shows the graphical user interface of the application. It is composed of two coordinated views the user can interact with. Several GUI elements allow for choosing the FTLE parameters. The FTLE field in the upper left widget is updated on the fly on parameter changes. Zooming is also supported, an important feature which helps in exploring fine FTLE ridge structures. For a fixed upper advection time T, a set of F-FTLE flow maps with different t_0 are precomputed and cached to support the user in gaining a fast overview of the available time interval. The widget in the upper right provides an FTLE value chart of all selected sampling points (Figure 13.3 (c)). Each FTLE sample is represented by one line in the diagram, showing the evolution of F-FTLE values $\sigma^{T_i}_{f,t_0}$ along the respective reference trajectory, with $T_i \in [0 \dots T]$. In practice, we plot the FTLE without the normalization by T in (13.3) to avoid that the lines converge toward zero for large T. While the FTLE is expected to converge against some value (at least in bounded

domains), this visualization can help in detecting patterns, like a temporary decrease of the FTLE or a periodic behavior as in the example. Such patterns can be characteristic for certain flow configurations. Due to the large number of lines, the system provides automatic algorithms that cluster the lines according to similarity measures. If the user selects single lines or whole clusters, the corresponding regions of the flow domain are highlighted in the FTLE widget, as shown in Figure 13.3 (b). Conversely, the set of plotted lines in the diagram can be restricted by marking a region of interest in the FTLE view. Additionally, seed points can be selected by the user to augment the FTLE view with particle trajectories. Similar to the method presented in the previous section, this adds quantitative field information.

13.4 FTLE Quality

Long FTLE advection times T often result in a very fine folded ridge structure. Hence, appropriate sampling can become a computationally infeasible challenge in practice, as the resolution of the FTLE field cannot be increased arbitrarily. This results in aliasing artifacts that hinder robust ridge extraction. While L-FTLE and R-FTLE are able to calculate accurate FTLE values by locally increasing the sampling resolution, they are prone to miss features like ridges. The popular flow map-based F-FTLE on the other hand captures those features but introduces a linearization error due to the finite resolution. To account for these issues, we propose a new nonlinear approach to FTLE computation based on the F-FTLE in the next section [197]. This is followed by a discussion of adaptive sampling techniques, as a means to further improve FTLE quality (Section 13.4.2). In this context, we introduce a novel refinement criterion based on the linearization error. Moreover, we demonstrate that supersampling can help in alleviating aforementioned aliasing issues.

13.4.1 Beyond First-Order Approximation

In the following, we introduce a novel approach based on F-FTLE that accounts for nonlinearity in the flow map. Secondly, a measure for quantifying nonlinearity is derived, leading to the definition of circumference FTLE, which measures the nonlinear deformation of seeding circles.

Multi-directional FTLE (MD-FTLE) combines the suitability for topological analysis of the F-FTLE (its absence of "sampling holes") with better predictability (higher accuracy of FTLE values). Within the FTLE framework introduced in Section 13.1, MD-FTLE is formulated by a seeding circle of radius h identical to F-FTLE (Figure 13.4 (a)), but with additional samples on the circle (Figure 13.4 (b)). The additional samples are used to obtain a better approximation of the advected seeding circle than the F-FTLE does, which employs a linear "ellipse" model. The MD-FTLE

$$\sigma^T_{md,t_0} = \frac{1}{|T|} \ln\Big(\max_{i=1,\dots,N} \big(|\mathbf{c}^T_{i,t_0} - \mathbf{x}^T_{t_0}| / h \big) \Big) \tag{13.5}$$

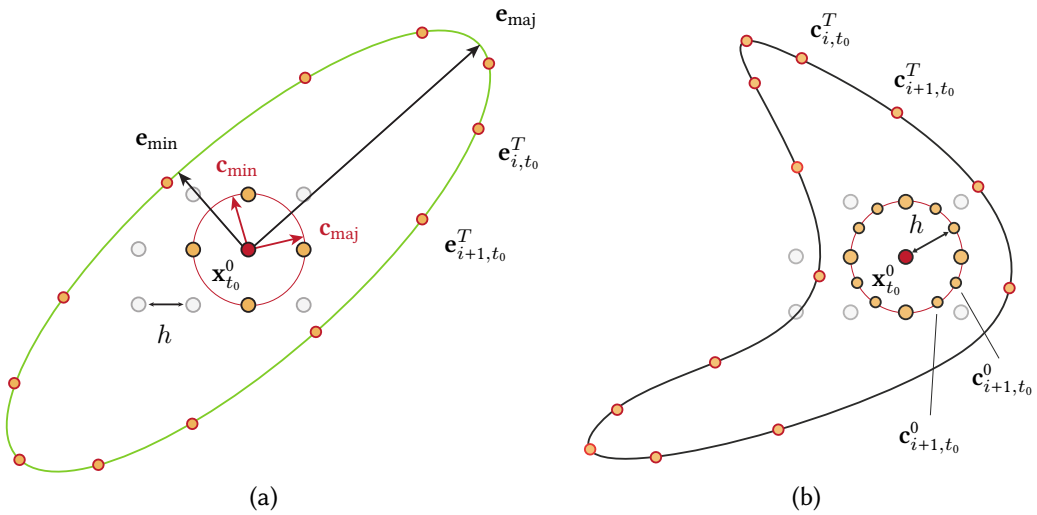


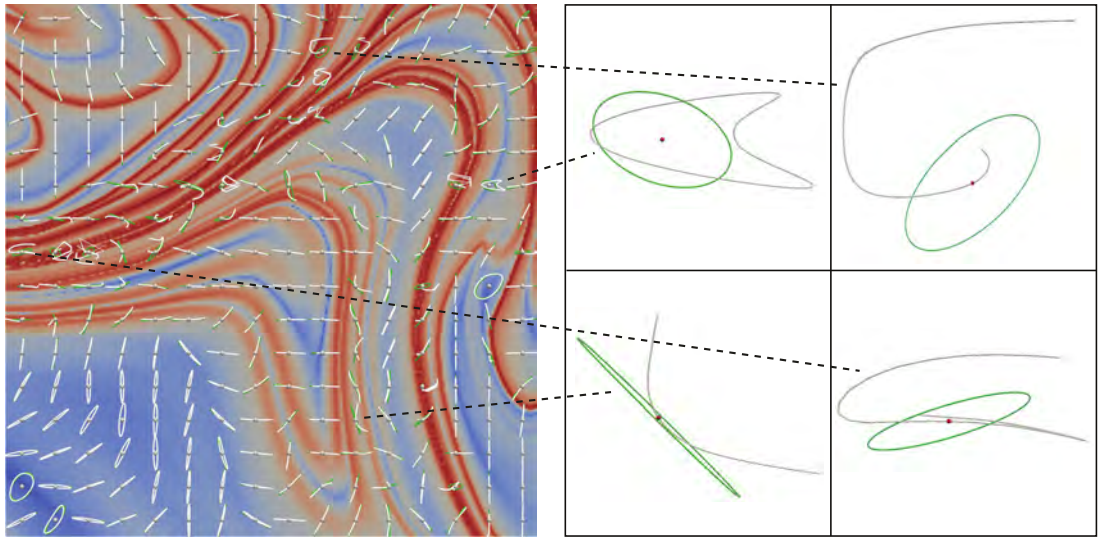
Figure 13.4 — (a) F-FTLE approximates the flow map linearly, e.g., by $\nabla\phi^T_{t_0}$ derived from four seeding trajectories. (b) Using more seeds on the circle captures the nonlinear behavior of $\phi^T_{t_0}$. Both, the linear separation ellipse in (a) (green) and the true deformed seed circle after advection for time T in (b) (black) are drawn at their seed location $\mathbf{x}^0_{t_0}$.

is directly obtained from the geometry of the advected circle. Conforming to (13.2), it computes the maximum distance between the N advected circle points $\mathbf{c}^T_{i,t_0}$ and the end point of the reference trajectory $\mathbf{x}^T_{t_0}$. In Figure 13.4, the advected points $\mathbf{c}^T_{i,t_0}$ have been drawn at their seed location, translated by $\mathbf{x}^0_{t_0} - \mathbf{x}^T_{t_0}$ for illustration purposes. Compared to the popular F-FTLE, the MD-FTLE exhibits a more isotropic evaluation, reducing the sensitivity with respect to the orientation of the initial seed point perturbations.[4] This can lead to a more appropriate sampling with less angular aliasing. This effect is visible at the FTLE valleys in Figure 13.6 (a) and (b). Using more than $N = 32$ samples did not further improve the results in our experiments. Note that MD-FTLE with $N = 4$ is identical to F-FTLE and that MD-FTLE, as F-FTLE, converges to exact values as $h \to 0$.

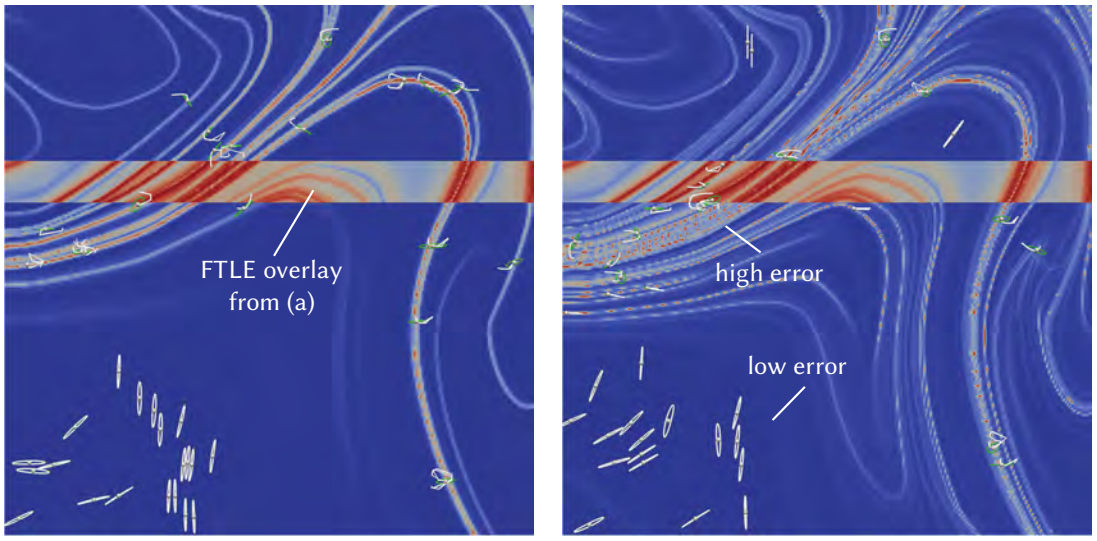
Figure 13.5 (a) shows several advected circles (white) placed at their start locations $\mathbf{x}^0_{t_0}$ and normalized in size. It can be seen that in some regions of the buoyant flow, the circle is deformed linearly to the shape of an ellipse, but in others in a nonlinear manner. While the shape of the advected circle in Figure 13.4 (a) reveals the true, potentially nonlinear separation behavior of the flow (with respect to h), the traditional F-FTLE, in contrast, is only exact as long as the investigated scale exhibits linear behavior, i.e., as long as the circles are deformed to ellipses. This is the case in the lower left quadrant of the domain. The true separation

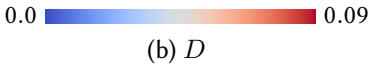
$$\delta^T_{t_0} = (\nabla\phi^T_{t_0})\delta^0_{t_0} + G(\mathbf{x}, \delta^0_{t_0}) \tag{13.6}$$

[4] Using more than four flow map samples for computing $\nabla\phi^T_{t_0}$ would not capture the local nonlinear separation behavior, as discussed later.

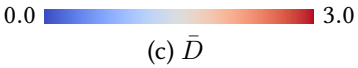


(a) σ^T_{f,t_0}



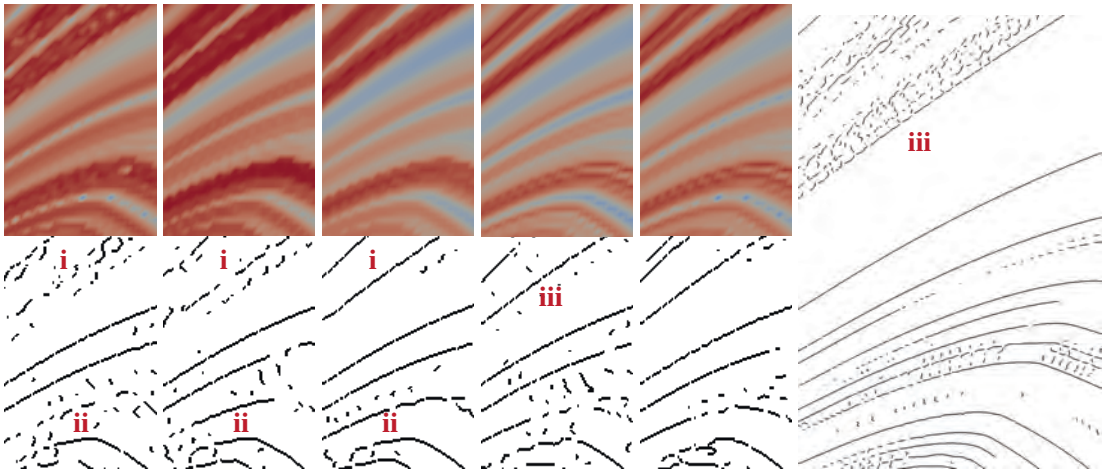


(b) D



(c) $\bar{D}$

Figure 13.5 — Nonlinear flow map behavior in a buoyant flow: (a) F-FTLE σ^T_{f,t_0} with advected seed circles (white) and corresponding linear approximation ellipses (green), both normalized in size and placed at their seeding location $\mathbf{x}^0_{t_0}$. The close-ups show four advected seed circles. (b) Corresponding linearization error field D with advected circles for a random subset of samples with highest and lowest error. (c) Same with the normalized field $\bar{D}$ (flow map resolution of the ROI= 256^2, $t_0 = 8.05$, $T = 0.75$).



(a) σ^T_{f,t_0} (b) σ^T_{md,t_0} (c) σ^T_{c,t_0} (d) ss 8× (e) ass 8× (f) hi res 8×

Figure 13.6 — FTLE and extracted ridges for all approaches: (a) F-FTLE. (b) MD-FTLE and (c) C-FTLE enable a more robust ridge extraction, see i and ii. Ridge extraction in 8× (adaptively) supersampled FTLE (d) and (e) can perform better than ridge extraction in the 8× hi res field (f), e.g., in regions with strong aliasing like iii. (Crop of the domain in Figure 13.5, 42 × 27, $N = 32$, $t_0 = 8.05$, $T = 1.0$).

involves a linear term $\nabla\phi^T_{t_0}$, as well as higher-order terms which are expressed by G [41]. In Figure 13.5 (a), the advected seeding circles near the ridges of the buoyant flow (white) deviate strongly from the ellipse model (green), indicating a substantial nonlinear error term G. Whereas the linear approximation is valid for $h \to 0$, merely increasing the flow map resolution to allow for linearization is typically not feasible. This is due to constraints on computation time and space, especially with larger advection times T and the resulting strong nonlinear deformations. Note that incorporating G in the computation of the initial direction $\delta^0_{t_0}$, leading to maximum separation $\delta^T_{t_0,\max}$, is difficult, also in terms of numerics.

Instead, we quantify the strength of the nonlinear deformation geometrically. We measure the deviation of the deformed advected circle from the ellipse representing the linear approximation $(\nabla\phi^T_{t_0})\delta^0_{t_0}$. Therefore, a geometric representation of the ellipse is required. The major axis $\mathbf{e}_{\text{maj}}$ of the ellipse, see Figure 13.4 (a), is calculated as $h\sqrt{\lambda_{\max}(C)}\epsilon^C_{\text{maj}}$, with ϵ^C_{maj} being the major eigenvector of the right Cauchy-Green tensor $C = (\nabla\phi^T_{t_0})^\top \nabla\phi^T_{t_0}$. Using ϵ^C_{min} instead results in the minor axis $\mathbf{e}_{\text{min}}$. To evaluate the nonlinearity, we establish a pointwise correspondence between advected circle points $\mathbf{c}^T_{i,t_0}$ and ellipse points $\mathbf{e}^T_{i,t_0}$, using $\mathbf{e}^T_{i,t_0} = \mathbf{x}^T_{t_0} + \nabla\phi^T_{t_0}(\mathbf{c}^0_{i,t_0} - \mathbf{x}^0_{t_0})$. The deviation of the advected circle from its corresponding ellipse is then obtained as

$$D = 1/N \sum_{i=1,\ldots,N} |\mathbf{c}^T_{i,t_0} - \mathbf{e}^T_{i,t_0}|. \tag{13.7}$$

This measure directly reflects the absolute error due to the linear approximation of

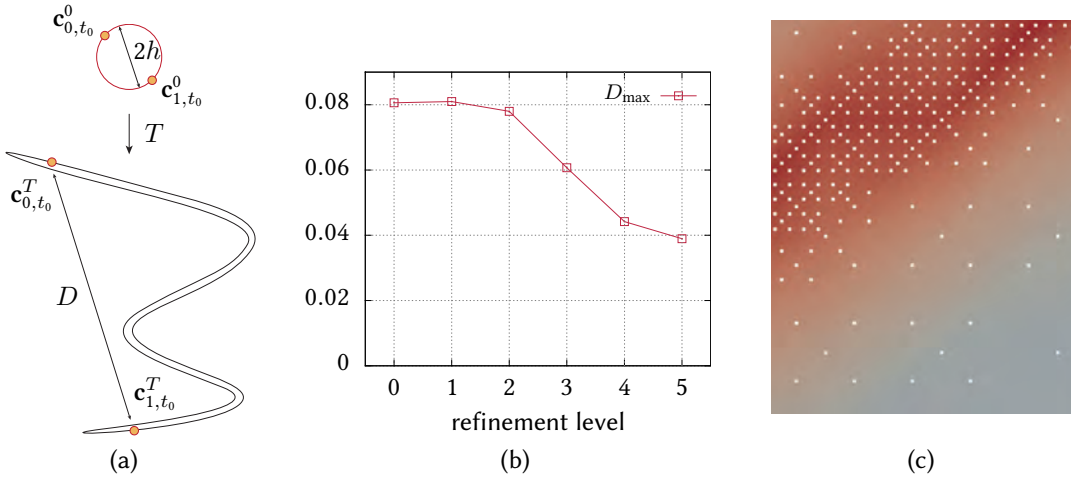


Figure 13.7 — (a) Nonlinear stretching of the seeding circle. (b) Maximum linearization error $D_{\max}$ for 5 iterations. (c) Close-up: using the linearization error measure D as refinement criterion adapts toward ridges (red), which are typically underresolved.

the flow map. In addition to D, we also define $\bar{D} = D/|\mathbf{e}_{\text{maj}}|$, the linearization error normalized by the maximum separation factor. Figure 13.5 (b) and (c) show the resulting fields for the buoyant flow. The linearization error $D(\mathbf{x})$ reveals regions with high error of F-FTLE, which in particular includes regions with sharp features, like ridges. The normalized error field $\bar{D}(\mathbf{x})$, in contrast, explicitly highlights regions with strongly nonlinear behavior. Interestingly, this also highlights the valleys, the generalized local minima of the FTLE field.

For a strong nonlinear deformation of the seed circle, as illustrated in Figure 13.7 (a), traditional FTLE underestimates the total stretching of the initial fluid element. While the FTLE measures the deformation by the Euclidean distance of a few samples, the total stretching rather corresponds to the length of the resulting curve, representing a "Lagrangian distance" instead. This motivates the definition of circumference FTLE (C-FTLE)

$$\sigma^T_{c,t_0} = \frac{1}{|T|} \ln\left(\frac{k}{4h}\right), \tag{13.8}$$

where k is the circumference of the advected circle. The results in Figure 13.6 (c) are very similar to σ^T_{md,t_0}, but exhibit slightly less aliasing. This leads to a more robust ridge extraction, as shown in the highlighted regions (i) and (ii). Note that σ^T_{c,t_0} is identical to σ^T_{f,t_0} in the limit case of $h \to 0$ (leading to elliptical transformation) and sufficiently strong separation (leading to ellipses with high eccentricity).

13.4.2 Adaptive Refinement and Supersampling

Due to the very fine ridge structure, undersampling is an intricate problem when working with FTLE fields. Sampling the field with a regular high resolution grid

is typically prohibitive because of the high computational costs, in particular in 3D. However, even at very high resolutions, the explicit extraction of ridges can still suffer from undersampling, as shown in Figure 13.6 (f). To be able to cope with the increasing costs to compute very high resolutions, adaptive algorithms have been devised. Garth et al., for example, use an estimator of flow map variation in order to refine the flow map locally [1, 63]. Sadlo and Peikert follow a similar approach based on AMR techniques, refining only those cells that contain ridges [156].

Low sampling resolutions do not only lead to wrong values, but they often underestimate the exact FTLE values obtained in the limit $h \to 0$. Hence, increasing the resolution can lead to the growth of ridge structures, and these structures might be missed if an adaptive algorithm does not account for this growth effect [156]. In Part IV, the approach of Sadlo and Peikert [156] is adopted for the efficient extraction of intersections from forward and backward time FTLE fields.

In the following, we demonstrate the suitability of the linearization error D (13.7) as a criterion for local FTLE refinement. The refinement is stopped as soon as the linear approximation is sufficiently well fulfilled, which is the case if D becomes very small. We start with a regular sampling grid on the coarsest level. A sample i on level n at location $\mathbf{x}_{n,i}$ is refined into five subsamples if $D(\mathbf{x}_{n,i})$ is larger than a defined threshold. For consistency reasons, we keep the original sample location $\mathbf{x}_{n,i}$ and distribute the other four samples uniformly within its cover, see Figure 13.7 (c). Starting with the same configuration as in Figure 13.5, but with $T = 0.5$, we performed five refinement iterations to compute the refinement hierarchy for σ^T_{f,t_0}. An overall decrease in the maximum linearization error $D_{\max}$ of all samples of a hierarchy level can be observed (Figure 13.7 (b)). Simultaneously, the refinement adapts toward the ridges (c). Interestingly, for heavily undersampled FTLE, $D_{\max}$ shows an initial increase during refinement.

If the high resolution field is still undersampled like in Figure 13.6 (f), we propose to use supersampling in order to enable a more robust ridge extraction. The advantage of supersampling was demonstrated for hyperbolicity time fields [161], which can serve as an alternative to ridges from FTLE. We downsample the refinement hierarchy to the resolution of the regular grid on the coarsest level to obtain a supersampled FTLE field, weighting the FTLE samples according to their covered area. Figure 13.6 (e) shows 8× adaptively supersampled F-FTLE obtained after 3 refinement iterations. It looks similar to brute-force supersampled FTLE in ridge regions (d), and substantially outperforms the F-FTLE of low resolution (a), justifying the additional expense. Compared to the 8× high resolution F-FTLE field (f), ridge extraction performs better in those regions of the supersampled field where the FTLE is still undersampled.

Part IV

Toward a Time-Dependent Vector Field Topology

Chapter 14

Introduction to Vector Field Topology

The classical vector field topology is successful in partitioning steady flow into regions of qualitatively different behavior (Section 14.1). It is based on concepts like critical points and distinguished streamlines, called separatrices, that converge against these points in the asymptotic limit $t \to \pm\infty$. Unsteady fields, however, are not not sufficiently amenable to analysis by these techniques. These fields often only provide a limited time domain, and typically exhibit more complex dynamics than steady fields. This makes the generalization of the classical vector field topology to unsteady fields a nontrivial problem. The development of alternative concepts has become an active research topic in recent years. A state of the art overview on unsteady vector field topology is given in Section 14.2. This leads to the definition of *Lagrangian coherent structures* (LCS) (Chapter 15), which provide a promising concept for structuring time-dependent flow into regions of qualitatively different behavior. First, LCS computation by means of ridge extraction from FTLE fields is covered. Afterward, we present a novel method that helps in determining appropriate FTLE advection times to ensure that the resulting FTLE ridges represent LCS. This is followed by the description of our new approach to time-dependent 3D vector field topology in Chapter 16. The topology is based on distinguished streak surfaces seeded near saddle-type flow regions that are obtained as intersections of attracting and repelling LCS.

14.1 Traditional "Steady" Topology

Topology-based approaches reduce the information contained in a vector field to its essential structure by segmenting the domain into regions of coherent flow behavior. For steady fields the *classical vector field topology* is an established concept. It first classifies characteristic flow patterns near stationary (critical) points and then derives a global skeleton of the flow using stream lines. The root of this approach lies in

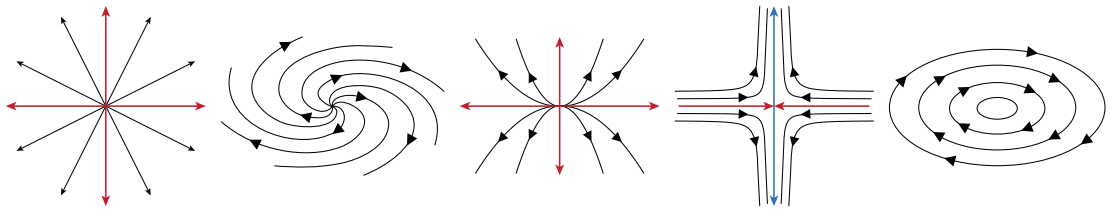


Figure 14.1 — Critical points in the classical 2D vector field topology. (a) isotropic source (2 out), (b) repelling focus (2 out), (c) node source (2 out), (d) saddle type (1 in, 1 out), (e) center. Blue vectors correspond to positive and red to negative eigenvalues of $\nabla\mathbf{u}$. Reversal of the vector field direction yields the missing cases.

dynamical systems theory of autonomous ODEs. The significance of critical points in vector fields that represent fluid flow, as solutions to the Navier-Stokes and Euler equations, was discussed by Perry and Chong [149]. Helman and Hesselink later introduced 2D flow topology to computational visualization [79], followed by extensions to 3D domains [33, 71, 80]. For an introduction from a dynamical systems perspective the reader is referred to the tutorial of Asimov [7]. In the visualization context a more recent state of the art survey is available as well [104].

Critical points are defined as points of the domain where velocity magnitude vanishes, i.e., $|\mathbf{u}| = 0$. Additionally, the Jacobian matrix $\nabla\mathbf{u}$ is required to be regular in order to restrict the investigation to isolated stationary points. At the location of a critical point the tangent of streamlines is indeterminate, i.e., critical points can also be seen as streamlines degenerated to points. In addition to critical points, Perry and Chong further highlight the importance of *separation and reattachment points and lines* (Figure 4.6) to vector field topology. The flow patterns in the vicinity of critical points are classified according to an asymptotic analysis based on the eigenvalues of the Jacobian $\nabla\mathbf{u}$. This represents a first-order Taylor approximation of the local velocity field characteristics. Possible flow patterns in 2D vector fields include nodes and foci, both of type sink and source, saddles, and centers. Some of these patterns are shown in Figure 14.1. Negative real parts $R_{1,2}$ of the eigenvalues indicate attracting field behavior (blue) and positive real parts indicate repelling field behavior (red). Rotational flow behavior is captured by non-zero imaginary parts $I_{1,2}$. By reversing the flow direction the cases missing in the illustration can be obtained. Sinks and sources as well as focus-type critical points cannot appear in divergence-free flows. Non-degenerate cases, i.e., where $R_{1,2} \neq 0$ are structurally stable. This ensures that small perturbations of the vector field, which can be introduced due to noise, do not induce a transition in the topological structure. A transition would be registered by a change in the classification of the point. Saddles exhibit both, repelling and attractive behavior. They play a special role because a lower-dimensional set of integral curves converges against or away from

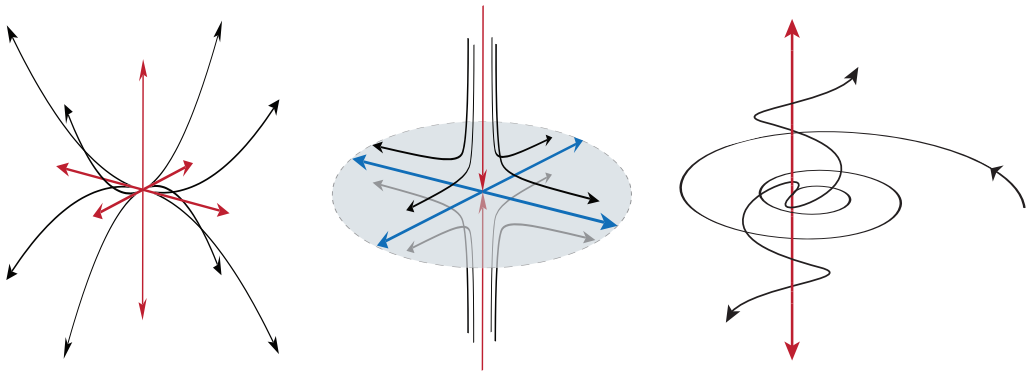


Figure 14.2 — Examples of critical points in 3D vector fields: (a) pure repelling source node, (b) isotropic 3D saddle with repelling 1D manifold (red) and attracting 2D manifold (blue). (c) Spiral saddle.

these points. These curves are called *separatrices*. They originate from the saddle point in the direction of its real eigenvectors. In practice, the computation of separatrices requires for a small seeding offset from the critical point to be able to escape from the zero velocity point. The resulting distinguished streamlines either terminate at other critical points, on no-slip boundaries, or leave the domain. They separate space into regions of qualitatively different flow behavior, forming the *topological skeleton* of the vector field. It is important to note, that the underlying asymptotic analysis with respect to $t \to \infty$ would not be possible in the case of general time dependent fields, since those are typically not available on infinite time domains. Depictions of the critical points together with the skeleton capture the essential characteristics of the field, both in a schematic and qualitative manner, such that the viewer is able to deduce the global flow structure easily. As an additional advantage, the topological approach allows for a drastic reduction of the raw vector field data.

The extension to critical points in 3D fields is straightforward. The classification of the Jacobian in 3D results in three pairs of eigenvalues and eigenvectors. Pure nodes (sources or sinks) are obtained in the case of real eigenvalues of same sign, see Figure 14.2 (a). The case of one eigenvalue differing in sign from the other two indicates a combination of 2D saddle and 2D node patterns. While such a critical point exhibits node character in one of the planes spanned by two eigenvectors, it is of saddle-type in the others. Figure 14.2 (b) shows the example of a repelling saddle, with a repelling 1D separatrix (inflow) and an attracting 2D separatrix (outflow). The case of an attracting saddle (2 in, 1 out) is obtained by reversing the vector field direction. Complex eigenvalues always appear in conjugate pairs such that the third eigenvalue is determined to have real value only. This allows for spiral saddles (Figure 14.2 (c)) and 3D foci that spiral in or out along the eigenvector corresponding to the real eigenvalue.

The *stable manifold* of a critical point p_c is defined as the set of points of $\mathbb{R}^n$ that yield

trajectories that converge to p_c in the limit $t \to \infty$. Conversely, *unstable manifolds* are obtained for convergence in reverse time $t \to -\infty$. The 3D node in Figure 14.2 (a), for example, has a unstable 3D manifold, and a stable 0D manifold—the critical point itself. The saddle in Figure 14.2 (b) in contrast has a stable 1D and an unstable 2D manifold, representing the separatrices, i.e., streamlines and stream surfaces.

Periodic orbits are also important to vector field topology. They are characterized by isolated closed streamlines, i.e., streamlines that pass through a fixed point more than once in contrast to neighboring streamlines. If they exhibit a stable or unstable manifold, they are also referred to as *limit cycles*, because neighboring streamlines approach them asymptotically in forward or reverse time. Periodic orbits exhibit local structural stability and are important because they indicate recirculating flow. However, they cannot be captured by eigenanalysis of $\nabla \mathbf{u}$ directly. The detection of reccurence requires for other approaches, e.g., fixed point analysis in *Poincaré maps* [7, 113]. Wischgoll and Scheuermann presented techniques for the extraction of periodic orbits in 2D [214] and 3D vector fields [215]. Asimov also classifies orbits of different character and describes their associated manifolds [7].

Topology of steady 2D vector fields is typically visualized by drawing the critical points and boundary points at their corresponding locations. Separatrices are typically rendered as lines [79]. Visualization of the topological skeleton is more intricate in the case of 3D fields, where the computation of separating stream surfaces gets more complicated. Moreover, with an increasing number of critical points, the 1D and 2D separatrices tend to overlap in image space, leading to clutter and occlusion. Consequently, a compromise between the completeness of the global skeleton and the visual interpretability has to be found. Some approaches ignore separatrices and restrict themselves to render meaningful glyphs at the location of the critical points [80]. Löffelmann et al. additionally use short streamlines (streamlets) to depict the local flow behavior in the direct vicinity of critical points [111] and along selected streamlines [112]. Theisel et al. reduce clutter by visualizing *saddle connectors* obtained as the intersection curves from two 2D separatrices of an attracting and a repelling saddle [185]. As the connectors fail to separate the flow into different regions, the complete surfaces can still be shown by the system on demand. Weinkauf et al. extend this concept to include separation surfaces obtained from boundary switch curves, where the flow direction is tangential to the domain boundary [205]. *Topological simplification* is an alternative approach for reducing visual complexity in the case of fields with many critical points. Tricoche et al., for example, reduce the number of critical points with clustering approaches [189] and with relevance measures [190].

The linearization assumption underlying the eigenanalysis of $\nabla \mathbf{u}$ only allows for a classification of linear flow characteristics. Multiple critical points can be induced in regions with nonlinear flow behavior that could have been better represented by a single higher-order flow pattern. Hence, nonlinear approaches can help in simplifying the global topology. Scheuermann et al. discuss these fundamental issues and propose a method based on Clifford algebra for the detection of higher-order critical points

in 2D fields [165]. Weinkauf et al. [204] place closed surfaces around higher-order critical points in 3D. On these enclosing manifolds they derive appropriate glyph representations of the higher-order points, based on the 2D skeleton computed from the projected flow. In this context, it is important to note that the detection of nonlinear flow patterns might be already prevented in the first place if low-order interpolation schemes are used for reconstructing a continuous field.

Being a concept that is not Galilean invariant, the classical vector field topology is sensitive to the chosen frame of reference when extracting critical points. In particular for moving and deforming flow structures, like vortices, it is hard to define an appropriate moving observer. This is because of the strong variation in fluid velocity that might appear near such features [149]. Hence, the classical vector field topology is primarily relevant for steady vector fields, but it can also be applied to instantaneous snapshots of unsteady fields that exhibit relatively small time dependence. This approach can be combined with critical point tracking. Nevertheless, such approaches do not account for the time-dependent dynamics of the field. The development of new topological approaches for unsteady flow is an ongoing challenge.

14.2 Topological Methods for Time-Dependent Fields

The literature on traditional (steady) vector field topology is manifold. However, there is no generally accepted concept of a time-dependent vector field topology yet. The following gives a short overview of existing works. For a more extensive survey, the reader is referred to [151]. First approaches for unsteady vector fields built upon the steady vector field topology. These methods follow a *feature tracking* scheme to establish correspondences between evolving critical points, periodic orbits, and separatrices extracted at different times. Helman and Hesselink [79], for example, linked instantaneous 2D topologies of neighboring time slices by using distance metrics. The trajectories of critical points are then visualized together with a surface representation of the evolving separatrices in a 3D space-time representation. This simple approach, however, does not account for *bifurcations*, which can appear as continuous topological transitions between discrete time slices. Discretized fields with linear time dependence, e.g., linear interpolation between time slices, allow for two cases of local bifurcations. *Hopf bifurcations* transform an attracting into a repelling focus, or the other way round, passing an unstable intermediate focus state. *Fold bifurcations*, in contrast, lead to the annihilation or creation of a connected saddle sink pair. Tricoche et al. [191] track critical points, the corresponding separatrices, and periodic orbits in unsteady 2D fields and detect the aforementioned bifurcation events. *Feature flow fields* where introduced as an alternative approach by Theisel and Seidel [184]. By adding an additional time dimension the feature flow field is constructed in a way such that the properties of a chosen feature criterion are constant along streamlines of this derived field. Hence, streamline integration can be used to track the evolution and transformation of critical points.

The traditional critical point definition based on the field velocity is not Galilean invariant, i.e., it is possible to turn every point of the domain into a critical point under an appropriate Galilean transformation. *Lagrangian equilibrium points* were proposed by Kasten et al. as a Galilean invariant generalization of the traditional critical point concept [88]. Based on the local minima of the acceleration magnitude field, their approach accounts for the Lagrangian view of moving critical points. Additionally, they propose to filter short-lived features to obtain visualizations that are more meaningful. The *motion compensated critical points* of Fuchs et al. are based on a similar idea [60]. Both approaches, however, do not provide a full description of time-dependent topology, which includes separatrices.

Topological techniques based on streamlines or stream surfaces fail to capture the true time-dependent separation behavior of the flow [175]. Theisel et al. presented an approach that determines the separation behavior of path lines locally [185], hence not allowing for an asymptotic analysis, like in the case of the steady vector field topology. The *path line oriented topology* by Shi et al. [178] accounts for this issue, however, it restricts the analysis to time-periodic vector fields. *Lagrangian coherent structures* can provide a solution to these limitations, as discussed in the next chapter.

Chapter 15

Lagrangian Coherent Structures

Lagrangian coherent structures (LCS) have become increasingly popular for studying time-dependent vector fields in recent years. They are commonly defined as manifolds that separate flow into regions of qualitatively different behavior. Thus, LCS are of lower dimensionality than the flow domain itself—they are typically of codimension one. In a topological sense LCS represent time-dependent separatrices. As particle trajectories stay in the respective regions of qualitatively different behavior over time in steady topology, this property should also hold in unsteady flows. Hence, the boundaries, i.e., the LCS, have to consist of particles too, i.e., be material lines or surfaces that advect with the time-dependent flow. While LCS have the required separating property of separatrices, a consistent concept of time-dependent critical points is still missing, including an asymptotic definition of the LCS with respect to these points, to obtain a complete time-dependent vector field topology. Note, that the generalized critical points of Fuchs et al. [60] and Kasten et al. [88] build upon an alternative approach, and thus cannot be directly used to complement LCS. In this and the following chapter we present a direct generalization of the traditional 3D vector field topology. Hyperbolic path surfaces take the role of "unsteady saddle-type critical points", and streak manifolds seeded from these manifolds represent LCS, or "time-dependent separatrices".

Section 15.1 discusses the computational aspects of LCS. While LCS are typically obtained as ridges from FTLE fields [78], only a subset of FTLE ridges adheres to the advection property, complicating the choice of appropriate FTLE scale parameters. In Section 15.2 a technique is presented that eases the choice of suitable FTLE advection times in 2D fields by measuring the discrepancy from the advection principle. The approach is based on the advection of hyperbolic points, obtained as intersections of repelling and attracting LCS. The resulting hyperbolic trajectories are later also used as seeding constructs to obtain time-dependent separatrices in our time-dependent streak-based 3D vector field topology (Chapter 16).[1]

[1] Parts of this chapter have previously been published in Sadlo et al. [159].

15.1 Computing LCS

The traditional Lyapunov exponent (LE) defined in (13.2) shares many aspects with vector field topology. By investigating the behavior of trajectories in the asymptotic limit $t \to \pm\infty$, structures resembling separatrices with locally strongest attracting or repelling behavior can be identified. The LE, similar to the traditional vector field topology, however, cannot be applied to time-dependent fields that are temporally bounded. This led to the definition of the finite-time Lyapunov exponent (FTLE) (13.3), as discussed in Chapter 13.

Lagrangian coherent structures of time-dependent vector fields are usually extracted as ridges in the FTLE field [77]. While reverse FTLE (advection time $T < 0$) results in attracting LCS, forward FTLE ($T > 0$) yields repelling LCS. According to Haller [77], Shadden et al. [175], and [159] these ridges, however, only represent LCS if they exhibit negligible cross-flux, i.e., satisfy the advection principle. Unfortunately, no concise and generic definition has been given so far what "negligible" means in that context. The LCS property depends on the chosen FTLE sampling resolution and advection time T. Theorem 4.4 in [175] would most likely suggest to use an infinite resolution and infinite advection time T for all t_0. For practical applications, a common approach is to use a high enough resolution together with sufficiently long advection times to obtain sharp and distinct ridges—that often represent LCS.

The choice of the FTLE advection time T also depends on the application as it determines the scale of the analyzed features. Short-lived features, for example, might be missed if too large advection times are chosen. While there is no general upper bound for the time scope, there is a need for a lower bound. It cannot be expected that the advection property still holds in the limit $T \to 0$, where the FTLE converges to the eigenvalue of the rate of strain tensor. Large advection times on the other hand often lead to a very fine folded ridge structure that can clutter the whole domain. In such cases the sharpest ridges can be extracted by imposing filter thresholds based on the Hessian eigenvalues and level of the FTLE [156].

Providing automatic strategies for finding a good choice of the FTLE parameters is very hard. Therefore, FTLE visualization, as other feature extraction procedures, is typically a trial-and-error procedure that can strongly benefit from interactive exploration, as discussed in Section 13.3. Shadden et al. [175] directly measured fluxes of the instantaneous velocity field across FTLE ridges as a means of verifying the advection property of given FTLE ridges. However, zero flux alone is a necessary but not sufficient condition for advection: it does not capture tangential motion, i.e., the motion component along the ridge cannot be assumed to satisfy the advection property even if the flux across the ridge is zero. In Section 15.2 we present an alternative approach that tracks distinguished points located on the ridges over time to validate the advection property [159].

Streak manifolds are another alternative to obtain LCS. According to their definition, they already fullfill the advection principle. However, in general, streak surfaces do

not feature the (asymptotic) separation property of LCS. This property can be achieved by using LCS from FTLE ridges as seeding structures, resulting in streak surfaces that inherit the separation property from the ridges. Compared to the very costly traditional ridge extraction approach, this allows for an interactive analysis related to LCS in 3D fields, as demonstrated by Ferstl et al. [50]. Using arbitrary FTLE ridges, however, does in general not lead to streak surfaces that are relevant for topology. Sadlo and Weiskopf developed a concept that goes one step further, offering a time-dependent 2D topology based on streak lines [161]. The generalization to 3D fields is presented in Chapter 16.

15.2 The Time-Scope in FTLE-Based LCS Computation

In the following, a semi-automatic technique is presented that helps in validating the LCS property of FTLE ridges in 2D fields, supporting the user in the choice of appropriate FTLE advection times T [159].

15.2.1 Method Overview

The presented technique considers the fundamental advection property of LCS to find appropriate FTLE advection times inside a prescribed temporal interval of interest. Once the user has found a sampling grid that sufficiently captures the FTLE structures of interest, has found an appropriate threshold filtering out insufficiently sharp ridges, and has found minimum and maximum FTLE advection times T_{min} and T_{max}, the technique takes over these parameters. As the result, a plot of advection discrepancy within the given range $[T_{\text{min}},T_{\text{max}}]$ is presented. Additionally, locally and globally optimal advection times are identified, and in particular, a lower bound on T with respect to a prescribed advection error is provided.

Tracking complete FTLE ridges that are extracted at discrete snapshots in time is difficult, because the ridges are not represented by identifiable particles that advect (Section 10.1). Hence, the presented technique follows a different approach: it measures the advection property only for distinguished points on the FTLE ridges, for which point correspondences can be established for different FTLE starting times t_0.

15.2.2 Tracking of FTLE Ridge Intersections

For sufficiently well defined LCS, the advection property holds for both repelling ridges from forward FTLE and attracting ridges from backward FTLE. These manifolds typically intersect at *hyperbolic manifolds* which give rise to *hyperbolic trajectories*, see Figure 15.1. For unsteady vector fields the hyperbolic trajectories play a similar role as critical points of type saddle in steady vector field topology—a fundamental observation that leads to the time-dependent topology presented in Chapter 16. If both types of ridges satisfy the advection property, this property must also hold for their intersections. As point intersections are obtained in 2D vector fields, one needs to

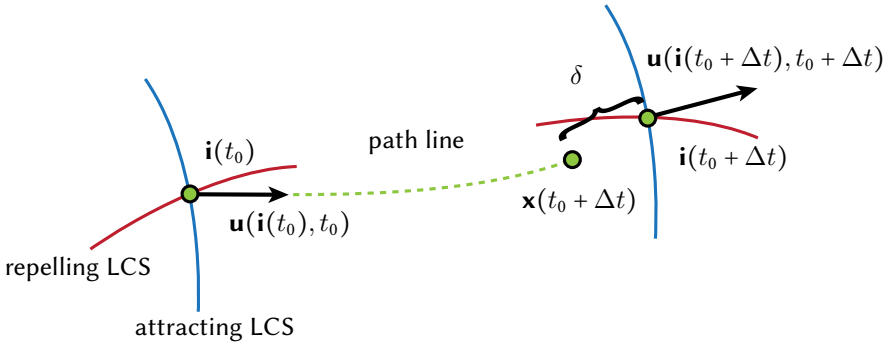


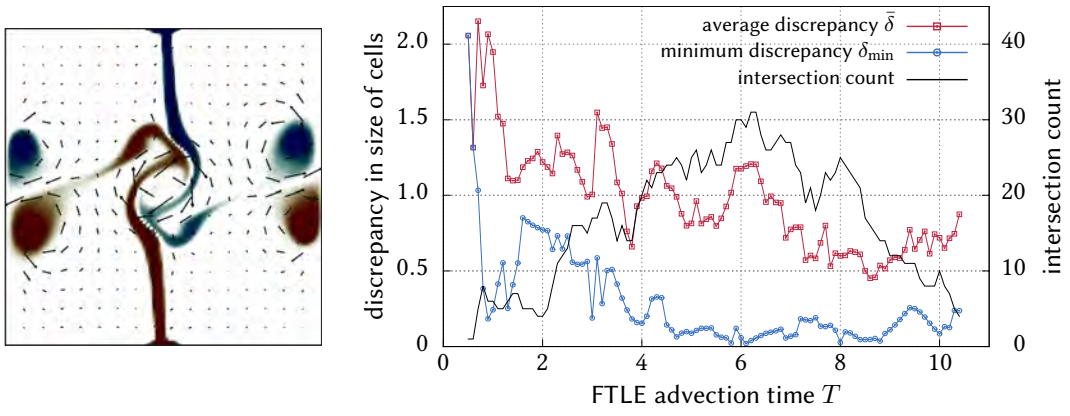
Figure 15.1 — The advection property of FTLE ridges is validated by tracking hyperbolic points $\mathbf{i}$ (green) obtained as intersections of attracting and repelling ridges at different times t_0 (left) and $t_0 + \Delta t$ (right), with fixed T. The spatial discrepancy δ is defined as the distance between the endpoint of the hyperbolic trajectory (green) and $\mathbf{i}(t_0 + \Delta t)$.

identify point correspondence between ridge intersections of successive FTLE time steps (at t_0 and $t_0 + \Delta t$). This is illustrated in Figure 15.1: ridge lines are extracted in forward and reverse FTLE fields for both t_0 and $t_0 + \Delta t$, using the Eberly criterion (10.3). This leads to two sets of intersections, $\mathbf{i}(t_0)$ at time t_0 and $\mathbf{i}(t_0 + \Delta t)$ at $t_0 + \Delta t$. A straightforward approach would use a very small Δt. This would produce almost identical ridges and hence finding correspondences between their intersections would lead to a trivial tracking problem. Further, the limit case $\Delta t \to 0$ would be used for defining the intersection velocity

$$\mathbf{u}_i(t_0 + \Delta t/2) = (\mathbf{i}(t_0 + \Delta t) - \mathbf{i}(t_0))/\Delta t. \tag{15.1}$$

Unfortunately, it turns out that ridge extraction tends to be only accurate in the order of the FTLE sampling grid cell size h. Hence, using small Δt leads to poor accuracy of $\mathbf{u}_i$ if the distance between $\mathbf{i}(t_0)$ and $\mathbf{i}(t_0 + \Delta t)$ is smaller than h. An appropriate large time span Δt can be estimated from the average speed $\bar{u}$ of the vector field and the cell size: $\Delta t = ch/\bar{u}$ with a constant $c > 1$. However, in this case, Δt is not small enough to allow for the linearization (15.1).

Sufficiently sharp ridges at t_0 are expected to fullfill the LCS property to a certain extent. For such ridges the correspondence problem can be solved by utilizing the LCS advection property. After extracting intersection points at time t_0 (see Figure 15.1), these points are advected along hyperbolic path lines to time $t_0 + \Delta t$. The advected points are then checked for correspondence with the ridge intersections at $t_0 + \Delta t$. To further avoid erroneous correspondences, a threshold limiting the discrepancy δ is imposed and the correspondence is identified as the closest remaining candidate with respect to δ. The measure δ reflects the Lagrangian advection consistency of the FTLE ridge intersections and is denoted as *advection discrepancy*. Depending on the flow Δt must not be chosen too large because ridges might be subject to deformation and disintegration, or even new ridges might appear.



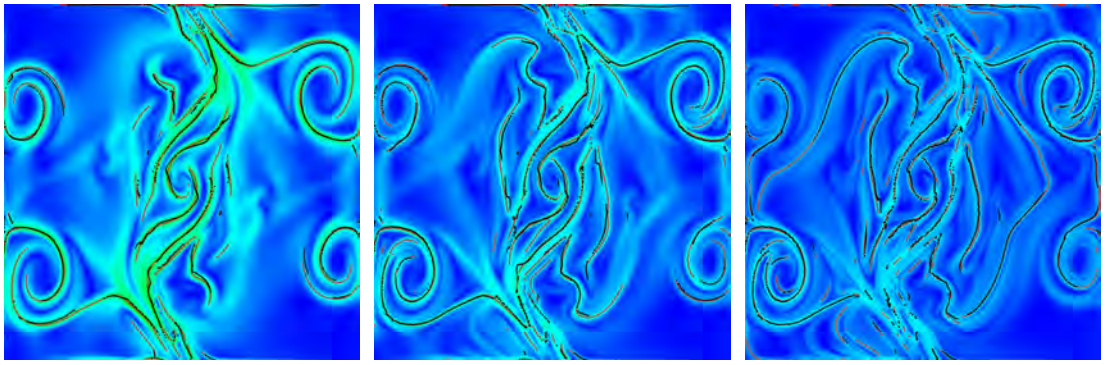
(a) (b)

Figure 15.2 — (a) Buoyant plumes data set overview. (b) Buoyancy plumes discrepancy plot: the average discrepancy $\bar{\delta}$ is clearly too high until reaching advection times of $T = 4$. Medium quality ranges can be found around $T = 6$, and comparatively high quality (lower than the size of an FTLE sampling cell) for $T = 8$ and larger.

15.2.3 Finding Locally Optimal Advection Times

In a typical scenario the field contains many intersection points, leading to a large number of discrepancy values δ. Different approaches are possible to aggregate these values into a single quality measure for the whole visualization. Potential problems are introduced by intersections that do not move or do only slowly move over time. It is likely that these intersections exhibit small δ and hence exhibit inappropriately high advection quality only because they stand still. This can be addressed by an inverse weighting of δ with the length of the corresponding hyperbolic trajectory. In practice, however, it is unlikely that a time-dependent vector field exhibits a zero over an extended time. In the following, the minimum δ of all intersections for a given advection time T is denoted as $\delta_{\min}(T)$, and the average is denoted as $\bar{\delta}(T)$. Whereas $\delta_{\min}$ tends to show the best case in terms of advection, $\bar{\delta}$ can be used to get an overall picture of the advection quality of the FTLE ridges. By measuring the discrepancy in units of the FTLE cell size h one can easily choose a limit of advection discrepancy and then visually or numerically identify which regions of T satisfy this requirement.

A uniform sampling of $T \in [T_{\min}, T_{\max}]$ is used for the analysis of advection discrepancy, and a plot is provided together with the advection times that produce the global minima of the discrepancy measures. The global minimum only serves as a simple example. More sophisticated data analysis techniques could be applied as well. The discrepancy plot answers an important question. It reveals the smallest FTLE advection time T for which the advection property is in the average satisfied up to a given tolerance. In our current approach, visual inspection of the plots is advocated, because they tend to exhibit outliers and are therefore susceptible to errors if simple automatic analysis techniques are applied.



(a) $T = 4$ (b) $T = 6$ (c) $T = 8$

Figure 15.3 — Visualization of ridge advection quality for different advection times T with an overlay of advected FTLE ridges from t_0 (black dots) and corresponding FTLE ridges at $t_0 + \Delta T$ (red lines). The advection property is sufficiently well fulfilled for $T = 8s$, in accordance with Figure 15.2 (b) ($t_0 = 20s$, $\Delta t = 1s$).

As an example, the technique is applied to a time-dependent CFD simulation of buoyant plumes inside a 2D box filled with air (Figure 15.2 (a)). The air is initially at rest and at 40°C. Gravity forces are acting downwards. No-slip conditions are used for all boundaries, and the left and the right walls are supplied with adiabatic boundary conditions. A region at the center of the lower wall is heated to 75°C. A corresponding region on the upper wall is cooled to 5°C, resulting in a complex ridge structure, as shown in Figure 15.3. Figure 15.2 (b) shows the plot from the analysis, using $T_0 = 20$ and $\Delta t = 1$. The average discrepancy $\bar{\delta}(T)$ exhibits a decreasing trend. Additionally, the number of used intersections are plotted to enable a judgment of uncertainty. From the $\bar{\delta}$ plot it is visible that $T = 4$ exhibits high error, $T = 6$ reduced, and $T = 8$ already average error below the size of an FTLE sampling cell. The global optimum inside the $\bar{\delta}$ plot is at $T = 8.6$. For validation purposes we also show the complete advected ridges (black) together with ridges at the $t_0 + \Delta t$ (red) in Figure 15.3. The images support the finding from the discrepancy measure. It can be seen that for $T = 8$ the advected ridges and the ridges of the corresponding time fit well almost everywhere. It has to be noted that the images only show the repelling ridges, i.e., those from forward FTLE. The same images of attracting ridges show a much smaller deviation. Nevertheless, according to the initial motivation of the method, such a comparison with complete advected ridges instead of intersections cannot detect tangential discrepancy between the motion of the ridges and the vector field behavior.

The presented technique complements the flux-based approach by Shadden et al. [175]. By tracking ridge intersections, not only flow discrepancy orthogonal to the ridges, but also tangential to the ridges can be revealed. Our measurements also support the theoretical behavior stated by Theorem 4.4 in [175]. The error in the advection property tends to decrease with increasing T. A generalization of the method to time-dependent 3D flow is topic of future work.

Chapter 16

Streak-Based Vector Field Topology

While several topological methods have been developed for time-dependent vector fields these methods do not offer a complete topology in the sense of the classical "steady" vector field topology introduced in Chapter 14. On the one hand there are concepts of generalized time-dependent critical points [60, 88], on the other there are LCS, the time-dependent counterpart to separatrices. However, until recently, both have not be unified consistently. This is, among other reasons, due to the fact that the most common approach for computing LCS involves several difficulties, as it is based on FTLE ridges (Chapter 15). Streak manifolds can offer an alternative approach to obtain significant LCS, provided that appropriate seed locations are used. While Ferstl et al. visualized arbitrary streak LCS [50], not aiming for a streak-based topology, Sadlo and Weiskopf [161] recognized that using *hyperbolic trajectories* as seed points can fix the aforementioned limitations. In the resulting time-dependent 2D vector field topology, the role of streamlines is replaced with streak lines. Saddle-type critical points represent degenerate *streak lines* that consist of a single point. These points move along hyperbolic trajectories as depicted in Figure 15.1. Separatrices represent streak lines converging toward those points in forward or reverse time. Sadlo and Weiskopf demonstrated that these manifolds are LCS, consistent with FTLE ridges. Besides revealing the true transport in time-dependent vector fields, their concept represents a true generalization of the traditional 2D vector field topology, as streak lines are identical to streamlines in stationary fields.

The extension of the 2D streak topology concept [161] to 3D vector fields involves several conceptual and algorithmic issues [196]. These are the topic of this chapter:[1]

Section 16.1 introduces and motivates the concepts of streak-based 3D vector field topology. The utility of *ridge intersection curves* between codimension-1 ridges in the forward and codimension-1 ridges in the reverse FTLE field as hyperbolic seeding constructs is shown. Intersections between codimension-1 and codimension-2 ridges turn out to be of limited use in general time-dependent flow. This is in contrast to

[1] Parts of this chapter have previously been published in Üffinger et al. [196].

traditional 3D vector field topology, where the counterparts are 1D and 2D manifolds intersecting at saddle-type critical points (Figure 14.2 (b)). Additionally, "reverse preadvection", i.e., reverse advection of the intersection curves prior to streak surface generation is introduced. This leads to longer streak generation phases and hence more pronounced streak manifolds.

Section 16.2 presents an accurate yet efficient extraction algorithm for the ridge intersection curves. It is based on [156] but extracts the intersections directly and is hence more efficient with respect to memory consumption and computation time. Additionally, all algorithmic aspects of the streak manifold generation process, including a new seeding strategy for the streak manifolds, are covered.

Section 16.3 evaluates the 3D streak topology with three data sets, an analytical flow and two computational fluid dynamics (CFD) results. The resulting streak LCS are not only of high quality, but visualizing their growth provides insight in the dynamics of hyperbolic regions, e.g., how they cause LCS.

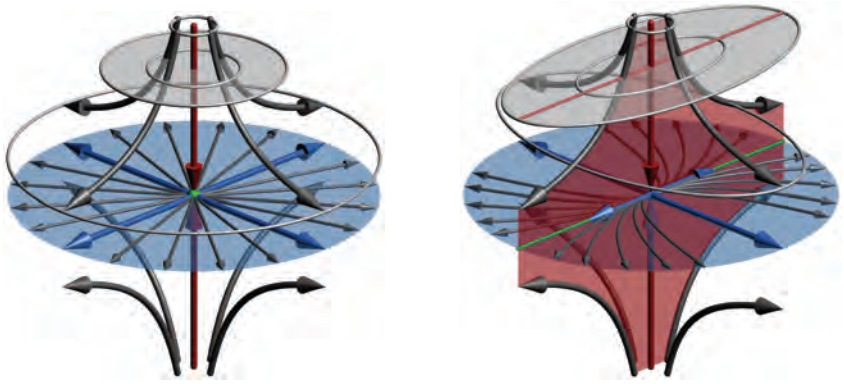
Section 16.4 summarizes the results and discusses open questions that have to be addressed in the future.

The presented method shares one limitation with all other topology-related works that are based on ridges in FTLE fields. It has to be ensured that the ridges used to extract the hyperbolic regions adhere to the LCS advection principle. Since a direct validation technique so far only exists for 2D fields (Section 15.2), the common practice of filter thresholds is employed to obtain ridges that are sufficiently sharp and thus represent LCS. Moreover, it has been demonstrated in the 2D concept [161] that it is sufficient to additionally require simple hyperbolicity of the intersections, i.e., both positive and negative real eigenvalues of $\nabla \mathbf{u}$. For the 3D topology this weak hyperbolicity is extended, and it is demonstrated that it is a sufficient criterion for obtaining streak LCS that are consistent with respective FTLE ridges.

16.1 Streak Topology Concepts for 3D Fields

This section describes the extension of the method for 2D vector fields by Sadlo and Weiskopf [161] to 3D vector fields. A time-dependent 3D vector field topology is derived based on saddle-type (hyperbolic) sets of path lines and their separatrices consisting of streak manifolds.

Similar to other finite-time concepts, like FTLE, the 3D topology relates to a point in time t_0 and a finite advection time T_s, which serves as a scale parameter. As in the 2D approach, the intersections of ridges in the forward FTLE field $\sigma_{t_0}^{T}$ and the reverse FTLE field $\sigma_{t_0}^{-T}$ are used as seeding constructs for the streak manifolds. Hence, to obtain *hyperbolic ridge intersections* (Section 16.1.1) two FTLE fields have to be computed, both using advection time T and starting at time t_0. As ridges in $\sigma_{t_0}^{T}$ represent repelling LCS, and those in $\sigma_{t_0}^{-T}$ represent attracting ones, their intersections



(a) (b)

Figure 16.1 — Saddle-type critical point (green sphere) in vector field $\mathbf{u}$ with streamlines (gray arrows) and eigenvalue-weighted eigenvectors of $\nabla\mathbf{u}$ (red and blue arrows). (a) Isotropic (degenerate) case. Contour lines (gray disc) visualize forward FTLE: it has isotropic profile, hence best represented by a 1D ridge (red arrow, consistent with stable manifold). (b) Anisotropic (non-degenerate) case. Forward FTLE has anisotropic profile, hence best represented by a ridge surface (red) oriented along red line in the profile. The green curve represents the intersection of the forward and reverse (blue) FTLE ridge surface.

exhibit saddle-like characteristics (Figure 16.1). Since high resolution FTLE fields are crucial for obtaining accurate seeding structures and because FTLE computation is very expensive we present an efficient adaptive technique for the accurate extraction of ridge intersections in Section 16.2. However, to ease description in the following it is assumed that complete ridges are extracted first and then intersected. For *streak manifold generation* (Section 16.1.2) the extracted hyperbolic intersections are advected with the flow for time T_s while producing seeding structures for the streak manifolds. In other words, a path line is generated for each point of the intersections and during integration of the path line particles are continuously released at the front of the path line. These particles represent the space-time streak manifolds. Streak generation is only performed as long as the respective path line stays hyperbolic, i.e., particles are only released in hyperbolic regions.

16.1.1 Hyperbolic Ridge Intersections

In the 2D approach [161], isolated hyperbolic *trajectories* have been identified by *hyperbolic intersections* of ridge lines in the forward and reverse FTLE fields (see Figure 15.1). These were then used for seeding streak *lines* over time, resulting in 2D *streak manifolds* in space-time. This is consistent with traditional vector field topology in terms of critical points and separatrices, as for instants of time, path lines represent points, and the streak manifolds represent lines.

As the concept is extended to 3D, we encounter the question if one should intersect 1D ridges with 2D ridges or 2D ridges with 2D ridges. Figure 16.1 illustrates a 3D saddle-type region where the flow is attracted by the (unstable) 2D manifold and is repelled from the (stable) 1D manifold. In this case two positive and one negative eigenvalues of the Jacobian $\nabla\mathbf{u}$ encode the hyperbolic behavior (1 in, 2 out) of the flow field $\mathbf{u}$—two eigenvectors spanning the surface, and one eigenvector oriented along the 1D manifold, respectively. In the isotropic (degenerate) case the positive eigenvalues are equal (Figure 16.1 (a)), whereas in the anisotropic (practical) case they are not (Figure 16.1 (b)). Hence, extracting ridge *lines* from FTLE is, in a strict sense, an ill-posed problem. Due to the anisotropy in the FTLE profile (gray disc in Figure 16.1 (b)) there is always a 2D ridge that fits better than a respective 1D ridge. Although 1D FTLE ridges have been given no attention so far in the visualization of 3D vector fields, they represent coherent structures in the sense that they tend to exhibit locally highest hyperbolicity [77]. However, like 1D manifolds in traditional 3D vector field topology, they fail at separating regions of different behavior.

In the following, first, a possible 3D streak topology based on 1D/2D ridge intersection is outlined and its shortcomings are discussed. This is followed by the explanation why 2D/2D ridge intersections are used throughout our approach. An accompanying video that helps in understanding the presented concepts has been published on the IEEE website.[2]

1D/2D Ridge Intersection

With 1D/2D ridge intersection one can extract the 1D ridges either from the forward or from the reverse FTLE field (and the 2D ridges vice versa). Although both results in the same intersection points in simple cases (Figure 16.4 (a)), it might affect the resulting intersections in complex cases. The following discussion is restricted to the (1 in, 2 out) case (Figure 16.1), the opposite case is obtained by reversal of the vector field). In a first step FTLE ridges are extracted from the forward and reverse FTLE field at time t_0 to obtain repelling and attracting LCS (Figure 16.2 (a)). The 1D ridges are extracted with the parallel vectors approach while the 2D ridges are formulated according to Eberly (Equation (10.3)). The intersection points are then advected for time T_s and the initial portion of the resulting path lines residing in hyperbolic regions are the *hyperbolic path lines* (Figure 16.2 (b)) which would serve for seeding structures for streak manifold generation. As in the case of traditional vector field topology and its time-dependent variant [161], the seeding points for the streak lines have to be slightly offset from the path line to allow the streaks to grow. A common choice is to apply an eigenanalysis of $\nabla\mathbf{u}$ which reflects the local flow behavior. In the isotropic case (equal positive eigenvalues), one could simply generate a seed circle spanned by the major eigenvectors to obtain a streak surface that coincides with the attracting LCS (Figure 16.2 (c) and (d)). In anisotropic cases, where the medium eigenvalue is smaller

[2] http://www.computer.org/cms/Computer.org/dl/trans/tg/5555/01/extras/ttg2012990066s.avi

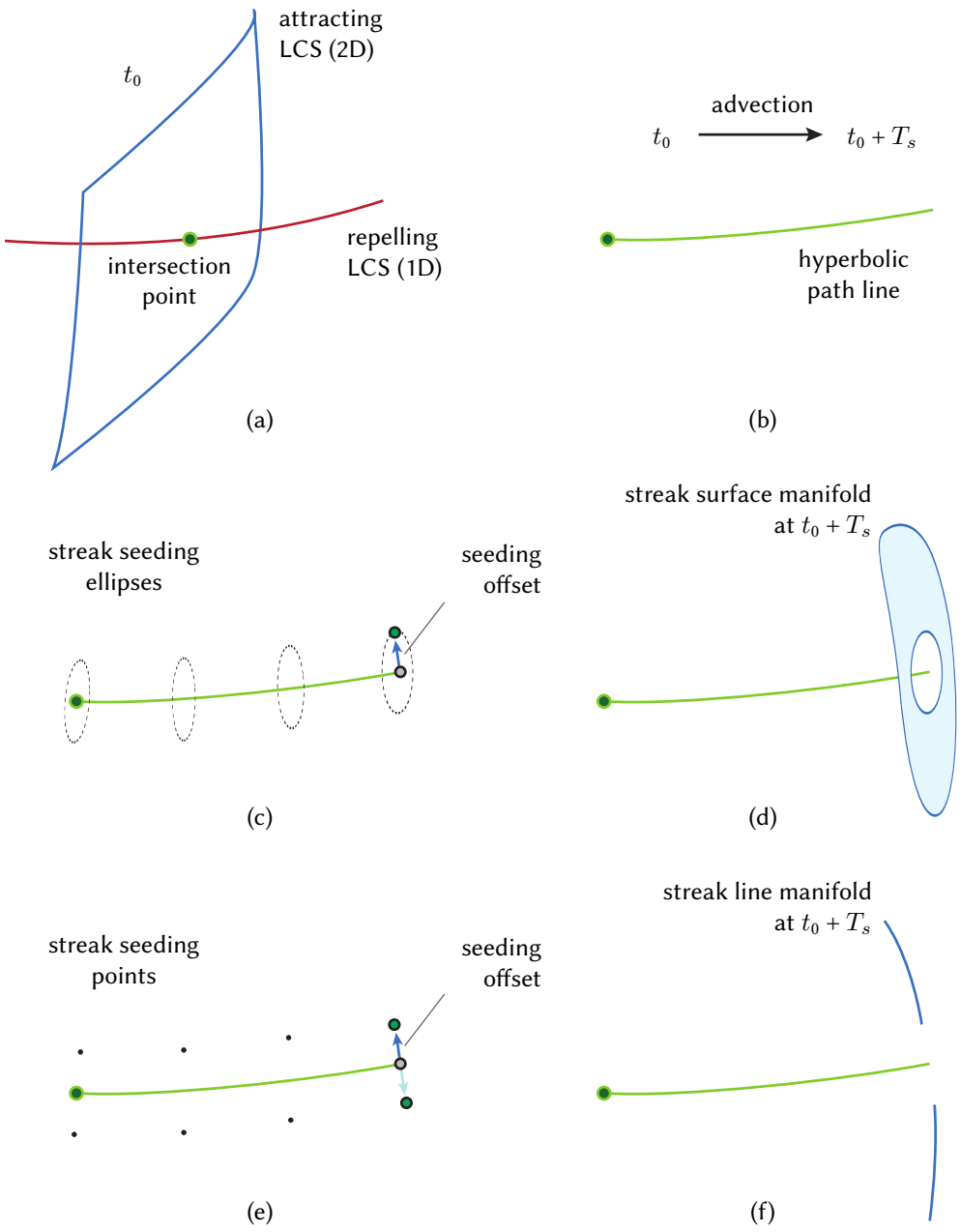


Figure 16.2 — Streak manifold construction for 1D/2D ridge intersection case. (a) Forward (1D, red) and reverse FTLE (2D, blue) ridges at t_0 are intersected, resulting in ridge intersection point (green). (b) The point is advected for time T_s, generating a hyperbolic path line. For non-degenerate cases (non-vanishing median eigenvalue of $\nabla \mathbf{u}$) ellipse seeding structures along the path line (c) generate a streak surface manifold (d). In degenerate cases seeding points (e) generate a streak line manifold (f).

but nonzero, a seeding ellipse corresponding to the major and medium eigenvalues (-vectors) would produce a streak surface consistent with the attracting LCS. However, if the medium eigenvalue is comparatively small (strong anisotropy), the seeding circle would degenerate to a line. Hence, using two seeds offset along the major eigenvector would produce a line-type subset of the desired attracting LCS (Figure 16.2 (e) and (f)).

2D/2D Ridge Intersection

For the 2D/2D case, ridge *surfaces* are extracted from both forward and reverse FTLE, with subsequent ridge intersection, as illustrated in Figure 16.3 (a). The resulting intersection curves are advected for time T_s, resulting in path surfaces. The path surface parts within hyperbolic regions constitute the *hyperbolic path surfaces* (green in Figure 16.3 (b)). The hyperbolic path surfaces take the role of critical points: they are used for seeding the streak manifolds at their front, see Figure 16.3 (c) and (d). Again, eigensystem-based offsetting is applied to generate the seeding curves for streak surface generation, described in Section 16.2.2.

Discussion

Figure 16.4 demonstrates the problems of 1D/2D ridge intersection in the simple synthetic Gyre-Saddle data set (Section 16.3.1) with $2:1$ anisotropy, which is below anisotropy in typical CFD data sets. Several stray 1D ridges are obtained (Figure 16.4 (a)). In contrast, 2D/2D ridge intersection is typically more robust, especially for FTLE fields from CFD data. Figure 16.4 (b) and (c) highlight the main advantage of the 2D/2D ridge intersection approach in anisotropic configurations. With the seeding *point* resulting from 1D/2D ridge intersection, only a small elongated streak surface is obtained, covering only a small fraction of the corresponding LCS at time $t_0 + T_s$ (Figure 16.4 (b)). Please note that using infinite advection time, as used in traditional vector field topology to allow the manifold to grow to the desired size also in "perpendicular direction", is usually not possible in time-dependent vector fields due to limited time domain and temporal variation. The seeding *line* resulting from the 2D/2D approach instead generates streak surfaces that capture the LCS much better within the available time scope (Figure 16.4 (c)). All this motivates the 2D/2D ridge intersection approach.

16.1.2 Streak Manifold Generation

As motivated in the previous section the method employs hyperbolic seeding curves from 2D/2D ridge surface intersection for streak manifold generation. Two offset seeding structures generate the two parts of the streak manifold at the front of each hyperbolic path surface during integration time T_s (Figure 16.3 (c) and (d)). The resulting streak surfaces are generalized streak manifolds [209], as the seeding location is moving with the flow in general time-dependent fields.

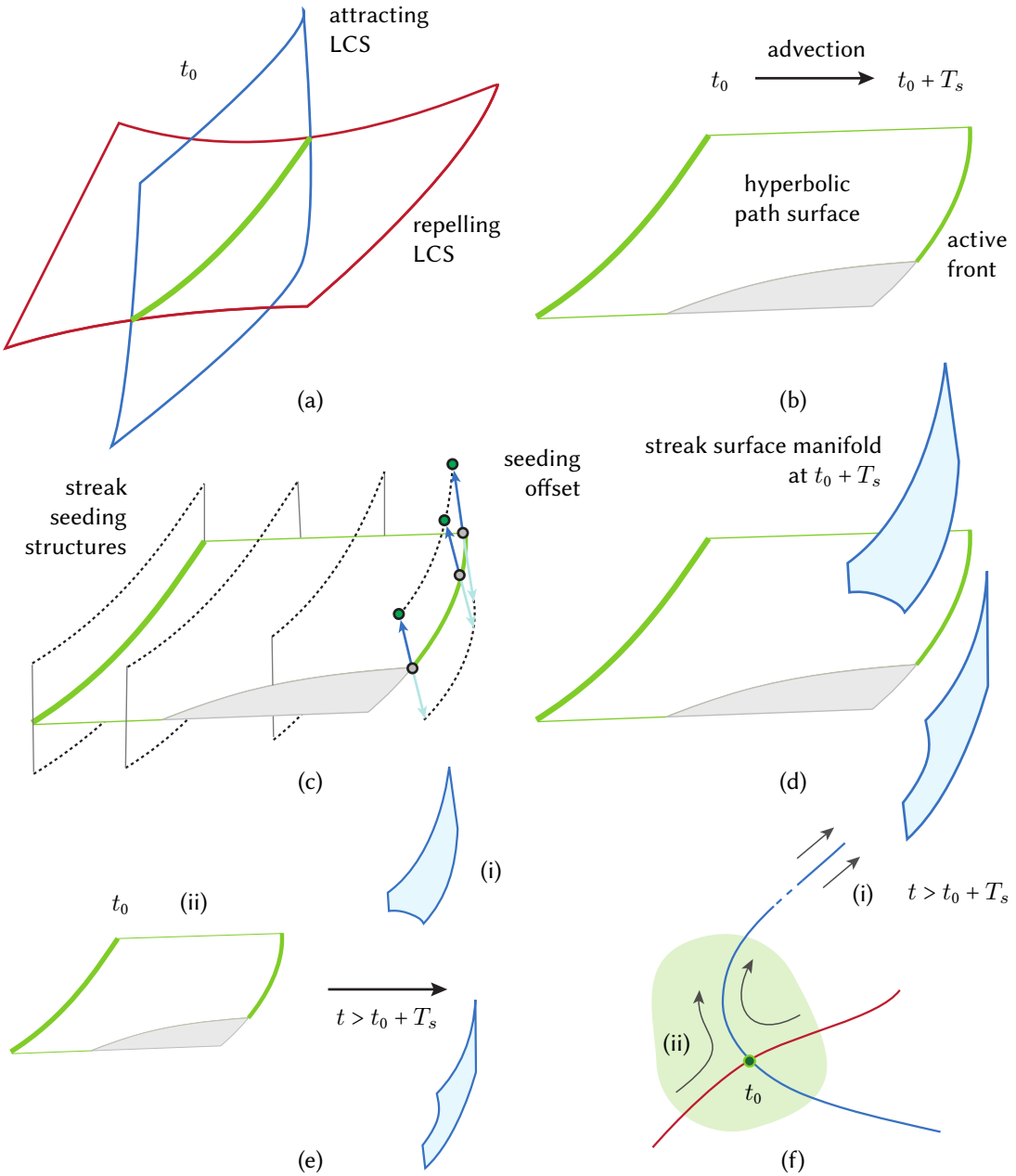


Figure 16.3 — Streak manifold construction for 2D/2D ridge intersection case. (a) Forward (red) and reverse (blue) FTLE ridges at t_0 are intersected, resulting in a ridge intersection curve (green). (b) The curve is advected for time T_s, resulting in a hyperbolic (green) path surface. (c) During integration, two seeding curves are generated by offsetting from the path surface front in direction of the major eigenvector (blue) of $\nabla \mathbf{u}$. (d) Resulting streak manifolds on either side of the hyperbolic path surface. (e) Streak surfaces can be advected even after generation has stopped, corresponding to remote space-time relation between (i) and (ii) in traditional vector field topology (f).

(a) (b) (c)

Figure 16.4 — (a) 1D and 2D forward (red) and reverse FTLE ridges (blue) allow three intersection cases: intersection point from 2D reverse and 1D forward FTLE ridges, the reverse case, and 2D/2D ridge intersection with resulting curve (green). (b) Streak manifold strip (dark blue) from 1D/2D intersection point together with matching FTLE ridge (light blue) and hyperbolic path line (green). (f) Superior streak surface (dark blue) from 2D/2D ridge intersection with hyperbolic path surface (green).

The growth of the streak surfaces depends on the hyperbolicity (repulsion) along the hyperbolic path surface and on the advection time T_s during which it acts on the streak surface. *Reverse preadvection* is introduced to allow for increased T_s and thus larger streak manifolds for more significant visualizations. The basic idea is to advect the ridge intersection curves in reverse-time direction for time T_p prior to streak generation. To simplify the streak generation stage and to allow the interpretation of hyperbolicity by the shape of the resulting streaks (Figure 16.15 (a)), the time T_p is limited such that the curves stay completely inside hyperbolic regions. Starting at the new space-time location of the seeding curve, streak generation for additional time T_p, provides larger streak manifolds. This is illustrated at the von Kármán street example in Figure 16.12.

Discussion

As in the 2D approach, the time T_s during which a streak manifold can be robustly generated is limited by several factors.[3] Note that preadvection time T_p does, except for numerics, not account in our reasoning because the intersection curve error that grows during reverse advection for time T_p is subject to annihilation (due to error reduction) during subsequent forward advection for time T_p as the streaks are generated. Similar to the 2D approach, a conservative choice $T_s \leq T$ with respect to FTLE advection time T is required. Furthermore, T_s is limited by the accuracy at which the intersection curve is extracted. This accuracy depends on the advection property of the ridge surfaces, i.e., on their sharpness and on the FTLE resolution, both not yet linked to T by appropriate models (see Section 15.1). Hence, insufficiently sharp ridges are rejected and T_s is limited by comparing the streak manifolds with corresponding FTLE ridges from time to time (see Figure 16.13). The same applies to the choice of the seeding offset distance (Section 16.2.2), which is chosen manually. The offset has to be chosen

[3] The streak manifold geometry has to be consistent with LCS.

large enough to allow the streak surface to escape the hyperbolic region and to grow. It is important to note, that the aimed LCS are attracting in the respective time direction, and hence, the streak manifolds are advected toward the LCS, reducing errors. Finally, streak generation, as already mentioned, is stopped when the respective part of a hyperbolic path surface front enters a non-hyperbolic region.

All in all, streak generation may take place inside the time interval $[t_0 - T_p, t_0 + T_s]$, provided that the respective part of the hyperbolic path surface front has not entered a non-hyperbolic region. Although streak surface *generation* is constrained to that time period, *advection* of a streak surface can be carried on even if its generation has stopped (Figure 16.3 (e)). The reason is that these streak surfaces still exhibit the property of separatrices: if a particle is seeded on either side of the surface and both the particles and the surface are advected back to the time interval $[t_0 - T_p, t_0 + T_s]$, the particles will separate at the hyperbolic region along the streak surface. This is in accordance with the fact that in traditional vector field topology separatrices may be locally indistinguishable from surrounding streamlines (see (i) in Figure 16.3 (f))—the relation to the hyperbolic region (ii) is remote in space-time, as in our case. To prevent clutter, one can decide to limit the advection time to an appropriate value after streak generation has stopped. In our examples, the advection was stopped at time $t_0 + T_s$.

Please also note that the presented approach performs streak surface integration within this time interval, i.e., it does not produce a single result but the streak integration itself visualizes the dynamics of LCS within the time interval. FTLE ridges, in contrast, visualize only an instant of time. Time series visualization of the advected and growing streak surfaces provide additional information within the space-time region of interest. One example is the amount of streak surface growth, which directly reflects the action of hyperbolicity.

As in the case of separatrices in traditional vector field topology, the streak manifolds are computed in both forward and reverse time. The application of the complete approach in reverse time direction within $[t_0 - T_s, t_0 + T_p]$ is straightforward, and therefore description in the following is constrained to forward-time extraction. Time reversal, however, switches the roles of the eigenvectors of the Jacobian $\nabla\mathbf{u}$ and therefore the computation of the seeding offsets has to be altered accordingly.

16.2 Adaptive Streak LCS Algorithm

The overall algorithm consists of two stages (Figure 16.5). In a first step (Section 16.2.1), the intersection curves between the ridge surfaces in the forward and the reverse FTLE field are extracted (Figure 16.3 (a)). Since even comparatively small perturbations of the position and shape of the intersection curves can deteriorate the utility of the hyperbolic path surfaces (Figure 16.16), high resolution FTLE sampling is a prerequisite for achieving the required precision in ridge intersection extraction. FTLE evaluation, however, has a high computational cost because a trajectory needs to be integrated

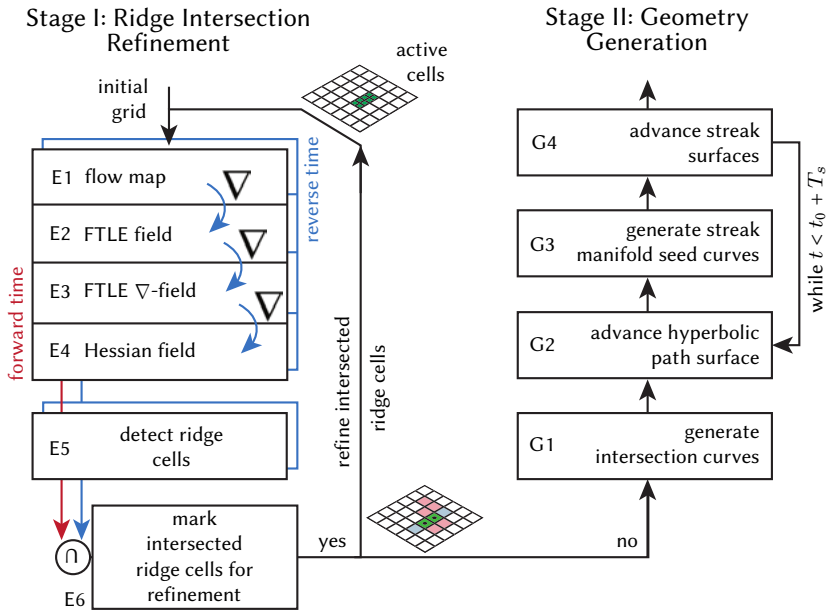


Figure 16.5 — Data flow pipeline for streak manifold generation.

for each sample of the flow map. Adaptive FTLE sampling techniques (Section 13.4.2) can be used to reduce this cost. Section 16.2.1 presents a new iterative refinement approach, using sparse sampling grids to be able to efficiently detect and refine the regions containing hyperbolic ridge intersections. The method is inspired by the AMR approach by Sadlo and Peikert [156], but follows a sparse multi-grid approach. In a second step the streak manifolds are generated using the previously extracted ridge intersection curves. For this a path surface is seeded from each intersection curve (Figure 16.3 (b)) and two streak surfaces are generated at its front during integration (Figure 16.3 (c) and (d)). The method and its derivation from the 2D streak topology are described in Section 16.2.2. This section concludes with an investigation of the computational complexity of the overall method (Section 16.2.3).

16.2.1 Stage I: Ridge Intersection Refinement

As in the 2D streak topology [161], the concept central to ridge intersection refinement are *ridge cells*. A ridge cell is a cell of the FTLE sampling grid that has at least one of its edges intersected by an FTLE ridge surface. The Eberly criterion (10.3) is used to detect these cells. Additionally, minimum FTLE and Hessian filters are imposed (Section 15.1). Having ridge cells in the forward and in the reverse FTLE field, the conjunction of these sets represents the set of *intersected ridge cells*. These cells, which possibly contain the ridge intersection curves, are refined by subdivision and then represent the set of *active cells* for the next refinement pass (Figure 16.5). Since the gradient estimation and Hessian estimation operators require additional support, a neighborhood of vertices is maintained around active cells during computations (Figure 16.6).

The main difference to the 2D approach is that instead of extracting ridge surfaces in isolated FTLE fields, only the intersection between ridges in the forward and reverse FTLE field is refined. This results in a substantial speedup and memory conservation, as discussed in Section 16.2.3. To speed up the tracing of the still large number of FTLE path lines the advection is performed on the GPU using a fourth order Runge-Kutta integrator implemented in CUDA. Another difference is that the difficulties coming with AMR data structures are avoided by following a multigrid approach. The multigrid consists of several Cartesian grids that differ by a power of 2 in cell size. These grids are stored in a sparse manner and only the finest level is kept during refinement, i.e., only cells are represented that are needed during refinement.

The process starts with an initial FTLE evaluation on a comparatively coarse uniform grid inside the region of interest. For the experiments, a resolution was used that oversamples the grid of the underlying vector field by a factor of two, although other resolutions are possible. If a too low initial sampling resolution is chosen, ridge intersections can be missed. This issue is common to most adaptive schemes, including those for FTLE computation. Although the GPU prototype is restricted to vector field input on uniform grids, it can be easily extended to unstructured grids. Only the integration of path lines and the estimation of the vector field Jacobian have to be adapted accordingly, e.g., by using the approach due to Garth and Joy [64]. In the following, the details of the pipeline execution are outlined with ridge intersection taking place at time t_0 and subsequent forward-time streak manifold generation from time t_0 to $t_0 + T_s$. The algorithmic steps for extracting ridge intersections are described next. They are executed once for forward and once for reverse FTLE (Figure 16.5, E1–E5).

For each refinement level, first the flow map is computed at the new grid vertices (Figure 16.5, E1). Then, the F-FTLE (13.3) is computed from the flow map (E2). To perform ridge detection according to Eberly [40] (Section 10.1) on the edge level of the grid, the gradient of the FTLE is computed at the active grid vertices (E3) and the Jacobian operator is applied once more to obtain the Hessian of the FTLE field (E4). Alternatively, the Hessian could also be computed directly from the FTLE field. The indirect way however allows for a straightforward implementation of the refinement algorithm on the sparse grid. Since eigenvectors lack orientation, and to make sure that edge intersections are consistent for all cells adjacent to an edge, the two instances $\mathbf{e}_3'$ and $\mathbf{e}_3''$ of the minor Hessian eigenvector are oriented consistently at the two end points of the edge (assuring $\mathbf{e}_3' \cdot \mathbf{e}_3'' \geq 0$). Then the ridge cells are detected (E5), and subsequently the cells are intersected (E6). Figure 16.6 illustrates the refinement of the intersected ridge cells and the selection of the required neighborhood in reverse computation order (green path) to account for the computation's operator stencils on the refined level (blue path, E1–E5).

Lowered FTLE thresholds are used to account for the FTLE underestimation issue, which was explained in Section 13.4.2. Other strategies, such as the look-ahead approach described in [156], could be used as well. Similar to [156] cells are marked that

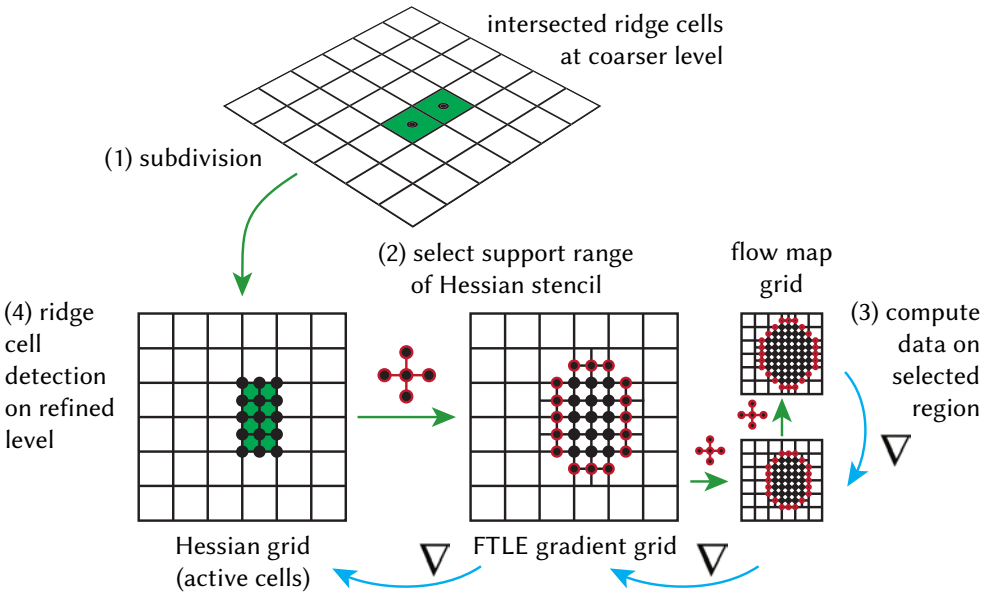


Figure 16.6 — Adaptive refinement of ridge cells. For clarity a 2D grid and a simple gradient stencil is shown. The green path shows the selection of the neighborhood with respect to the stencil of the active cells in the Hessian grid (green). Data required on the refined level is computed for forward and reverse FTLE (blue path).

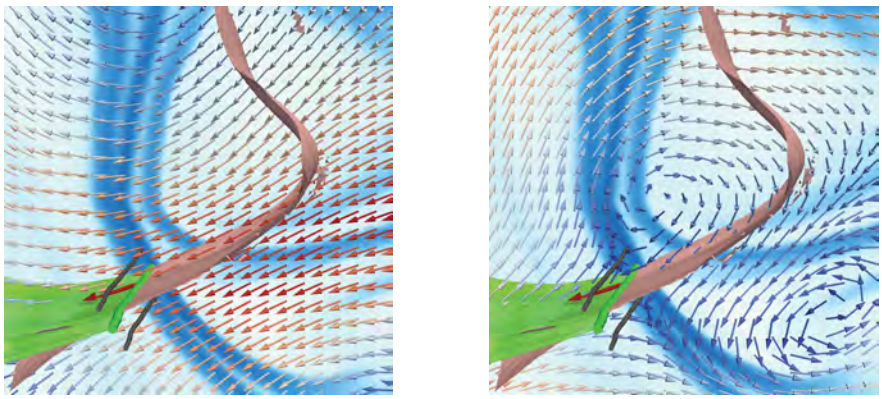
are adjacent to ridge cells to enable detection of new ridge cells when increasing the sampling resolution. This allows the set of intersected ridge cells to grow at the ends of the intersection curves during refinement.

16.2.2 Stage II: Geometry Generation

The input to this stage is the set of intersected ridge cells, which possibly contain ridge intersection curves. The algorithm first extracts the line geometry of intersections from these data, and then proceeds with streak manifold generation from these curves.

Intersection Curve Extraction

The intersection curves between the ridge surfaces in the forward and reverse FTLE field are determined from the set of intersected ridge cells in a cell-wise manner (G1 in Figure 16.5). For each of these cells, the ridge surface case is evaluated using marching cubes on the Eberly criterion (10.3), and the resulting triangles (at most 5 plus 5 from forward and reverse FTLE, respectively) are intersected to obtain the segments that compose the polyline representation of the intersection curves. Since the resulting curves shall serve as seeding curves for hyperbolic path surfaces, segments that are located outside hyperbolic regions are rejected first. Having the segments connected, the lines are filtered with respect to their length, i.e., curves below a



(a) (b)

Figure 16.7 — Eulerian view (a) compared with Lagrangian view (b) of the vector field $\mathbf{u}$ near a ridge intersection curve (green) (Buoyant Flow data set at $t_0 = 50.02\,\mathrm{s}$). The bold red arrow and the hyperbolic path surface (green) depict the motion of the intersection. In the frame of reference of the intersection (b) it is revealed that the two streak surface seed curves (black) should be offset in direction of the major eigenvector of $\nabla\mathbf{u}$.

predetermined length are rejected. Filtering features by size is a common procedure in feature extraction to get rid of spurious or insignificant solutions. This allows to produce LCS visualizations without unnecessary clutter.

Streak Manifold Generation

The streak manifolds are generated by stages G2 to G4 of the pipeline (Figure 16.5). The ridge intersection curves extracted at time t_0 are advected with the flow for streak advection time T_s in an iterative manner, each step producing a new instance of the advancing front of the hyperbolic path surface. In the 2D streak topology approach the streak generating path line is terminated when it enters a non-hyperbolic region, i.e., when $\det(\nabla\mathbf{u}) \geq 0$. This includes cases with complex eigenvalues of $\nabla\mathbf{u}$. We apply a corresponding filtering to our path surface front: those parts exhibiting complex eigenvalues of $\nabla\mathbf{u}$ or real eigenvalues of equal sign (non-hyperbolic case) are stopped, i.e., no further path surface is generated therefrom (see Figure 16.15 (a)). Two streak surfaces are generated from each remaining path surface front part, as described in Figure 16.3. The fronts of the hyperbolic path surfaces cannot be directly used as seeding structures for the generalized streak surfaces. This would simply reproduce the advected seed curve, i.e., the streak surface would degenerate to its seeding curve moving along the hyperbolic path surface. Conceptually, the vertices of the front need to be offset to both sides of the repelling LCS to allow the streak surfaces to escape the hyperbolic region, as illustrated in Figure 16.7. Hence, we use the major eigenvector of the Jacobian $\nabla\mathbf{u}$ as offsetting direction, as shown in Figures 16.3 and 16.7 (b).

The offset vectors have to be oriented consistently in space and time for both seeding

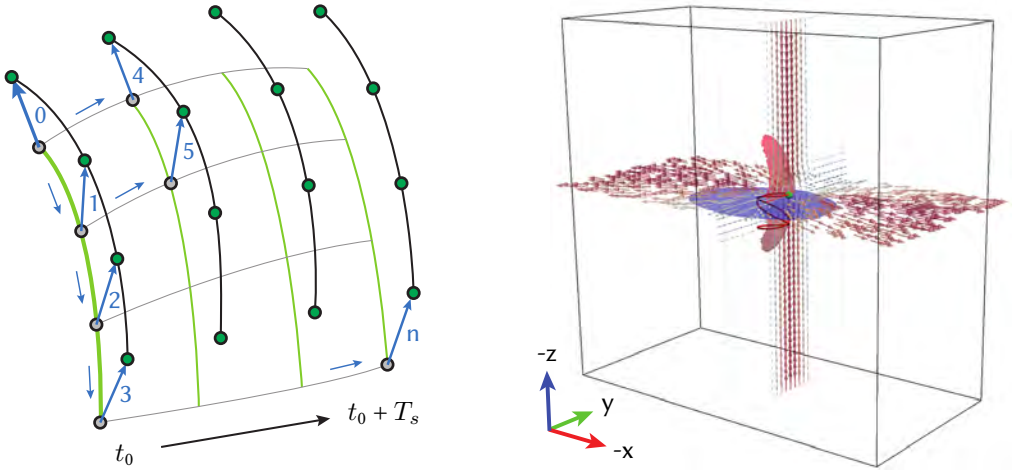


Figure 16.8 — (a) Offsetting seed curves (black) at the front of the hyperbolic path surface (green). (b) Gyre-Saddle example at $t = 3.5$ s. The saddle-type critical point of the field (green) translates along a Lissajous curve (red curve).

curves to prevent the two parts of the streak manifold flipping from one side to the other side of the repelling LCS during their generation. Figure 16.8 (a) illustrates how consistency is achieved. Eigenvector orientation at the first vertex at time t_0 is arbitrarily chosen (bold blue arrow, 0) and made consistent in space only by propagating the orientation along the front of the path surface (green curve), e.g., $0 \rightarrow 1$. At later advection instances the offset vector orientations are chosen consistently with their direct predecessors in time, e.g., $0 \rightarrow 4$. In cases where the offset vectors are tangent to the path surface our approach also produces consistent streak manifolds. In our current approach, we keep the length of the offset vector constant during the whole generation process. With this choice, the growth of the streak manifolds directly visualizes their hyperbolic strength, enabling a quantitative comparison of the different hyperbolic regions, as shown in Figures 16.14 (b) and 16.15. As an alternative, to close the offset gap at the end of the streak generation process, one could let the length of the offset vectors drop to zero as approaching $t_0 + T_s$. This, however, would not reveal the seeding offset in the visualization and therefore we did not apply this strategy. Regarding the quality of the streak surfaces we achieved high quality results with our prototype. Nevertheless, the streak surface integration part could be replaced with an adaptive streak surface integration algorithm (Section 5.4) to alleviate potential issues in problematic field regions, such as regions with strong divergence $\nabla \cdot \mathbf{u}$.

16.2.3 Complexity Analysis

In the following, we analyze the computational complexity of our algorithm and compare it to the traditional approach of obtaining LCS (time series) as ridges from

FTLE fields. For the complexity of the ridge intersection curve extraction we assume a regular grid with resolution N^3. The straightforward approach for computing a single FTLE instance on a regular grid has complexity $\mathcal{O}(N^4)$—the cost of the flow map path lines $\mathcal{O}(N)$ times the resolution $\mathcal{O}(N^3)$. Our adaptive approach starts with a coarse base resolution of M^3, with $M \ll N$, and then, during refinement, focuses its computational effort on 1D subregions of the region of interest, which potentially contain ridge intersection curves. With k refinement iterations, and M chosen such that $2^k M = N$, our method has costs $\mathcal{O}(M^3 N + \sum_{i=1}^{k} 2^i MN)$. This equals $\mathcal{O}(M^3 N + 2(N - M)N)$, and with $k \to \infty$ the ridge intersection stage has complexity $\mathcal{O}(N^2)$. This is backed by our measurements provided in Section 16.3.1. There, k is naturally limited and the operations on the sparse grid during refinement have a larger constant factor, as revealed in Table 16.1. A similar complexity analysis yields $\mathcal{O}(N^3)$ for the adaptive ridge surface extraction of Sadlo and Peikert [156]. For the second stage of our approach, namely the streak surface generation from ridge intersection curves, the size of the intersection curves is in the order of $\mathcal{O}(N)$. Since streak lines exhibit complexity $\mathcal{O}(N^2)$, our streak surfaces have complexity $\mathcal{O}(N^3)$, leading to an overall complexity of $\mathcal{O}(N^3)$ for our approach. Nevertheless, our method delivers the dynamics of hyperbolic LCS within the whole time interval $[t_0 - T_p, t_0 + T_s]$ at no extra cost, i.e., the streak generation process already represents a visualization of LCS dynamics within $[t_0 - T_p, t_0 + T_s]$. Extracting corresponding time series with the traditional FTLE ridge-surface approach, in contrast, has complexity $\mathcal{O}(N^5)$ and $\mathcal{O}(N^4)$ for the adaptive ridge extraction [156], respectively.

16.3 Results and Evaluation

We show results of our method at the example of three time-dependent data sets. First the synthetic Gyre-Saddle data set is investigated, containing a single prominent hyperbolic region. It enables us to exemplify the different aspects of our method, which would be harder to achieve with CFD data. It is also used for a detailed analysis of the adaptive refinement algorithm presented in the previous section, and is used to compare the quality of our streak LCS with the quality of FTLE ridges. We then apply our method to a von Kármán vortex street CFD simulation and exemplify the usability of reverse preadvection, as well as the superior quality of streak LCS. Lastly, a buoyant flow with a complex LCS structure is investigated. An overview visualization of the domain first helps in selecting a region of interest (ROI) containing a subset of prominent streak manifolds. These are then used for an analysis of the LCS advection principle and to demonstrate how visualizing streak growth can help in understanding the dynamics of hyperbolic flow regions. In cases where it was sufficient to employ a single filtering threshold (minimum FTLE τ_σ or maximum minor Hessian eigenvalue τ_H only) we omit the respective value of the other filter ($\tau_H = 0$ or $\tau_\sigma = -\infty$). Timings are given for a quad-core Intel CPU at 2.67 GHz, and an NVIDIA Fermi 480 GTX GPU.

(a) $t_0 = 3.5\,\mathrm{s}$ (b) $t_0 = 4.49\,\mathrm{s}$ (c) $t_0 = 5.5\,\mathrm{s}$

Figure 16.9 — Ridge intersection curves (green) of attracting (blue) and repelling FTLE ridges (red) extracted at different t_0, illustrating the dynamics of the hyperbolic region.

16.3.1 Time-Dependent Gyre-Saddle

The synthetic Gyre-Saddle field (Section 16.3.1) serves as a means to illustrate the concepts and results of our method at a simple saddle-type region. A detailed analysis of the timings and the corresponding problem sizes on the different refinement levels is given. Moreover, the simple FTLE ridge structure of this data set lends itself well to demonstrate the superior accuracy of our streak manifolds compared to the traditional visualization by FTLE ridges.

Data Set Description and Experiment Setup

The straightforward choice for a simple hyperbolic region would be a linear saddle field. However, for a Lagrangian (Galilean-invariant) concept such a field is futile: it exhibits uniform Jacobian and hence, e.g., a uniform FTLE field without ridges. Therefore, we have constructed a 3D variant of the 2D example presented in [161], exhibiting a cosine velocity profile. First, the symmetric (zero skew) 2D saddle is rotated around the central z-axis. Afterward, the resulting 3D saddle is made anisotropic by multiplying its component in x-direction by a factor of 2. The extent of the domain is chosen as $[-2.5, 2.5] \times [-1.25, 1.25] \times [-2.5, 2.5]\ \mathrm{m}^3$. From this we construct a time-dependent field by translating the region in terms of t along the Lissajous curve

$$\mathbf{x}(t) = \begin{pmatrix} \cos(5t)/4 \\ \sin(5t)/8 \\ \cos(\frac{5}{3}t)/4 \end{pmatrix}. \tag{16.1}$$

The time-dependent field is discretized on an equidistant grid at resolution $51 \times 26 \times 51$ within the time interval $[0, 10]$ s, at regular sampling of $\Delta t = 0.0125\,\mathrm{s}$. Figure 16.8 (b) sketches the field at $t = 3.5\,\mathrm{s}$ together with the Lissajous curve and FTLE ridges (red - forward, blue - reverse). In terms of traditional vector field topology, there is a stable 1D manifold in z-direction and an unstable 2D manifold in x- and y-direction.

In the following, we examine the case with $t_0 = 3.5$ s, FTLE advection time $T = 2$ s, and streak generation time $T_s = 2$ s. Figure 16.9 shows forward and reverse FTLE ridges for different t_0 in the time scope of the analysis, showing that the hyperbolic region first moves to the left, and then returns to the right. We start with an initial FTLE resolution of $81 \times 41 \times 81$ at refinement level 0 in the region $[-2, 2] \times [-1, 1] \times [-2, 2]$ m^3 , and use 4 iterations to refine the ridge intersection curve during Stage I of our algorithm. During refinement we set the FTLE filter to $\tau_\sigma = 1.0$ and restrict it to 1.3 on the final level. A offset of 0.01 m is chosen for streak seeding.

Performance and Cost Analysis

Table 16.1 shows timings and statistics for the four refinement levels. The extraction of the ridge intersection cells on refinement level 4 required 41.4 s in total, with 25 s being allotted to the extraction at refinement level 0, for which, in contrast, the computation steps E1 to E6 need to be performed for all cells of the initial grid (256000 active cells using 31.4 MB for all quantities). It is apparent that our technique strongly benefits from the reduction of the ridge intersection problem from $\mathcal{O}(N^4)$ to $\mathcal{O}(N^2)$ during refinement. The level 0 grid yields 48 potential ridge cells, resulting in 384 refined cells that are processed on the following level. It can also be seen that our algorithm accounts for growing ridges due to increased flow map resolution. Level 2, for example, starts with 856 input cells but detects 910 ridge surface cells in forward FTLE. The more restrictive FTLE filter threshold on the last level is also noticeable. The refinement levels exhibit relatively long execution times compared to the base level with respect to the number of active cells. This is caused by the overhead of the underlying sparse grid data structure. The very costly computations of the flow map, the path surfaces, and the streak manifolds are executed in parallel on the GPU. Currently, all other stages of our algorithm are implemented on the CPU. In particular the ridge extraction stage could strongly benefit from a parallel implementation, allowing for further speedup.

The geometry generation stages G1 to G4 are less time consuming, also due to GPU-accelerated streak generation. The single ridge intersection curve found in the data

Table 16.1 — Timings and problem sizes on different ridge intersection refinement levels for the Gyre-Saddle data set starting with a flow map resolution of $81 \times 41 \times 81$ on level 0 and ending with a virtual resolution of $1296 \times 656 \times 1296$ on level 4.

refinement level	0	1	2	3	4	total
active cells	256000	384	856	1832	5772	264844
2D ridge cells, bck	1481	342	751	1756	4764	-
2D ridge cells, fwd	908	385	910	2168	2365	-
intersec. ridge cells	48	107	229	659	374	-
t in s	25	1.1	2.0	3.8	9.6	41.4

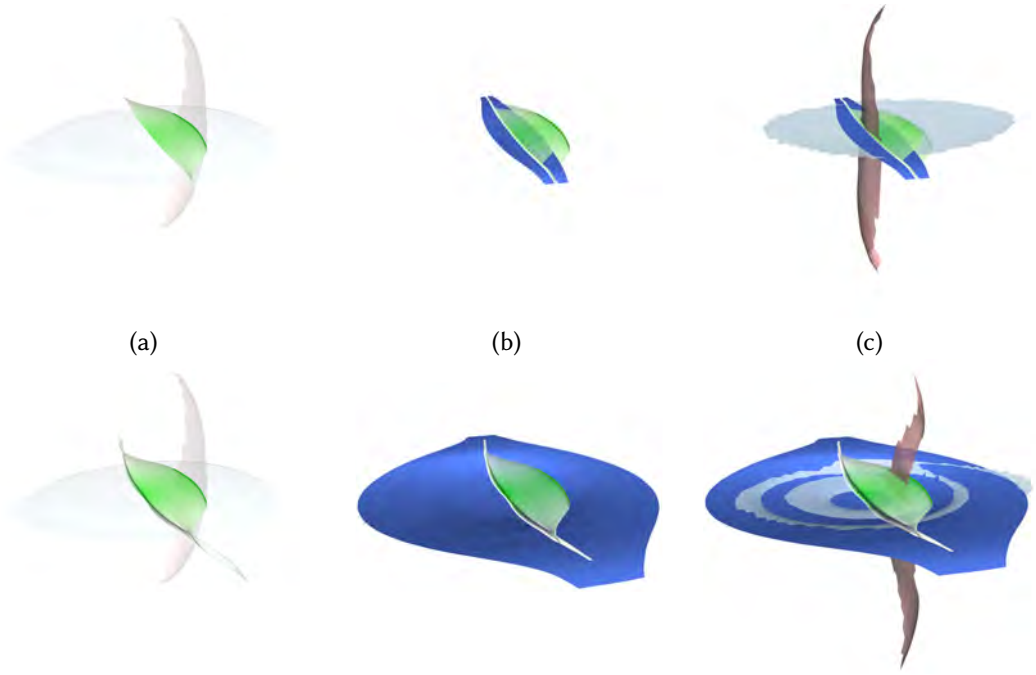


Figure 16.10 — Streak generation at the Gyre-Saddle example. Refined ridge intersection curves are advected for T_s = 0.99 s (first row), and T_s = 2.0 s (second row), producing hyperbolic path surfaces (green), (a) and (d). Simultaneously, streak surfaces (dark blue) are generated at the front of the growing path surface, (b) and (e). Streak manifolds provide a better LCS representation than the attracting FTLE ridge (light blue), (c) and (f) (t_0 = 3.5 s).

set consisted of 651 vertices. Its advection for time T_s with a time step of Δt = 0.05 s required 0.4 s, and 11.1 s with streak manifold generation enabled, resulting in a total time of 52.5 s.

The final refinement level corresponds to an FTLE resolution of 1296 × 656 × 1296 with approximately 10^9 active input cells. In contrast to a total of only 264844 active cells for the adaptive technique, the extraction on such a grid would take several orders of magnitude longer. Moreover, the performance advantage grows further when it comes to the visualization of the time-dependent behavior of LCS, i.e., the computation of traditional time series of FTLE ridges compared to our streak generation (Section 16.2.3).

Streak Manifolds vs. FTLE Ridges

Figure 16.10 shows different aspects of our technique. While the first row shows an intermediate state after T_s = 0.99 s, the images in the second row show the final state after T_s = 2.0 s. The first column depicts the evolution of the path surface (hyperbolicity mapped to saturation) seeded at our refined ridge intersection curve in context of the

(a) (b) (c)

Figure 16.11 — Superior accuracy of streak manifold (dark blue, cut in the middle) in comparison to reverse time FTLE ridge (light blue). The ridges show aliasing artifacts. With increasing FTLE resolution, $81 \times 41 \times 81$ (a), $161 \times 81 \times 161$ (b), $321 \times 161 \times 321$ (c), the ridges converge toward the streak manifold (Gyre-Saddle data set, $t_0 = 3.5\,\mathrm{s}$, $T_s = 2\,\mathrm{s}$).

FTLE ridges at $t_0 = 3.5\,\mathrm{s}$. The second row shows the streak manifold generation process during path surface integration, and the last row compares the streak manifold with the corresponding attracting FTLE ridge (at $t_0 = 4.49\,\mathrm{s}$ and $t_0 = 5.5\,\mathrm{s}$, respectively). The LCS advection principle [175] is reasonably well fulfilled in the first half of the time interval: the front of the hyperbolic path surface matches the FTLE ridge intersection. Although it starts to deviate in the second half of the time interval, the resulting streak manifold still fits the actual LCS (as detailed in section 5.2 of [161]), resulting in a highly accurate visualization compared to traditional visualization by FTLE ridges, which are inherently limited by the FTLE sampling resolution. As demonstrated in Figure 16.11, the ridges converge toward the streak manifold with increasing FTLE resolution.

16.3.2 Von Kármán Vortex Street

The time-dependent CFD flow field of the von Kármán vortex street phenomenon was computed on a domain with extent $[0, 10] \times [0, 60] \times [0, 10]\,\mathrm{m}^3$ and a time interval of $[0, 1.1]\,\mathrm{s}$. The domain was discretized with an equidistant grid of resolution $41 \times 241 \times 41$ and a temporal resolution of $0.001\,\mathrm{s}$. A cuboid obstacle was placed near the inflow region (Figure 16.12), with inflow velocity in direction of the largest extent and vertical linear profile from $200\,\mathrm{m\,s^{-1}}$ at its basis to $250\,\mathrm{m\,s^{-1}}$ at its top.

An FTLE advection time of $T = 0.1\,\mathrm{s}$ was used for the ridge intersection at $t_0 = 0.4\,\mathrm{s}$ shown in Figure 16.12 (a). Here, aliasing artifacts are obvious, a common problem with FTLE ridge extraction in regions of strong separation [156]. They are most prominent in the upstream part at reverse FTLE ridges (red). At the downstream end, forward FTLE did not produce sufficiently sharp ridges. We excluded both regions during ridge intersection extraction. This left a valid ridge intersection region of roughly 20 m length—located at the center of the domain. By enforcing a threshold of $\tau_H = -30$ we made sure to extract sufficiently sharp ridges that represent LCS. Starting with an FTLE grid resolution of two times the flow field resolution, two refinement iterations were performed, requiring 98 s of computation time. Finally, short intersection curves were discarded, leaving four ridge intersection curves generating streak manifolds that capture two periods of the flow's prominent vortex structure (Figure 16.12 (b)).

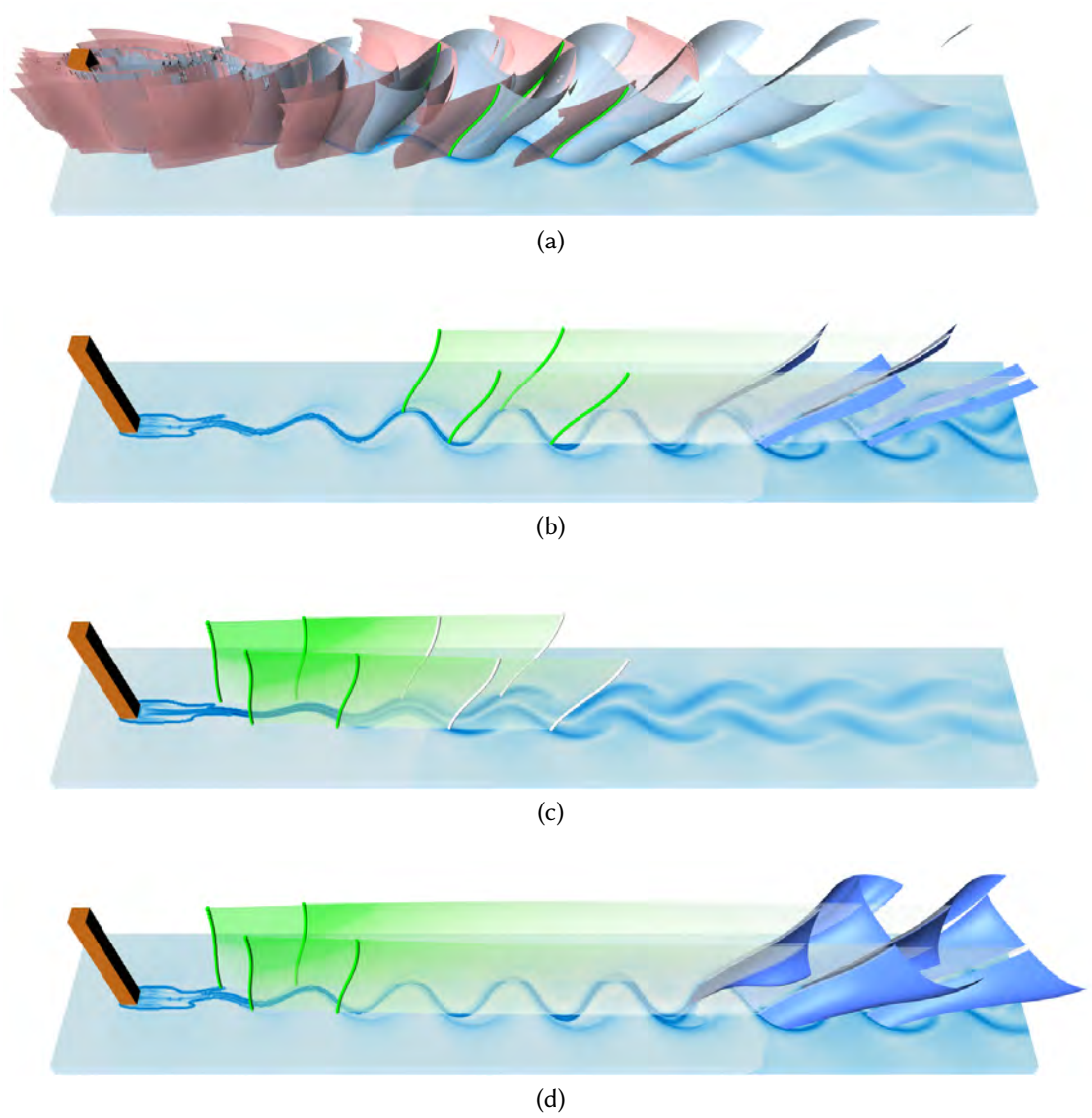


Figure 16.12 — Streak manifold generation at the von Kármán vortex street example. (a) Attracting (blue) and repelling FTLE ridge surfaces (red) at $t_0 = 0.4$ s are extracted and intersected, resulting in four ridge intersection curves (green). (b) The hyperbolic curves are advected along hyperbolic path surfaces (green) for $T_s = 0.1$ s, and during advection streak manifolds (dark blue) are seeded at the front. The matching reverse FTLE at $t = 0.5$ s is depicted on the bottom plane. (c) Reverse preadvection of the intersection curves for $T_p = 0.07$ s yields larger streak manifolds (d).

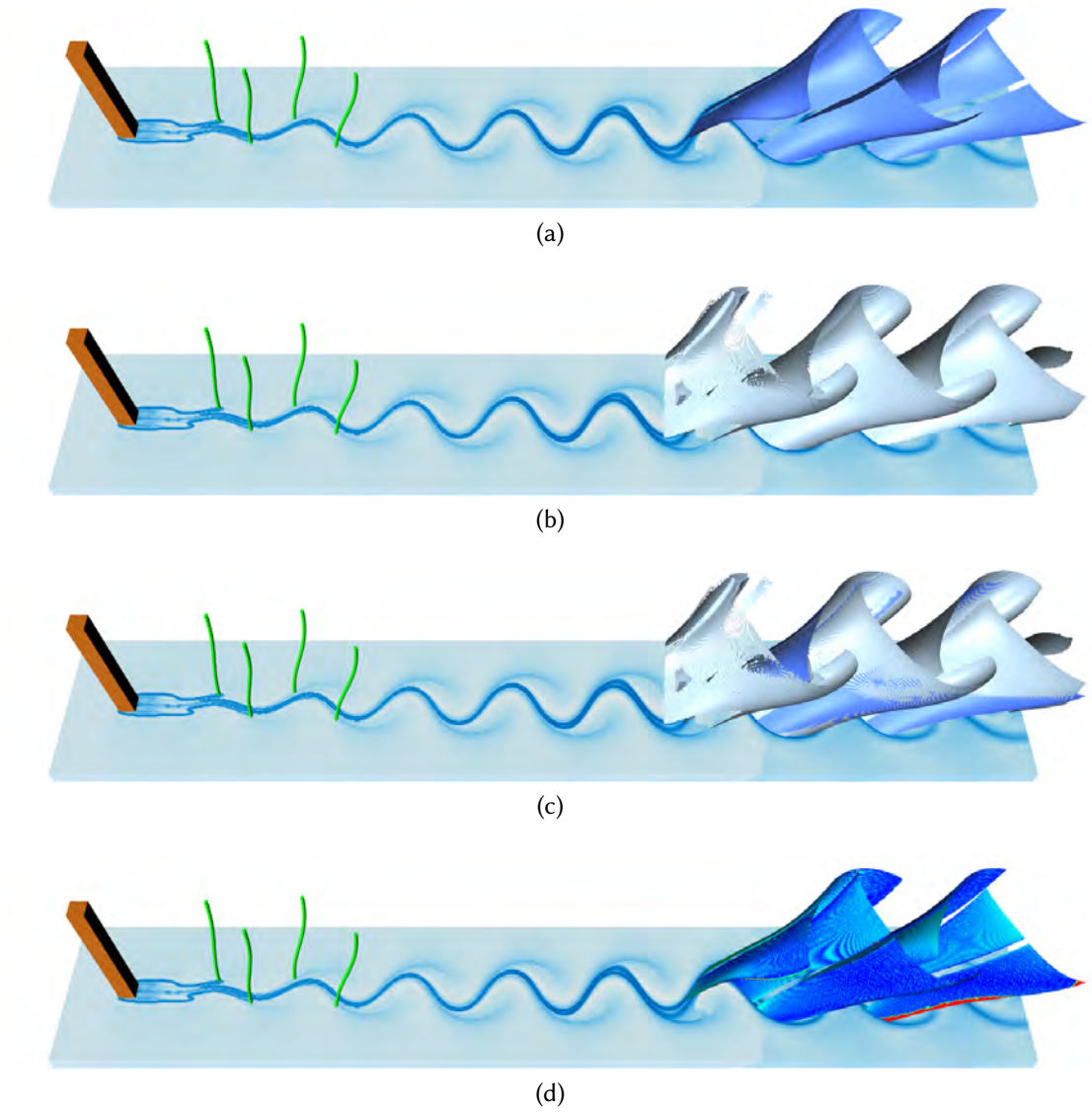


Figure 16.13 — Comparison of streak LCS with LCS obtained from FTLE ridges at the von Kármán vortex street example. (a) The streak manifolds from Figure 16.12 (d) are very smooth and match to the 2D FTLE ridges on the bottom plate. (b) Ridges extracted from a very high resolution FTLE field at $t_0 = 0.5\,\mathrm{s}$ show some aliasing artifacts and (d) match to the streak manifolds. (d) The geometrical distance to the FTLE ridges is mapped to color on the streak manifolds (zero distance, dark blue). Only regions with no corresponding ridge surface part exhibit distance larger than one FTLE cell (red). Overall, the ridges exhibit aliasing, whereas the streak manifolds are of superior quality.

As a first test, streak surfaces were generated for $T_s = 0.1\,\text{s}$ with a seeding offset of $0.15\,\text{m}$, resulting in comparatively small streak manifolds (Figure 16.12 (b)). Applying reverse preadvection (Section 16.1.2) prior to streak generation proves particularly useful. A preadvection of the intersection curves by $T_p = 0.07\,\text{s}$ (Figure 16.12 (c)) enables a total streak generation time of $0.17\,\text{s}$, allowing the streak surfaces to grow larger with the same seeding offset (Figure 16.12 (d)).

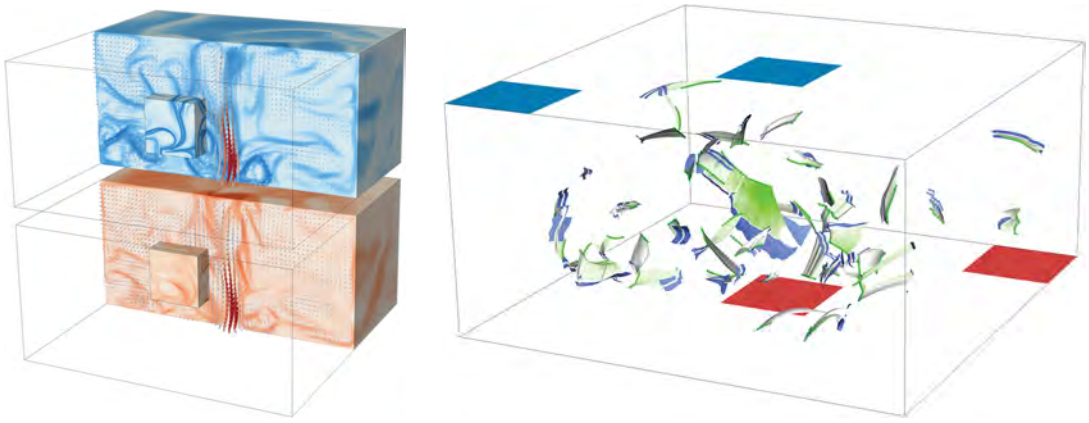
The streak manifolds are compared with the reverse FTLE ridges extracted at $t_0 + T_s = 0.5\,\text{s}$. The aforementioned wide range in separation strength quickly leads to aliasing of the FTLE ridges, making it difficult to choose the FTLE advection time T properly to obtain appropriate FTLE ridges for comparison. Here, at the downstream end, an advection time of $-0.2\,\text{s}$ yields sufficiently sharp reverse FTLE ridges (Figure 16.13 (b)). The ridges were extracted from an FTLE field of four times the flow field resolution. After suppressing insufficiently sharp ridges for comparison purposes with a filter $\tau_H = -20$, aliasing is apparent, whereas our streak manifolds are very smooth. Figure 16.13 (c) shows both ridges and streak manifolds. A quantitative analysis of the deviation between streak manifolds and ridges is shown in Figure 16.13 (d): the shortest distance to the ridge surfaces is color-coded on the streak surfaces. The regions of the streak surface that have an FTLE ridge surface counterpart feature a distance much smaller (blue color) than the size of a cell of the FTLE grid used for extracting the ridge surfaces (red color). This is consistent with the finding in Section 15.2 that the accuracy of ridge extraction is limited to the order of the cell size. The color-coding also reflects the aliasing artifacts of the ridges.

16.3.3 Buoyant Flow

This data set was computed with a CFD simulation of buoyant airflow inside a closed container. The complex flow behavior and LCS structure allows for investigating the properties of our approach in cases with many streak manifolds. The influence of hyperbolic strength and the validity of the LCS advection property are discussed.

Data Set Description and Overview

The data set has extent $[0, 10] \times [0, 5] \times [0, 10]\,\text{m}^3$ on an equidistant grid of resolution $61 \times 31 \times 61$ for the time interval $t = [0, 100]\,\text{s}$. Time discretization is also uniform with $\Delta t = 1\,\text{s}$. No-slip adiabatic boundary conditions are applied to all walls. There is a heated plate at the center of the floor and a cooled one at the center of the ceiling. As depicted in Figure 16.14 (b), another heated plate was added in one of the floor's corners and a cooled one in the diagonally opposite corner of the ceiling, to increase time-dependent complexity. Figure 16.14 (a) shows the domain with forward and reverse FTLE visualization ($t_0 = 50.889\,\text{s}$, $T = 20\,\text{s}$, $61 \times 31 \times 61$ FTLE resolution), enhanced with a hedgehog plot overlay at t_0. In the middle of the domain, a strong downward flow is apparent.



(a) (b)

Figure 16.14 — Buoyant flow overview with the two heated plates (red) and cooled plates (blue) in (b). (a) Reverse FTLE (top) and forward FTLE (bottom) with the ROI at higher resolution, and the vector field sketched on a plane. (b) Streak manifold topology with hyperbolicity color mapping on the generating path surfaces ($t_0 = 50.02$ s, $T = 20$ s, $T_s = 10$ s, seeding offset 0.025 m).

Figure 16.14 (b) shows an overview of the streak manifold topology at time $t_0 = 50.02$ s, using $T_s = 10$ s. A ridge sharpness threshold of $\tau_H = -3$ was imposed, and all intersection curves shorter than 0.34 m were discarded. Due to the complex LCS structure, we restrict the following investigations to the region of interest from Figure 16.14 (a), with its strong hyperbolicity yielding the largest streak manifold. It is located near the center, with an extent of $[2.5, 5] \times [2, 4.5] \times [4, 5.5]$ m^3, and a 6 times finer FTLE resolution of $161 \times 161 \times 121$. Note, half of the ROI is hidden in the overview image. In the following, the FTLE integration time is set to $T = 20$ s and $t_0 = 50.02$ s. For ridge intersection at final level 3, an FTLE threshold of $\tau_\sigma = 0.15$ was enforced (to obtain the sharpest ridges) for the reverse FTLE, and 0.16 in the forward one. During refinement a less strict $\tau_\sigma = 0.13$ was employed in both fields. We set the seeding offset to 0.02 m and discarded intersection curves shorter than 0.41 m.

Hyperbolicity and the LCS Advection Principle

Figure 16.15 (b) shows an overview of the ROI with the most prominent ridge intersections, where $T_s = 10$ s. The FTLE at time t_0 as well as at $t_0 + T_s$ is depicted by cross sections. We observe that two nearby reverse FTLE ridges at t_0 "merged" after streak advection time T_s. At this later instance, the streak manifold is consistent with the attracting FTLE ridge depicted on the section, emphasizing the validity of the advection principle for the attracting LCS. The merging of the less sharp ridge intersections with the sharpest in the center during streak advection is an example for so-called foliation, i.e., mixing processes captured by LCS.

We additionally investigated the advection principle for the hyperbolic intersection

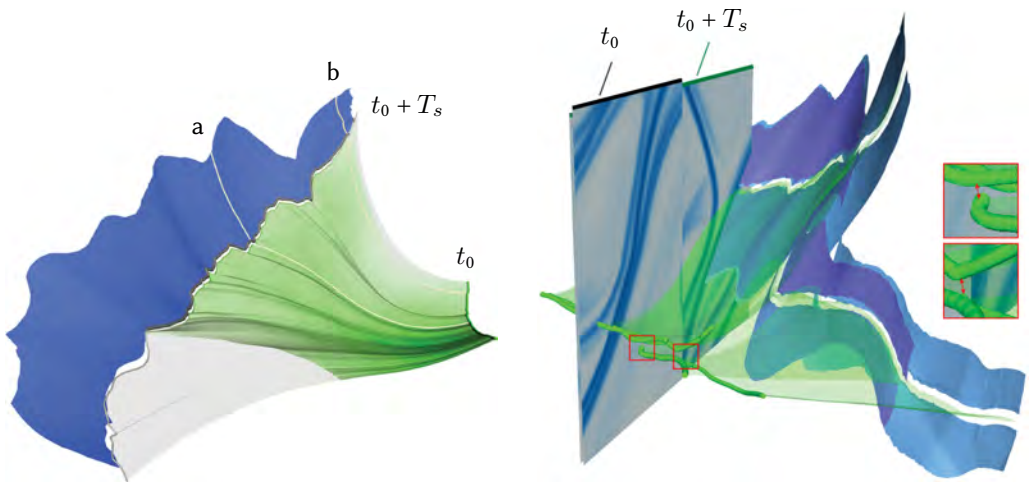


Figure 16.15 — (a) Hyperbolic path surface (green) from a ridge intersection curve of the buoyant flow ROI. Streak (blue) generation is stopped as non-hyperbolic field regions are entered (gray parts). Streak lines a and b reveal the effect of hyperbolicity strength. (b) Streak manifolds of four intersection curves within the same ROI with corresponding attracting FTLE ridges on the large cutting planes (t_0 = 50.02 s, T_s = 10 s).

curve. We visualized multiple hyperbolic path surfaces with T_s = 7.047 s emanating from different refinement levels of the ridge intersection curves. Figure 16.16 shows the ROI with the camera at the bottom pointing upwards, with respect to the camera setting of Figure 16.15 (b). Again, even at refinement level 0, the advection principle is well satisfied for the attracting LCS, as can be seen at the intersection with the cross section. However, there is a larger discrepancy to the repelling LCS. However, with increasing ridge intersection refinement this discrepancy decreases. Figure 16.16 (b) and (c) additionally visualize hyperbolicity strength mapped to saturation on the hyperbolic path surfaces of level 0 and level 2, respectively. An interesting observation can be made by inspecting hyperbolicity along the path surface: close to the observer it is relatively small, leading to a smaller discrepancy at $t_0 + T_s$, whereas near the cross section, the (hyperbolic) effect of the repelling ridge is quite strong, amplifying small errors made in the ridge intersection stage at t_0. This highlights the importance of our adaptive ridge intersection approach. However, please note, that deviation from the repelling LCS (or attracting one in the case of reverse streak manifold generation) has usually negligible impact on the accuracy of the resulting streak manifold as long as the streak gets seeded on the correct side of the repelling LCS.

Another important aspect of our streak manifold visualization is that the influence of hyperbolicity strength is directly visible in streak growth (Figure 16.15 (a)), enabling a comparative visualization of hyperbolic LCS regions in the ROI (Figure 16.15 (b)) and the whole domain (Figure 16.14 (b)).

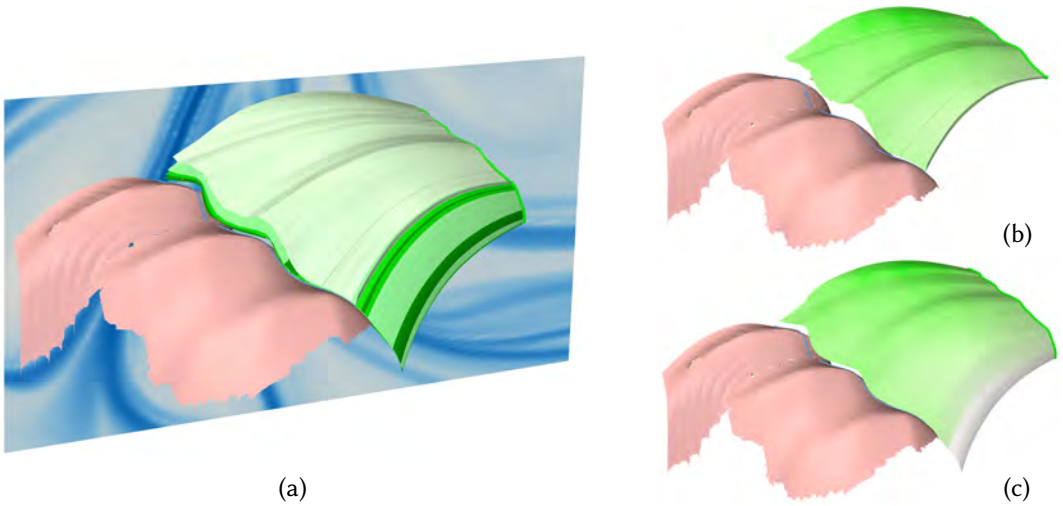


Figure 16.16 — LCS advection principle. Hyperbolic path surfaces of length T_s = 7.047 s generated by ridge intersection curves at t_0 = 50.02 s obtained at four different refinement levels: Level 0 (light green) to level 3 (dark green). With increasing refinement level of ridge intersection extraction the front of the respective path surface converges toward the ridge intersection at $t_0 + T_s$, depicted as blue curve on the forward FTLE ridge (red) (a). Hyperbolicity is mapped to color for refinement level 0 (b) and level 2 (c), with strongly hyperbolic (green) to non-hyperbolic (white).

16.4 Conclusion and Outlook

We presented the extension of the streak-based time-dependent 2D vector field topology [161] to 3D vector fields. We have showed that in 3D vector fields the counterparts of hyperbolic trajectories are distinguished hyperbolic path surfaces rather than hyperbolic trajectories. These surfaces can be obtained by advecting the intersection curves of attracting and repelling LCS, represented as ridge surfaces in the forward and reverse FTLE fields. We presented an adaptive extraction technique for these intersection curves to account for the complex FTLE ridge structure appearing on a wide range of length scales. The extraction algorithm provides high accuracy and efficiency in terms of computation time and memory consumption. Our seeding scheme allows us to generate streak manifolds from the hyperbolic path surfaces. This approach not only gives insight in the space-time structure of time-dependent LCS, but also provides high-quality visualization of LCS compared to the traditional technique by ridge surface extraction from FTLE fields. We evaluated our approach with CFD data and showed that the growth of our streak manifolds additionally visualizes the action of hyperbolic regions and the LCS they cause. As our concept builds upon the integration of streak surfaces, it allows for large time savings when visualizing LCS time series, in comparison to the computationally expensive extraction of LCS by FTLE ridges. We discussed factors that might limit the time interval admissible for inspection by our

approach. To extend this interval "reverse preadvection" has been introduced, allowing for longer streak generation phases and hence more significant streak manifolds.

While our streak topology concept provides a direct generalization of the traditional vector field topology it has to be investigated further, as there are still a few open questions. Our topology, for example, only provides a time-dependent generalization of critical points of type saddle. Time-dependent variants of nodes, foci, or centers are not integrated, yet. Hence, LCS that are caused by non-hyperbolic mechanisms, like those induced by shear, cannot be captured. Although periodic orbits were not directly addressed in this paper, the separatrices of saddle-type periodic orbits that coincide with intersection curves of forward and reverse FTLE ridges are implicitly captured by our approach. However, a detailed analysis is topic of future work. Besides, time-periodic flows are rather rarely encountered in practical fluid dynamics. Finally, several parameters have to be set manually by the user. Our approach would benefit from an automatic choice of appropriate advection times, like the FTLE advection time T, to ensure the LCS property of the ridges, and the streak advection time T_s. In particular, it is necessary to choose different times for hyperbolic regions appearing on different time and length scales. Interactive analysis techniques (Section 13.3) as well as the extension of our LCS validation technique (Section 15.2) to 3D fields could be helpful in this context.

Chapter 17

Concluding Remarks

This work shows the importance of advanced visualization techniques for the analysis of data from computational fluid dynamics (CFD). Novel methods for the direct and accurate visualization of higher-order flow simulations, as well as time-dependent vector field topology concepts, including interactive analysis techniques have been presented. While the focus clearly lies on CFD data, the presented methods can also be applied to scalar and vector fields from other sources. The interdisciplinary aspect of computational science has been emphasized throughout this thesis, as the research was conducted within the SimTech research cluster.[1] Background information on the simulations and the underlying physics were given to identify the main challenges involved in computing and analyzing complex CFD data. While many fluids are described accurately by the Navier-Stokes equations using a compact continuum model, the actual solutions to the involved set of differential equations can exhibit very complex characteristics, which are not fully understood yet. As discussed, this complexity is mainly due to the nonlinear behavior of fluid flow, resulting in phenomena on multiple time and length scales. Advanced visualization techniques can support the analysis and help in gaining a deeper understanding of fluids. In the following, three common questions arising from the fluid dynamics challenge are formulated (Q1—Q3). The contributions of this thesis to the field of visualization and scientific computing, as well as the experiences gained, are discussed in their context. This is followed by a summary and outlook.

Q1: How do simulations cope with the complexity of fluids?

While direct numerical simulation (DNS) resolves all time and length scales down to the Kolmogorov microscales of turbulence, larger problems can often only be handled if appropriate turbulence models are applied. By using such models, coarser grids become possible, reducing the overall computational cost. Adaptivity is another successful

[1] The project title within the cluster of excellence simulation technology (EXC 310 SimTech) was "Interactive visualization of multi-scale, multi-physics simulations".

approach for handling nonlinearity, be it in the choice of the model, or its discretization. All this introduces additional complexity into the simulation methods and their data structures (compared to DNS solvers working with regular grids). This development goes along with an overall increase in the problem sizes that can be handled, as parallel compute environments with higher performance become available.

Consequences for interactive visualization of the resulting data (Part II) have been exemplified by higher-order discontinuous Galerkin (DG) simulations and generalized FEM approaches, like the particle partition of unity method (PPUM). It has been demonstrated that, analogous to the simulation side, new visualization techniques are required to be able to handle nonlinear fields directly and accurately, as traditional techniques are mostly limited to linear approximations. First, a pixel-accurate direct visualization technique for 2D fields from PPUM simulations, which use algebraic refinement in addition to hp-adaptivity, has been presented (Chapter 7). Second, an interactive higher-order volume rendering system for hp-adaptive DG data has been developed (Chapter 8). While PPUM uses an adaptive particle-based domain discretization approach, the DG simulations use adaptive grids, including curved elements. The adaptivity in the complex grid as well as in the polynomial degree of the field solution is directly accounted for in the adaptive sampling approach of the volume ray casting technique, enabling the computation of high quality visualizations at interactive display rates. Moreover, it was shown that acceleration with GPUs is a basic prerequisite when visualizing higher-order fields. While the traditional shader pipeline proved to be a perfect match for direct 2D field visualization, the higher flexibility of GPGPU programming languages was utilized for the volume renderer. Direct visualization not only provides high-quality visualizations but also alleviates visual debugging of complex adaptive simulations, as demonstrated in this thesis. Being able to handle higher-order simulation data directly might also become necessary in the case of in-situ analysis of large-scale simulations (see Q3 for a discussion on the tighter integration of simulation and visualization environments). Further, it has been investigated how simulation models, like the law of the wall, can be harnessed to improve interpolation schemes for more physically-accurate and consistent visualization (Chapter 9).

Q2: How does abstraction support the analysis of complex CFD data?

This thesis has emphasized the importance of particle tracing as a basic building block for advanced vector field visualization techniques. At the example of nonlinear solutions of the Navier-Stokes equations, it has been demonstrated that abstraction not only plays an important role on the simulation side (using various models on different scales), but also in the analysis of the resulting data. Feature-based approaches (Part III and IV) were discussed as one means to obtain abstract representations of the raw data, supporting a more focused investigation of the relevant structures and their interrelation on different time and length scales.

In a first step, a technique has been presented that employs a surface-guided seeding

strategy in order to highlight the important structures in three-dimensional vector fields with integral curves (Chapter 11). An interactive exploration is enabled by leaving the choice of appropriate feature criteria and seeding surfaces to the user. Exploring large time-dependent vector fields with such an approach, however, can be tedious, as the time dimension has to be explored separately in addition to the spatial domain. While traditional feature tracking methods can help in visualizing the time evolution of features, alternative concepts based on dynamical systems theory have been employed in this thesis in order to extract global field characteristics. As one example, finite-time Lyapunov exponent (FTLE) fields were used. Their computation is based on techniques from predictability analysis, extracting information from a densely seeded set of path lines. To improve the quality of FTLE fields nonlinear approaches to FTLE computation have been presented (Chapter 13). These techniques help to reduce the linearization error of traditional algorithms, accounting for the error sensitivity of ridge extraction algorithms. Ridge extraction in FTLE fields yields global space-time structures that exhibit an attracting or repelling behavior with respect to neighboring particles. As discussed, this can help to reveal the global structure or topology of the flow. Vector field topology can provide an abstract compact representation of the field, with separatrices partitioning the domain into regions of coherent behavior. The extension of the classical steady vector field topology to time-dependent fields, the topic of Part IV, however, is not straightforward. Lagrangian coherent structures (LCS), the time-dependent counterpart to separatrices, are an important building block. While LCS are traditionally obtained as FTLE ridges, not all FTLE ridges represent LCS. In Chapter 15, a method has been presented that is able to validate the LCS property of one-dimensional ridges in 2D FTLE fields. In this context, the intersections of repelling and attracting LCS were identified as distinct points that can be tracked over time. The hyperbolic intersections among these have additionally been found to be of great importance in the generalization of the traditional steady vector field topology to time-dependent fields. In traditional vector field topology separatrices are obtained by seeding streamlines near critical points. In time-dependent flow the hyperbolic intersections take the role of critical points. Moving along hyperbolic trajectories—a property that was harnessed by the aforementioned tracking technique—the intersections are used for seeding generalized streak manifolds, resulting in time-dependent separatrices. This again highlights the importance of integral manifolds. From a small subset of the domain (representing the hyperbolic intersections) a topological segmentation can be derived. As a contribution of this thesis, the time-dependent streak topology concept for 2D fields [161] has been extended to 3D domains, and its usefulness has been demonstrated (Chapter 16). The streak manifolds offer an LCS representation of higher quality than traditional FTLE ridges, and animations of streak manifold growth, which can be obtained at low computational cost, give additional insight into the constituent dynamics of the flow.

Figure 16.12 demonstrates the topological aspect of the streak manifolds at the von Kármán vortex street. The streak LCS separate the different vortices that emerge and

detach from the boundary layer of the obstacle—as expected from such flow configurations (cf. Figure 4.4 (c)). The trajectories of the moving LCS intersections, represented by hyperbolic path surfaces, additionally help to understand global field dynamics. To give more detailed insight into the vortex structure, geometrical representation of vortices could be extracted, e.g., from the λ_2 field, and visualized together with the topology. To further enrich the visualization, the surface-guided seeding technique (Chapter 11) could be used to visualize a selected set of path lines starting on the λ_2 isosurfaces, similar to Figure 11.4. This simple example demonstrates the need for different levels of abstraction in the analysis of time-dependent CFD data. While a static visualization of an instantaneous field snapshot, e.g., with LIC on a cutting plane, would reveal the large-scale vortices, no reliable information would be conveyed about their emergence or future development. Visualizing a dense field of path lines, on the other hand, would lead to serious visual clutter, making it difficult to detect the important structures at all. The shortcomings of such simple integral manifold visualizations further manifest if flow phenomena are investigated that take place on different length and time scales. The presented feature-based approaches can facilitate such an analysis. Using different time scopes in FTLE analysis, for example, allows to separate short-lived from long-lived structures.

Q3: What role does interactive exploration play?

A successful analysis of field data from CFD often requires a combination of different visualization techniques, as sketched at the end of the previous section. However, there is typically no single set of techniques and corresponding parameters that automatically leads to optimal results. Hence, an interactive explorative approach is often required, supported by automatic tools that help to shed light on the complex nonlinear nature of fluid phenomena.[2]

All visualization techniques presented in this thesis are designed as interactive applications or are suitable for integration into an interactive environment. They are all based on adapted implementations of the abstract visualization pipeline (Figure 2.2), providing different means of interaction. GPU acceleration has proven to be one enabling factor for an efficient visualization of complex data. In the case of the higher-order visualization techniques (Part II), for example, an instant feedback is achieved when changing the rendering or mapping parameters, like the camera or transfer function. For three-dimensional higher-order fields it has also been demonstrated that using a compute cluster equipped with GPUs can help to obtain interactive display rates. While the higher-order techniques focused on the development of a single visualization technique, like DVR, the surface-guided seeding system (Chapter 11) combines multiple techniques to enable an efficient interactive exploration of 3D vector fields. The user is able to interact with the filtering stage to restrict the investigation to feature

[2] Gleick compared the analysis of the Navier-Stokes equations with a walk through a maze that continuously rearranges its walls (see Section 4.4 and [70]).

surfaces. The system demonstrates how traditional techniques can be complemented with automatic feature-based techniques to guide the exploration to regions of interest within the raw data. Giving the user the freedom in the choice of the feature criteria, correlations between multiple fields, for example, between the vector field and an associated scalar field, can be revealed in an interactive exploratory manner. A flexible visualization environment allows the experienced user to choose pipeline algorithms and parameters freely, but also provides tools that help to narrow the choice in an intelligent manner. To this end, visual analytics (VA) provides promising concepts, complementing interactive exploration and traditional feature extraction algorithms with machine learning techniques. While VA has become popular in the context of information visualization, its interdisciplinary approach can also be useful in scientific visualization, helping to discover unexpected correlations and features. As an example in the flow context, an interactive tool for exploring the FTLE parameter space has been developed (Chapter 13). It uses multiple views and automatic clustering algorithms to refine the investigation in order to detect hidden correlations.

Experimentation is of great importance in research—it is a substantial ingredient to enable scientific progress. As scientists strive to understand physical systems and their behavior, they pose *what-if* questions. While numerical simulation (Chapter 4) can help to answer these questions, an intuitive experimentation, i.e., an efficient exploration of the simulation parameter space, is often hindered due to long simulation run times. In an ideal computational science environment the change of a parameter would result in immediate feedback, enabling a more holistic interactive experimentation workflow. Scientists would like to be able to steer their simulations in real time and simultaneously interact with the resulting data [128]. Using three-dimensional fluid simulation as an example, a change in the boundary conditions would be instantly reflected in the structure of the flow, which, for example, could be visualized and analyzed with the streak-based vector field topology (Chapter 16.1). Yet, due to the inherent complexity of fluid flow (Q1), interactive experiments are only possible for extremely simplified scenarios. For those, however, the effectiveness of such environments has been demonstrated [4, 202]. Besides model exploration, computational steering can also enable simulation monitoring, experimentation with different solvers, visual debugging and performance optimization [128]. Computational steering and visual analytics of large-scale simulations, however, still faces many challenges [83, 217]. The architectural complexity of distributed high performance computing environments (HPC) is the most obvious factor that hinders the integration of interactive workflows into large-scale simulations.

In-situ analysis and visualization, i.e., the direct processing of simulation data during simulation, is one important cornerstone of future analysis environments. The in-situ approach can help to cope with the huge amount of generated data by employing a tight coupling model, enabling efficient in-memory communication between simulation and visualization algorithms. Moreover, as data is preprocessed and analyzed automatically, costly interruptions due to user interaction can be minimized. While this topic is

not covered in the main part of the thesis, some research has been conducted in this direction. The results are shortly summarized in the following. A tightly coupled in-situ visualization of 3D flow has been implemented on a single GPU in the context of a student thesis [187, 188]. The simulation computed a simple two-phase flow in porous media based on the Buckley-Leverett equation, and the VTK-Toolkit was used for concurrent visualization. The sparse system of nonlinear equations of the problem was solved with an iterative conjugate gradient algorithm on the GPU [25]. Compared to the multi-threaded CPU implementation provided by the SimTech research partners[3], the GPU simulation achieves a speed-up of larger than 4 at problem sizes with more than 10^5 degrees of freedom—corresponding to a data size of a few MB per time step. As the computation of a single time step already takes up to one minute on the GPU, tight coupling only provides insignificant advantages for such small-scale simulations. The generated amount of data is simply too small to pose a real challenge with respect to its processing, i.e., its transfer to host memory, storage, or interactive visualization. Moreover, from the visualization point of view the resulting flow solutions are not very complex, and hence do not require an in-depth analysis. In order to demonstrate the usefulness of in-situ processing with grid-based flow simulations, one either has to reduce the computational complexity, e.g., by solving 2D problems, or harness cluster-based HPC environments, which substantially increases the complexity from the software engineering point of view. As an example of the former case, a GPU solver for the Navier-Stokes equations on relatively coarse two-dimensional grids has recently been implemented in our research group. Using this real-time solver demonstrated how in-situ visualization and simulation steering can facilitate an interactive FTLE-based LCS exploration [4]. The design of distributed in-situ environments that are able to handle more generic real world problems requires a significantly larger effort. Moreover, to be of real use to simulation research, such a system should be based upon existing simulation and visualization frameworks. Within SimTech, several projects use the *Distributed and Unified Numerics Environment* (DUNE), a modular framework for solving partial differential equations with grid-based methods [14, 13]. In close cooperation with the SimTech partners responsible for DUNE we developed a DUNE visualization interface. The interface eases the tight coupling of simulation codes (the back end) with visualization environments (the front end). Both parts are coupled tightly in the sense that they run within the same process. The interface itself provides methods for querying the simulation state, accessing simulation data, and steering the simulation. To this end, a callback mechanism is used for synchronization, i.e., breakpoints can be integrated into the simulation code at which data can be exchanged with the visualization part. While the support for distributed computation on HPC clusters is inherently built into the DUNE framework, e.g., as parallel grids and solvers, a generic parallelization framework is not available on the visualization side. This is mostly due to the variety of available parallelization strategies, with the choice

[3] The CPU solver is implemented in DuMu^x, a simulation framework for porous media flow [53], which is based on the software infrastructure DUNE [13, 14].

often depending on the specific visualization technique. Distributed DVR and the distributed computation of integral manifolds, for example, have largely different requirements (Section 8.4 and Section 12.3). In many cases the simulation data cannot be handled locally on the compute node, but has to be redistributed for efficient visualization. Hence, to be able to cope with the complexity, having reliable software infrastructure on the visualization side is as important as on the simulation side. For in-situ visualization with DUNE we used the modular MegaMol visualization framework, which is developed in our research group [74]. While the GPU-based techniques presented in this thesis can be integrated easily into this framework, the framework provides limited support for distributed rendering on compute clusters, restricting the system to rather small problem sizes. Open-source visualization frameworks, like ParaView, that are designed for distributed execution, are better suited for large problems. While support for in-situ processing was added to ParaView recently [46], the design of efficient interactive applications is hindered as the underlying VTK pipeline is largely CPU-based. Code running on GPU cluster environments nowadays typically still has to be designed carefully and optimized for specific hardware and their complex memory hierarchies. However, there is an ongoing research for abstraction layers, as discussed in Section 2.3. A "middleware" can simplify the programming of the distributed architecture and increase the maintainability of the resulting software. The use of many-core architectures on the simulation side is also expected to grow. The integration of GPU-based solvers into the modular DUNE framework, for example, is already planned. This might additionally drive the development of respective software frameworks for modern compute platforms, which are required to enable interactive large-scale experiments.

Summary and Outlook

This thesis has proposed several advanced visualization techniques for the analysis of field data from computational fluid dynamics simulation. Three important cornerstones of a successful CFD analysis have been identified. First, as a basic prerequisite, the visualization algorithms have to be able to handle the complex data generated by modern CFD solvers accurately, i.e., the error introduced by visualization has to be minimized (Q1). Secondly, visual representations have to be provided that allow for an analysis on different levels of abstraction (Q2). Finally, means for effective interaction are required to increase the overall efficiency of numerical experiments. Future environments could follow a more holistic approach than today, providing easy to use interfaces for interaction with integrated simulation and visualization pipelines, as well as automatic analysis algorithms (Q3).

To obtain more accurate visualization results the details of how the data was generated have to be taken into greater account than in the past. Working directly with the field representations computed by the simulation is a first step, as potential sources of error, e.g., introduced by resampling, can be eliminated. As the use of higher-order

simulation methods is becoming more widespread in the future, the demand for accurate higher-order visualization techniques will also increase, e.g., further methods for vector fields of higher-order, and large time-dependent fields have to be developed. The discontinuities in the DG solution pose a particular challenge for non-local vector field visualization techniques that involve integral manifold integration. Solutions to this problem have to be found, as many feature-based techniques are based on such manifolds, and the popularity of DG simulation methods is expected to increase due to its efficient parallel computation. Besides visualizing polynomial field data, new methods have to be developed for complex data from generalized FEM methods, model reduction techniques, or field data with uncertainty information. To account for this variety, CFD data sets should be annotated with meta-information describing the mathematical models and numerical techniques used for their computation. This would ease the choice of appropriate interpolation models during visualization, as model information can be harnessed like in our law of the wall interpolation approach. Information from the underlying PDE model could also be used for a posteriori error visualization, or to highlight regions that exhibit a particular physical behavior, e.g., by evaluating and visualizing different terms of the equations. With the increase in available compute power the solution of PDE problems during visualization—beyond simple particle advection—will become more common. Only recently the effectiveness of a posteriori visualization of advection-diffusion processes in CFD data has been demonstrated [86]. Overall, there is a trend toward a tighter integration of visualization and simulation techniques. As simulation techniques become amenable for an integration into visualization environments, the potential of interactive experimentation can be unfolded. On the other hand, it is also promising to transfer knowledge from visualization research to simulation. While most CFD simulations are still far from real time execution, automatic feature-based techniques could be integrated into CFD solvers in order to steer adaptive discretization schemes, complementing traditional error estimation techniques. With increasing problem sizes the importance of combining automatic feature-extraction and interactive analysis techniques will also increase. While the time-dependent topology concepts presented in this thesis provide compact abstract field representations, in the case of very large fields, visual analytics techniques could help in detecting and classifying transitions in the topological structure.

Scientific visualization has become an indispensable part of the interdisciplinary field of scientific computing. It not only helps in gaining insight into the simulated phenomena, but also enables an efficient communication of scientific results across research disciplines, e.g., within SimTech, as well as to the general public. This thesis has demonstrated that there is a great demand for advanced flow visualization concepts and techniques, which is expected to grow in the future, as large interactive CFD experiments become available.

Bibliography

[1] A. Agranovsky, C. Garth, and K. I. Joy. Extracting flow structures using sparse particles. In *Vision, Modeling, and Visualization (VMV 2011)*, pages 153–160, 2011. 141

[2] J. Ahrens, E. W. Bethel, and K. Moreland. Scientific discovery at the exascale: report from the DOE ASCR 2011 workshop on exascale data management, analysis and visualization. Technical Report February, 2011. 12, 14

[3] J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. In *Eurographics 1987*, pages 3–10, 1987. 88

[4] M. Ament, S. Frey, F. Sadlo, T. Ertl, and D. Weiskopf. GPU-based two-dimensional flow simulation steering using coherent structures. In *2nd International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering, paper 18*, 2011. 187, 188

[5] Ansys. *ANSYS CFX-solver modeling guide (release 13)*. Number November. ANSYS Inc., 13 edition, 2010. 103

[6] ANSYS. *ANSYS CFX-solver theory guide (release 13)*. ANSYS Inc., 2010. 103, 105

[7] D. Asimov. Notes on the topology of vector fields and flows. Technical report, NASA Ames Research Center (RNR-93-003), 1993. 146, 148

[8] C. Aykanat, B. A. Cambazoglu, F. Findik, and T. Kurc. Adaptive decomposition and remapping algorithms for object-space-parallel direct volume rendering of unstructured grids. *Journal of Parallel and Distributed Computing*, 67(1):77–99, 2007. 94

[9] I. Babuska, G. Caloz, and J. E. Osborn. Special finite element methods for a class of second order elliptic problems with rough coefficients. *SIAM Journal on Numerical Analysis*, 31(4):945–981, 1994. 63

[10] S. Bachthaler, F. Sadlo, C. Dachsbacher, and D. Weiskopf. Space-time visualization of dynamics in Lagrangian coherent structures of time-dependent 2D vector fields. In *International Conference on Information Visualization Theory and Applications*, pages 573–583, 2012. 134

[11] S. Bake, D. G. W. Meyer, and U. Rist. Turbulence mechanism in Klebanoff transition: a quantitative comparison of experiment and direct numerical simulation. *Journal of Fluid Mechanics*, 459:217–243, 2002. 42

[12] W. Barth and W. Stürzlinger. Efficient ray tracing for Bezier and B-spline surfaces. *Computers & Graphics*, 17(4):423–430, 1993. 89

[13] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, M. Ohlberger, R. Kornhuber, and O. Sander. A generic grid interface for parallel and adaptive scientific computing. Part II: implementation and tests in DUNE. *Computing*, 82(2-3):121–138, 2008. 188

[14] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, M. Ohlberger, and O. Sander. A generic grid interface for parallel and adaptive scientific computing. Part I: abstract framework. *Computing*, 82(2-3):103–119, 2008. 188

[15] T. Belytschko and T. Black. Elastic crack growth in finite elements with minimal remeshing. *International Journal for Numerical Methods in Engineering*, 45:601–620, 1999. 63

[16] G. Benettin, L. Galgani, A. Giorgilli, and J.-M. Strelcyn. Lyapunov characteristic exponents for smooth dynamical systems and for Hamiltonian systems; a method for computing all of them. Part 1: theory. *Meccanica*, 15(1):9–20, 1980. 130

[17] S. Bergner, T. Möller, D. Weiskopf, and D. J. Muraki. A spectral analysis of function composition and its implications for sampling in direct volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1353–1360, 2006. 50, 51, 90

[18] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. Technical report, Xerox Palo Alto Research Center, 1992. 22

[19] J. F. Blinn. Models of light reflection for computer synthesized pictures. *Computer Graphics (Proceedings of SIGGRAPH 1977)*, 11(2):192–198, 1977. 46

[20] D. Blythe. The Direct3D 10 system. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006)*, 25(3):724–734, 2006. 13

[21] K. Brodlie. Visualization techniques. In *Scientific Visualization: Techniques and Applications*, pages 37–86. Springer, 1992. 8

[22] K. Brodlie, D. Duce, J. Gallop, and J. Wood. Distributed cooperative visualization. In *EuroGraphics 1998 State of the Art Reports (STARs)*, pages 27–50, 1998. 10

[23] A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik, and O. O. Storaasli. State-of-the-art in heterogeneous computing. *Scientific Programming*, 18(1):1–33, 2010. 14

[24] S. Bryson and C. Levit. The virtual wind tunnel. *IEEE Computer Graphics and Applications*, 12(4):25–34, 1992. 54

[25] L. Buatois, G. Caumon, and B. Levy. Concurrent number cruncher: a GPU implementation of a general sparse linear solver. *International Journal of Parallel, Emergent and Distributed Systems*, 24(3):205–223, 2009. 188

[26] K. Bürger, F. Ferstl, H. Theisel, and R. Westermann. Interactive streak surface visualization on the GPU. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1259–1266, 2009. 54

[27] K. Bürger, J. Schneider, P. Kondratieva, J. Krüger, and R. Westermann. Interactive visual exploration of unsteady 3D flows. In *Joint Eurographics IEEE-VGTC Symposium on Visualization (EuroVis 2007)*, pages 251–258, 2007. 54

[28] B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Symposium on Volume Visualization (VVS 1994)*, pages 91–98, 1994. 52

[29] B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In *SIGGRAPH 1993*, pages 263–270, 1993. 55

[30] A. Cedilnik, B. Geveci, K. Moreland, J. Ahrens, and J. Favre. Remote large data visualization in the ParaView framework. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV 2006)*, pages 162–170, 2006. 10

[31] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org. 104

[32] L. Chen, O. Villa, S. Krishnamoorthy, and G. R. Gao. Dynamic load balancing on single- and multi-GPU systems. In *IEEE International Symposium on Parallel & Distributed Processing (IPDPS 2010)*, pages 1–12. IEEE, 2010. 98

[33] M. S. Chong, A. E. Perry, and B. J. Cantwell. A general classification of three-dimensional flow fields. *Physics of Fluids A: Fluid Dynamics*, 2(5):765–777, 1990. 146

[34] P. Cignoni, P. Marino, C. Montani, E. Puppo, and R. Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158–170, 1997. 45

[35] B. Cockburn, G. E. Karniadakis, and C.-W. Shu. *Discontinuous Galerkin methods. Theory, computation and applications.* Lecture Notes in Computational Science and Engineering. Springer, 2000. 60

[36] C. D. Correa, R. Hero, and K.-L. Ma. A comparison of gradient estimation methods for volume rendering on unstructured meshes. *IEEE Transactions on Visualization and Computer Graphics*, 17(3):305–319, 2011. 25

[37] P. Crossno, D. Rogers, and C. Garasi. Case study: visual debugging of finite element codes. In *IEEE Visualization 2002*, pages 517–520, 2002. 98

[38] T. J. Cullip and U. Neumann. Accelerating volume reconstruction with 3D texture hardware. Technical report, University of North Carolina, 1994. 52

[39] J. Danskin and P. Hanrahan. Fast algorithms for volume ray tracing. In *Workshop on Volume Visualization (VVS 1992)*, pages 91–98, 1992. 50

[40] David Eberly. *Ridges in image and data analysis*. Kluwer Academic Publishers, 1996. 113, 167

[41] R. Ding and J. Li. Nonlinear finite-time Lyapunov exponent and predictability. *Physics Letters A*, 364(5):396–400, 2007. 139

[42] R. A. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. *Computer Graphics (Proceedings of SIGGRAPH 1985)*, 22(4):65–74, 1988. 52

[43] C. A. M. Duarte and J. T. Oden. Hp clouds – a meshless method to solve boundary value problems. *Numerical Methods for Partial Differential Equations*, 12:673–705, 1996. 63

[44] J. Duato, A. J. Peña, F. Silla, R. Mayo, and E. S. Quintana-Ortí. rCUDA: reducing the number of GPU-based accelerators in high performance clusters. In *International Conference on High Performance Computing and Simulation (HPCS)*, pages 224 – 231, 2010. 14

[45] K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *SIGGRAPH/Eurographics Workshop on Graphics Hardware (HWWS '01)*, pages 9–16, 2001. 51

[46] N. Fabian, K. Moreland, D. Thompson, A. C. Bauer, P. Marion, B. Geveci, M. Rasquin, and K. E. Jansen. The ParaView coprocessing library: a scalable, general purpose in situ visualization library. In *IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV 2011)*, pages 89–96, 2011. 8, 189

[47] M. Falk, A. Seizinger, F. Sadlo, M. Üffinger, and D. Weiskopf. Trajectory-augmented visualization of Lagrangian coherent structures in unsteady flow. In *International Symposium on Flow Visualization (ISFV14)*, 2010. 4, 134

[48] M. Falk and D. Weiskopf. Output-sensitive 3D line integral convolution. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):820–834, 2008. 55

[49] C. L. Fefferman. Existence and smoothness of the navier–stokes equation. *The millennium prize problems*, pages 57–67, 2006. 41

[50] F. Ferstl, K. Bürger, H. Theisel, and R. Westermann. Interactive separating streak surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1569–1577, 2010. 153, 157

[51] J. H. Ferziger and M. Peric. *Computational methods for fluid dynamics*. Springer, 3 edition, 2002. 29, 32, 38

[52] R. P. Feynman, R. B. Leighton, and M. L. Sands. *The Feynman lectures on physics: volume II.* Addison-Wesley, 6th edition, 1976. 38, 39

[53] B. Flemisch, M. Darcis, K. Erbertseder, B. Faigle, A. Lauser, K. Mosthaf, P. Nuske, A. Tatomir, M. Wolff, and R. Helmig. DuMux : DUNE for multi-{phase , component , scale , physics, ...} flow and transport in porous media. *Advances in Water Resources*, 34(9):1102–1112, 2010. 188

[54] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer graphics: principles and practice.* Addison Wesley, 2nd edition, 1990. 11

[55] J. Freire, D. Koop, E. Santos, and C. T. Silva. Provenance for computational tasks: a survey. *Computing in Science and Engineering*, 10(3):11–21, 2008. 9

[56] P. J. Frey and P. L. George. *Mesh Generation: application to finite elements.* Hermes Science Publishing, 2000. 21

[57] S. Frey and T. Ertl. PaTraCo: a framework enabling the transparent and efficient programming of heterogeneous compute networks. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV 2010)*, pages 131–140, 2010. 15

[58] T.-P. Fries and T. Belytschko. The extended/generalized finite element method: an overview of the method and its applications. *International Journal for Numerical Methods in Engineering*, 84(3):253–304, 2010. 63

[59] R. Fuchs and H. Hauser. Visualization of Multi-Variate Scientific Data. *Computer Graphics Forum*, 28(6):1670–1690, Sept. 2009. 116

[60] R. Fuchs, J. Kemmler, B. Schindler, J. Waser, F. Sadlo, H. Hauser, and R. Peikert. Toward a Lagrangian vector field topology. *Computer Graphics Forum*, 29(3):1163–1172, 2010. 131, 150, 151, 157

[61] J. D. Furst and S. M. Pizer. Marching ridges. In *International Conference on Signal and Image Processing (IASTED 2001)*, pages 22–26, 2001. 114

[62] M. P. Garrity. Raytracing irregular volume data. In *Workshop on Volume Visualization (VVS 1990)*, pages 35–40, 1990. 52, 84

[63] 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. 131, 141

[64] C. Garth and K. I. Joy. Fast, memory-efficient cell location in unstructured grids for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1541–1550, 2010. 52, 167

[65] C. Garth, H. Krishnan, X. Tricoche, T. Bobach, and K. I. Joy. Generation of accurate integral surfaces in time-dependent vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1404–1411, 2008. 54

[66] C. Garth, X. Tricoche, T. Salzbrunn, T. Bobach, and G. Scheuermann. Surface techniques for vortex visualization. In *IEEE TCVG Symposium on Visualization (VisSym 2004)*, pages 155–164, 2004. 54

[67] G. Gassner. *Discontinuous Galerkin methods for the unsteady compressible Navier-Stokes equations*. PhD thesis, Universität Stuttgart, 2009. 61

[68] G. Gassner, F. Lörcher, and C.-D. Munz. A discontinuous Galerkin scheme based on a space-time expansion II: viscous flow equations in multi dimensions. *Journal of Scientific Computing*, 34(3):260–286, 2008. 42, 61

[69] M. Geimer and O. Abert. Interactive ray tracing of trimmed bicubic Bézier surfaces without triangulation. In *Winter School of Computer Graphics (WSCG 2005)*, pages 71–78, 2005. 89

[70] J. Gleick. *Chaos - making a new science*. Penguin Books, 1987. 41, 186

[71] A. Globus, C. Levit, and T. Lasinski. A tool for visualizing the topology of three-dimensional vector fields. In *IEEE Visualization 1991*, pages 33–40, 1991. 146

[72] I. Goldhirsch, P.-L. Sulem, and S. A. Orszag. Stability and Lyapunov stability of dynamical systems: a differential approach and a numerical method. *Physica D: Nonlinear Phenomena*, 27(3):311–337, 1987. 130

[73] M. Griebel and M. A. Schweitzer. A particle-partition of unity method—part III: a multilevel solver. *SIAM Journal on Scientific Computing*, 24(2):377–409, 2002. 64

[74] S. Grottel. *Point-based visualization of molecular dynamics data set*. PhD thesis, Universität Stuttgart, 2012. 189

[75] B. Haasdonk, M. Ohlberger, M. Rumpf, A. Schmidt, and K. G. Siebert. Multiresolution visualization of higher order adaptive finite element simulations. *Computing*, 70(3):181–204, 2003. 67

[76] R. Haber and D. A. McNabb. Visualization idioms: a conceptual model for scientific visualization systems. In *Visualization in Scientific Computing*, pages 74–93. IEEE Computer Society Press, 1990. 8

[77] G. Haller. Distinguished material surfaces and coherent structures in three-dimensional fluid flows. *Physica D*, 149(4):248–277, 2001. 152, 160

[78] G. Haller. Lagrangian structures and the rate of strain in a partition of two-dimensional turbulence. *Physics of Fluids*, 13(11):3365–3385, 2001. 131, 151

[79] J. Helman and L. Hesselink. Representation and display of vector field topology in fluid flow data sets. *Computer*, 22(8):27–36, 1989. 146, 148, 149

[80] J. Helman and L. Hesselink. Visualizing vector field topology in fluid flows. *IEEE Computer Graphics and Applications*, 11(3):36–46, 1991. 146, 148

[81] M. Hlawatsch and F. Sadlo. Hierarchical line integration. *IEEE Transactions on Visualization and Computer Graphics*, 17(8):1148–1163, 2011. 54

[82] J. Jeong and F. Hussain. On the identification of a vortex. *Journal of Fluid Mechanics*, 285:69–94, 1995. 114

[83] C. R. Johnson, C. Chen, and R. B. Ross. Visualization viewpoints: the top 10 challenges in extreme-scale visual analytics. *IEEE Computer Graphics and Applications*, 32(4):63–67, 2012. 15, 187

[84] C. R. Johnson, S. Parker, D. Weinstein, and H. Sean. Component-based problem solving environments for large-scale scientific computing. *Concurrency and Computation: Practice & Experience*, 14(13-15):1337–1349, 2002. 8

[85] T. Ju, S. Schaefer, and J. Warren. Mean value coordinates for closed triangular meshes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2005)*, 24(3):561–566, 2005. 23, 104

[86] G. Karch, F. Sadlo, D. Weiskopf, C.-D. Munz, and T. Ertl. Visualization of advection-diffusion in unsteady fluid flow. *Computer Graphics Forum*, 31(3):1105–1114, 2012. 190

[87] G. Karniadakis and S. Sherwin. *Spectral/hp element methods for CFD*. Numerical Mathematics and Scientific Computation. Oxford University Press, 1999. 61, 62

[88] J. Kasten, I. Hotz, B. R. Noack, and H.-C. Hege. On the extraction of long-living features in unsteady fluid flows. In *Topological Methods in Data Analysis and Visualization*, pages 115–126. Springer, 2010. 150, 151, 157

[89] J. Kasten, C. Petz, I. Hotz, B. R. Noack, and H.-C. Hege. Localized finite-time Lyapunov exponent for unsteady flow analysis. *Vision Modeling and Visualization (VMV 2009)*, pages 265–274, 2009. 131

[90] P. Kaufmann, S. Martin, M. Botsch, E. Grinspun, and M. Gross. Enrichment textures for detailed cutting of shells. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2009)*, 28(3):50:1–50:10, 2009. 82

[91] D. Keim, J. Kohlhammer, G. Ellis, and F. Mansmann. *Mastering the information age: solving problems with visual analytics*. Eurographics Association, 2010. 7

[92] D. N. Kenwright. Automatic detection of open and closed separation and attachment lines. In *IEEE Visualization 1998*, pages 151–158, 1998. 113

[93] D. N. Kenwright, C. Henze, and C. Levit. Feature extraction of separation and attachment lines. *IEEE Transactions on Visualization and Computer Graphics*, 5(2):135–144, 1999. 101, 113

[94] P. Kipfer, F. Reck, and G. Greiner. Local exact particle tracing on unstructured grids. *Computer Graphics Forum*, 22(2):133–142, 2003. 54

[95] A. Klöckner, T. Warburton, J. Bridge, and J. S. Hesthaven. Nodal discontinuous Galerkin methods on graphics processors. *Journal of Computational Physics*, 228(21):7863–7882, 2009. 60

[96] J. Kniss, G. Kindlmann, and C. Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):270–285, 2002. 49

[97] J. Kniss, S. Premoze, C. Hansen, P. Shirley, and A. McPherson. A model for volume lighting and modeling. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):150–162, 2003. 49

[98] A. Knoll, Y. Hijazi, A. Kensler, M. Schott, C. Hansen, and H. Hagen. Fast ray tracing of arbitrary implicit surfaces with interval and affine arithmetic. *Computer Graphics Forum*, 28(1):26–40, 2009. 67

[99] M. Kraus and T. Ertl. Cell-projection of cyclic meshes. In *IEEE Visualization 2001*, pages 215–222, 2001. 52, 84

[100] H. Krishnan, C. Garth, and K. I. Joy. Time and streak surfaces for flow visualization in large time-varying data sets. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1267–1274, 2009. 54

[101] J. Krüger, P. Kipfer, P. Kondratieva, and R. Westermann. A particle system for interactive visualization of 3D flows. *IEEE Transactions on Visualization and Computer Graphics*, 11(6):744–756, 2005. 123, 126

[102] J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *IEEE Visualization 2003*, pages 287–292, 2003. 52

[103] R. Laramee, C. Garth, H. Doleisch, J. Schneider, H. Hauser, and H. Hagen. Visual analysis and exploration of fluid flow in a cooling jacket. In *IEEE Visualization 2005*, pages 623–630, 2005. 101

[104] R. S. Laramee, H. Hauser, L. Zhao, and F. H. Post. Topology-based flow visualization: the state of the art. In *Topology-based Methods in Visualization*, pages 1–19. Springer, 2007. 146

[105] R. S. Laramee, J. Schneider, and H. Hauser. Texture-based flow visualization on isosurfaces from computational fluid dynamics. In *Joint Eurographics IEEE-TCVG Symposium on Visualization (VisSym 2004)*, pages 85–90, 2004. 55

[106] B. Launder and D. Spalding. The numerical computation of turbulent flows. *Computer Methods in Applied Mechanics and Engineering*, 3(2):269–289, 1974. 102

[107] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, 1988. 49, 52

[108] L. Li and H. W. Shen. Image-based streamline generation and rendering. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):630–640, 2007. 116

[109] T.-Y. Li and J. A. Yorke. Period three implies chaos. *The American Mathematical Monthly*, 82(10):985–992, 1975. 41

[110] B. Liu, A. Bock, E. Sund, and W. Burkhard. Coherency-based curve compression for high-order finite element model visualization. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2315–2324, 2012. 67, 84, 100

[111] H. Löffelmann, H. Doleisch, and E. Gröller. Visualizing Dynamical Systems near Critical Points. In *Spring Conference on Computer Graphics and its Applications 1998*, pages 175–184, 1998. 148

[112] H. Löffelmann and E. Gröller. Enhancing the visualization of characteristic structures in dynamical systems. In *Eurographics Workshop on Visualization in Scientific Computing (VisSci 1998)*, pages 59–68, 1998. 148

[113] H. Löffelmann, T. Kucera, and E. Gröller. Visualizing Poincaré maps together with the underlying flow. In *International Workshop on Visualization and Mathematics 1997*, pages 315–328, 1997. 148

[114] W. E. Lorensen and H. E. Cline. Marching cubes: a high resolution 3D surface construction algorithm. *Computer Graphics (Proceedings of SIGGRAPH 1987)*, 21(4):163–169, 1987. 45

[115] E. N. Lorenz. Deterministic nonperiodic flow. *Journal of Atmospheric Sciences*, 20(2):130–148, 1963. 41

[116] K.-L. Ma. Parallel volume ray-casting for unstructured-grid data on distributed-memory architectures. In *IEEE Symposium on Parallel Rendering 1995*, pages 23–30, 1995. 94

[117] G. Marmitt, H. Friedrich, and P. Slusallek. Interactive volume rendering with ray tracing. *Eurographics 2006 State of the Art Reports (STARs)*, pages 115–136, 2006. 84

[118] S. R. Marschner and R. J. Lobb. An evaluation of reconstruction filters for volume rendering. In *IEEE Visualization 1994*, pages 100–107, 1994. 24

[119] N. Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995. 47, 48, 52

[120] N. Max, R. Crawfis, and C. Grant. Visualizing 3D velocity fields near contour surfaces. In *IEEE Visualization 1994*, pages 248–255, 1994. 116

[121] B. H. McCormick, T. A. DeFanti, and M. D. Brown. Visualization in scientific computing. *Computer Graphics (Special Issue)*, 21(6):1–99, 1987. 7

[122] 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. 53

[123] M. Meyer, B. Nelson, R. M. Kirby, and R. Whitaker. Particle systems for efficient and accurate high-order finite element visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(5):1015–1026, 2007. 68

[124] H. Miura and S. Kida. Identification of tubular vortices in turbulence. *Journal of the Physics Society Japan*, 66(5):1331–1334, 1997. 113

[125] N. Moës, J. Dolbow, and T. Belytschko. A finite element method for crack growth without remeshing. *International Journal for Numerical Methods in Engineering*, 46:131–150, 1999. 63

[126] T. Möller, R. Machiraju, K. Mueller, and R. Yagel. Classification and local error estimation of interpolation and derivative filters for volume rendering. In *Symposium on Volume Visualization*, pages 71–78, 1996. 25

[127] T. Möller, R. Machiraju, K. Mueller, and R. Yagel. A comparison of normal estimation schemes. In *IEEE Visualization 1997*, pages 19–26, 1997. 25

[128] J. D. Mulder, J. J. V. Wijk, and R. V. Liere. A survey of computational steering environments. *Future Generation Computer Systems*, 15(1):119–129, 1999. 9, 187

[129] C. Müller, M. Strengert, and T. Ertl. Adaptive load balancing for raycasting of non-uniformly bricked volumes. *Parallel Computing*, 33(6):406–419, 2007. 94, 95

[130] A. Munshi, B. Gaster, T. G. Mattson, J. Fung, and D. Ginsburg. *OpenCL programming guide*. Addison-Wesley, 2011. 13

[131] Z. Nagy and R. Klein. Depth-peeling for texture-based volume rendering. In *Pacific Conference on Computer Graphics and Applications 2003*, pages 429–433, 2003. 117

[132] B. Nelson, R. Haimes, and R. M. Kirby. GPU-based interactive cut-surface extraction from high-order finite element fields. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1803–1811, 2011. 67, 100

[133] B. Nelson and R. M. Kirby. Ray-tracing polymorphic multidomain spectral/hp elements for isosurface rendering. *IEEE Transactions on Visualization and Computer Graphics*, 12(1):114–125, 2006. 67, 84

[134] B. Nelson, E. Liu, R. M. Kirby, and R. Haimes. ElVis: a system for the accurate and interactive visualization of high-order finite element solutions. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2325–2334, 2012. 67

[135] J. Nese. Quantifying local predictability in phase space. *Physica D: Nonlinear Phenomena*, 35(1-2):237–250, 1989. 130, 131

[136] J. Nickolls and W. J. Dally. The GPU computing era. *IEEE Micro*, 30(2):56–69, 2010. 13, 14

[137] G. M. Nielson. Tools for computing tangent curves for linearly varying vector fields over tetrahedral domains. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):360–372, 1999. 54

[138] B. Nouanesengsy, T.-Y. Lee, and H.-W. Shen. Load-balanced parallel streamline generation on large scale vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1785–1794, 2011. 127

[139] NVIDIA Corporation. CUDA C Programming Guide, 2012. 13, 14

[140] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, T. J. Purcell, and E. Aaron. A survey of general-purpose computation on graphics hardware. In *Eurographics 2005 State of the Art Reports (STARs)*, pages 21–51, 2005. 13

[141] C. A. Pagot, D. Osmari, F. Sadlo, D. Weiskopf, T. Ertl, and J. L. D. Comba. Efficient parallel vectors feature extraction from higher-order data. *Computer Graphics Forum*, 30(3):751–760, 2011. 68

[142] C. A. Pagot, J. E. Vollrath, F. Sadlo, D. Weiskopf, T. Ertl, and J. L. D. Comba. Interactive isocontouring of high-order surfaces. In *Scientific Visualization: Interactions, Features, Metaphors (Dagstuhl Follow-Ups)*, pages 276–291, 2011. 68

[143] A. Panagiotidis, D. Kauker, S. Frey, and T. Ertl. DIANA: a device abstraction framework for parallel computation. In *International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering*, pages 20:1–20:16, 2011. 15

[144] S. W. Park, B. Budge, L. Linsen, B. Hamann, and K. I. Joy. Dense geometric flow visualization. In K. Brodlie, D. Duke, and K. Joy, editors, *Joint Eurographics IEEE-VGTC Symposium on Visualization (EuroVis 2005)*, pages 21–28, 2005. 123

[145] S. Parker, M. Miller, C. Hansen, and C. Johnson. An integrated problem solving environment: the SCIRun computational steering system. In *Proceedings of the Thirty-First Hawaii International Conference on System Sciences*, pages 147–156, 1998. 9

[146] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P.-P. Sloan. Interactive ray tracing for isosurface rendering. In *IEEE Visualization 1998*, pages 233–238, 1998. 45

[147] R. Peikert and M. Roth. The "parallel vectors" operator: a vector field visualization primitive. In *IEEE Visualization 1999*, pages 33–41, 1999. 113, 114

[148] J. Peiro and S. Sherwin. Finite difference, finite element and finite volume methods for partial differential equations. In *Handbook of Materials Modeling*, volume M, pages 2415–2446. Springer, 2005. 29, 30, 31

[149] A. E. Perry and M. S. Chong. A description of eddying motions and flow patterns using critical point concepts. *Annual Review of Fluid Mechanics*, 19(1):125–155, 1987. 146, 149

[150] C. Petz, S. Prohaska, L. Goubergrits, U. Kertzscher, and H.-C. Hege. Near-wall flow visualization in flattened surface neighborhoods. In *Simulation and Visualization (SimVis 2008)*, pages 93–106, 2008. 101

[151] A. Pobitzer, R. Peikert, R. Fuchs, B. Schindler, A. Kuhn, H. Theisel, and K. Matkovi. On the way towards topology-based visualization of unsteady flow – the state of the art. In *EuroGraphics 2010 State of the Art Reports (STARs)*, pages 137–154, 2010. 149

[152] A. Pobitzer, R. Peikert, R. Fuchs, B. Schindler, A. Kuhn, H. Theisel, K. Matković, and H. Hauser. The state of the art in topology-based visualization of unsteady flow. *Computer Graphics Forum*, 30(6):1789–1811, 2011. 131

[153] 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. 113

[154] J.-F. Remacle, N. Chevaugeon, E. Marchandise, and C. Geuzaine. Efficient visualization of high-order finite elements. *International Journal for Numerical Methods in Engineering*, 69(4):750–771, 2007. 66

[155] P. Sabella. A rendering algorithm for visualizing 3D scalar fields. *Computer Graphics (Proceedings of SIGGRAPH 1988)*, 22(4):51–58, 1988. 47

[156] 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. 114, 131, 141, 152, 158, 166, 167, 171, 175

[157] F. Sadlo and R. Peikert. Visualizing Lagrangian coherent structures and comparison to vector field topology. In *Topology-Based Methods in Visualization II*, pages 15–29. Springer, 2007. 131

[158] 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. 101

[159] F. Sadlo, M. Üffinger, T. Ertl, and D. Weiskopf. On the finite-time scope for computing Lagrangian coherent structures from Lyapunov exponents. In *Topological Methods in Data Analysis and Visualization II (TopoInVis 2011)*, pages 269–281. Springer, 2012. 4, 151, 152, 153

[160] F. Sadlo, M. Üffinger, C. Pagot, D. Osmari, J. L. D. Comba, T. Ertl, C.-D. Munz, and D. Weiskopf. Visualization of cell-based higher-order fields. *Computing in Science and Engineering*, 13(3):84–91, 2011. 59, 66

[161] 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. 141, 153, 157, 158, 159, 160, 166, 172, 175, 181, 185

[162] J. Sahner, J. Sahner, T. Weinkauf, and H.-C. Hege. Galilean invariant extraction and iconic representation of vortex core lines. In *Joint Eurographics IEEE-VGTC Symposium on Visualization (EuroVis 2005)*, pages 151–160, 2005. 113

[163] W. Sander and B. Weigand. Direct numerical simulation and analysis of instability enhancing parameters in liquid sheets at moderate Reynolds numbers. *Physics of Fluids*, 20(5):053301, 2008. 43, 115

[164] T. Schafhitzel, J. E. Vollrath, J. P. Gois, D. Weiskopf, A. Castelo, and T. Ertl. Topology-preserving $\lambda 2$-based vortex core line detection for flow visualization. *Computer Graphics Forum*, 27(3):1023–1030, 2008. 114

[165] G. Scheuermann, H. Krüger, M. Menzel, and A. Rockwood. Visualizing nonlinear vector field topology. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):109–116, 1998. 149

[166] T. Schilli. *Analyse von FTLE-Feldern (Analysis of FTLE fields).* Diploma thesis, Universität Stuttgart, 2011. 135

[167] H. Schlichting and K. Gersten. *Boundary-layer theory.* Springer, 8th edition, 2000. 39, 102

[168] J. Schlottke and B. Weigand. Direct numerical simulation of evaporating droplets. *Journal of Computational Physics*, 227(10):5215–5237, 2008. 43

[169] W. J. Schroeder, F. Bertel, M. Malaterre, D. Thompson, P. P. Pebay, R. O'Bara, and S. Tendulkar. Methods and framework for visualizing higher-order finite elements. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):446–460, 2006. 66

[170] T. Schultz, H. Theisel, and H.-P. Seidel. Crease surfaces: from theory to extraction and application to diffusion tensor MRI. *IEEE Transactions on Visualization and Computer Graphics*, 16(1):109–119, 2010. 114

[171] M. A. Schweitzer. *A parallel multilevel partition of unity method for elliptic partial differential equations*, volume 29. Springer, 2003. 43

[172] M. A. Schweitzer. Stable enrichment and local preconditioning in the particle-partition of unity method. *Numerische Mathematik*, 118:307–328, 2011. 64

[173] M. A. Schweitzer. Generalizations of the finite element method. *Central European Journal of Mathematics*, 10(1):3–24, 2012. 63

[174] M. Segal and K. Akeley. *The OpenGL graphics system: a specification (version 4.3)*. The Khronos Group Inc., 2012. 10, 11

[175] S. C. Shadden, F. Lekien, and J. E. Marsden. Definition and properties of Lagrangian coherent structures from finite-time Lyapunov exponents in two-dimensional aperiodic flows. *Physica D*, 212(3-4):271–304, 2005. 134, 150, 152, 156, 175

[176] C. Shannon. Communication in the presence of noise. *Proceedings of the Institute of Radio Engineers (IRE)*, 37(2):10–21, 1949. 22

[177] D. Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *ACM National Conference*, pages 517–524, 1968. 24, 65

[178] K. Shi, H. Theisel, T. Weinkauf, H. Hauser, H.-C. Hege, and H.-P. Seidel. Path line oriented topology for periodic 2D time-dependent vector fields. In *Joint Eurographics IEEE-VGTC Symposium on Visualization (EuroVis 2006)*, pages 139–146, 2006. 150

[179] P. Shirley and A. Tuchman. A polygonal approximation to direct scalar volume rendering. *Computer Graphics (Proceedings of SIGGRAPH 1990)*, 24(5):63–70, 1990. 52

[180] S. Stegmaier, M. Strengert, T. Klein, and T. Ertl. A simple and flexible volume rendering framework for graphics-hardware-based raycasting. In *Joint Eurographics IEEE-VGTC Conference on Volume Graphics (VG 2005)*, pages 187–195, 2005. 52, 117

[181] M. Strengert, C. Müller, C. Dachsbacher, and T. Ertl. CUDASA: compute unified device and systems architecture. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV 2008)*, pages 49–56, 2008. 14

[182] T. Strouboulis, I. Babuška, and K. Copps. The design and analysis of the generalized finite element method. *Computer Methods in Applied Mechanics and Engineering*, 181(I 3):43–69, 2000. 63

[183] C. Teitzel, R. Grosso, and T. Ertl. Efficient and reliable integration methods for particle tracing in unsteady flows on discrete meshes. In *Visualization in Scientific Computing*, pages 31–41. Springer, 1997. 27, 54

[184] H. Theisel and H.-P. Seidel. Feature flow fields. In *Joint Eurographics IEEE-TCVG Symposium on Visualization (VisSym 2004)*, pages 141–148, 2003. 149

[185] H. Theisel, T. Weinkauf, H.-C. Hege, and H.-P. Seidel. Saddle connectors - an approach to visualizing the topological skeleton of complex 3D vector fields. In *IEEE Visualization 2003*, pages 225–232, 2003. 148, 150

[186] J. J. Thomas and K. A. Cook, editors. *Illuminating the path: the research and development agenda for visual analytics*. IEEE Computer Society Press, 2005. 7

[187] B. Thomaß. *GPU-basierte simulation und in-situ visualisierung von 3D Fluiden in porösen Medien*. Diploma thesis, Universität Stuttgart, 2010. 188

[188] B. Thomaß, M. Üffinger, S. Müthing, and T. Ertl. GPU-based simulation and in-situ visualization of 3D flow in porous media. In *SimTech Conference 2011 (Posters)*, 2011. 188

[189] X. Tricoche, G. Scheuermann, and H. Hagen. A topology simplification method for 2D vector fields. In *IEEE Visualization 2000*, pages 359–366, 2000. 148

[190] X. Tricoche, G. Scheuermann, and H. Hagen. Continuous topology simplification of planar vector fields. In *IEEE Visualization 2001*, pages 159–166, 2001. 148

[191] X. Tricoche, T. Wischgoll, G. Scheuermann, and H. Hagen. Topology tracking for the visualization of time-dependent two-dimensional flows. *Computers & Graphics*, 26(2):249–257, 2002. 149

[192] G. Turk and D. C. Banks. Image-guided streamline placement. In *SIGGRAPH 1996*, pages 453–460, 1996. 54

[193] M. Üffinger. *GPU-basierte Visualisierung von Strömungen auf impliziten Flächen*. Diploma thesis, Universität Stuttgart, 2007. 123

[194] M. Üffinger, S. Frey, and T. Ertl. Interactive high-quality visualization of higher-order finite elements. *Computer Graphics Forum*, 29(2):115–136, 2010. 3, 59, 69, 83

[195] M. Üffinger, T. Klein, M. Strengert, and T. Ertl. GPU-based streamlines for surface-guided 3D flow visualization. In *Vision, Modeling, and Visualization (VMV 2008)*, pages 90–100, 2008. 4, 54, 115, 123, 126

[196] 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. 4, 157

[197] M. Üffinger, F. Sadlo, R. M. Kirby, C. Hansen, and T. Ertl. FTLE computation beyond first-order approximation. In *Eurographics 2012 (Short Papers)*, pages 61–64, 2012. 4, 129, 136

[198] M. Üffinger, F. Sadlo, C.-D. Munz, and T. Ertl. Toward wall function consistent interpolation of flow fields. In *EuroVis (Short Papers)*, pages 85–89, 2013. 3

[199] M. Üffinger, M. A. Schweitzer, F. Sadlo, and T. Ertl. Direct visualization of particle-partition of unity data. In *Vision, Modeling, and Visualization (VMV 2011)*, pages 255–262, 2011. 3, 59, 69

[200] C. Upson, T. Faulhaber, D. Kamins, D. H. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. V. Dam. The application visualization system: a computational environment for scientific visualization. *Computer Graphics and Applications*, 9(4):30–42, 1989. 8

[201] H. T. Vo, S. P. Callahan, N. Smith, C. T. Silva, W. Martin, D. Owen, and D. Weinstein. iRun : interactive rendering of large unstructured grids. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV 2007)*, pages 93–100, 2007. 94

[202] J. Waser, R. Fuchs, H. Ribicic, B. Schindler, G. Blöschl, and E. Gröller. World lines. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1458–1467, 2010. 187

[203] T. Weinkauf and H. Theisel. Streak lines as tangent curves of a derived vector field. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1225–1234, 2010. 54

[204] T. Weinkauf, H. Theisel, M. Saarbrücken, K. Shi, and H.-C. Hege. Extracting higher order critical points and topological simplification of 3D vector fields. In *IEEE Visualization 2005*, pages 559–566, 2005. 149

[205] T. Weinkauf, H. Theisel, and H.-P. Seidel. Boundary switch connectors for topological visualization of complex 3D vector fields. In *Joint Eurographics IEEE-TCVG Symposium on Visualization (VisSym 2004)*, pages 183–192, 2004. 148

[206] D. Weiskopf. *GPU-based interactive visualization techniques*. Springer, 2006. 8, 9

[207] D. Weiskopf and T. Ertl. A hybrid physical/device space approach for spatio-temporally coherent interactive texture advection on curved surfaces. In *Graphics Interface (GI 2004)*, pages 263–270, 2004. 55

[208] L. Westover. Footprint evaluation for volume rendering. *Computer Graphics (Proceedings of SIGGRAPH 1990)*, 24(4):367–376, 1990. 52

[209] A. Wiebel, X. Tricoche, D. Schneider, H. Jaenicke, and G. Scheuermann. Generalized streak lines: analysis and visualization of boundary induced vortices. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1735–1742, 2007. 54, 101, 162

[210] D. F. Wiley, H. R. Childs, B. F. Gregorski, B. Hamann, and K. I. Joy. Contouring curved quadratic elements. In *IEEE TCVG Symposium on Visualization (VisSym 2003)*, pages 176–176, 2003. 67

[211] D. F. Wiley, H. R. Childs, B. Hamann, and K. I. Joy. Ray casting curved-quadratic elements. In *Joint Eurographics IEEE-TCVG Symposium on Visualization (VisSym 2004)*, pages 201–210, 2004. 67

[212] J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, 1992. 45

[213] P. L. Williams, N. L. Max, and C. M. Stein. A high accuracy volume renderer for unstructured data. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):37–54, 1998. 49, 52, 67

[214] T. Wischgoll and G. Scheuermann. Detection and visualization of closed streamlines in planar flows. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):165–172, 2001. 148

[215] T. Wischgoll and G. Scheuermann. Locating closed streamlines in 3D vector fields. In *Joint Eurographics IEEE-TCVG Symposium on Visualization (VisSym 2002)*, pages 227–232, 2002. 148

[216] M. Woo, J. Neider, T. Davis, and D. Shreiner. *OpenGL programming guide: the official guide to learning OpenGL*. Addison Wesley, 7th edition, 2009. 11

[217] 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. 9, 187

[218] R. Yagel and Z. Shi. Accelerating volume animation by space-leaping. In *IEEE Visualization 1993*, pages 62–69, 1993. 50

[219] S. Yoden and M. Nomura. Finite-time Lyapunov stability analysis and its application to atmospheric predictability. *Journal of Atmospheric Sciences*, 50(11):1531–1541, 1993. 130

[220] Y. Zhou and M. Garland. Interactive point-based rendering of higher-order tetrahedral data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1229–1236, 2006. 68

[221] Y. Zhou, M. Garland, and R. Haber. Pixel-exact rendering of spacetime finite element solutions. In *IEEE Visualization 2004*, pages 425–432, 2004. 67

[222] G. Ziegler, A. Tevs, C. Theobalt, and H.-P. Seidel. GPU point list generation through histogram pyramids. Technical report, Max Plack Institut für Informatik, 2006. 117

[223] M. Zöckler, D. Stalling, and H.-C. Hege. Interactive visualization of 3D-vector fields using illuminated stream lines. In *IEEE Visualization 1996*, pages 107–113, 1996. 46