Visulization of Complex
Three-Dimensional Flow Structures

Fachbereich Informatik der Technischen Universität Kaiserslautern

Zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)
eingereichte Dissertation von

Christoph Garth

Kaiserslautern, 27. Juni 2007

Datum der wissenschaftlichen Aussprache: 19. Oktober 2007

Dekan: Prof. Dr. Reinhard Gotzhein
Vorsitzender der Prüfufungskommission: Prof. Dr. Klaus Madlener
Erster Berichterstatter: Prof. Dr. Hans Hagen
Zweiter Berichterstatter: Prof. Dr. Gerik Scheuermann

Abstract

The visualization of numerical fluid flow datasets is essential to the engineering processes that motivate their computational simulation. To address the need for visual representations that convey meaningful relations and enable a deep understanding of flow structures, the discipline of Flow Visualization has produced many methods and schemes that are tailored to a variety of visualization tasks. The ever increasing complexity of modern flow simulations, however, puts an enormous demand on these methods. The study of vortex breakdown, for example, which is a highly transient and inherently three-dimensional flow pattern with substantial impact wherever it appears, has driven current techniques to their limits. In this thesis, we propose several novel visualization methods that significantly advance the state of the art in the visualization of complex flow structures.

First, we propose a novel scheme for the construction of stream surfaces from the trajectories of particles embedded in a flow. These surfaces are extremely useful since they naturally exploit coherence between neighboring trajectories and are highly illustrative in nature. We overcome the limitations of existing stream surface algorithms that yield poor results in complex flows, and show how the resulting surfaces can be used a building blocks for advanced flow visualization techniques.

Moreover, we present a visualization method that is based on moving section planes that travel through a dataset and sample the flow. By considering the changes to the flow topology on the plane as it moves, we obtain a method of visualizing topological structures in three-dimensional flows that are not accessible by conventional topological methods. On the same algorithmic basis, we construct an algorithm for the tracking of critical points in such flows, thereby enabling the treatment of time-dependent datasets.

Last, we address some problems with the recently introduced Lagrangian techniques. While conceptually elegant and generally applicable, they suffer from an enormous computational cost that we significantly use by developing an adaptive approximation algorithm. This allows the application of such methods on very large and complex numerical simulations.

Throughout this thesis, we will be concerned with flow visualization aspect of general practical significance but we will particularly emphasize the remarkably challenging visualization of the vortex breakdown phenomenon.

Acknowledgements

I am grateful to have the chance to thank all those people who have helped and supported me working on this dissertation. First of all, I would like to express my sincere thankfulness to Hans Hagen for his advice and support during my doctoral endeavor. As my doctoral advisor, he provided me with valuable suggestions and comments in both scientific and non-scientific matters. I would also like to thank my co-advisor Gerik Scheuermann for providing ongoing support and much needed perspective.

Among the most pleasing aspects of scientific research I consider the intellectual exchange with collaborating researchers. First and foremost, I wish to express my gratitude to Xavier Tricoche for the intense intellectual exchange he provided and the many minute details of scientific practice I was able to learn from him, in spite of geographical separation by the Atlantic Ocean. Furthermore, I am also indebted to my friends and colleagues at both the University of Kaiserslautern and the University of Leipzig, most notably Inga Scheler and Tom Bobach, for providing the friendly, collegial and supportive atmosphere that carried me throughout the genesis of this work. I also wish to thank Markus Rütten, who, by providing me with datasets and discussion, made possible much of this thesis.

Last - but certainly not least - I wish to thank my family: my parents, for their unwavering yet unobtrusive support, and my wife Leonie and my son Alexander, for enduring many moods and working nights, for their ongoing encouragement and confidence, and for showing me that anything is possible.

Contents

1 A Brief Introduction to Flow Visualization 5
- 1.1 Modeling and Simulation . . . 5
  - 1.1.1 The Navier-Stokes Equation . . . 6
  - 1.1.2 Basic Equation . . . 6
  - 1.1.3 Numerical Representation . . . 8
  - 1.1.4 Lagrangian and Eulerian Perspectives . . . 9
  - 1.1.5 Flow Features . . . 10
- 1.2 Feature-Based Visualization . . . 11
  - 1.2.1 Vortices . . . 12
  - 1.2.2 Separation and Attachment Lines . . . 12
- 1.3 Topology-Based Visualization . . . 14
- 1.4 Datasets . . . 14
  - 1.4.1 Delta Wing . . . 14
  - 1.4.2 Rotating Lid Cylinder . . . 15
  - 1.4.3 High-Speed Train . . . 15
  - 1.4.4 Kármán vortex street . . . 16

2 Vector Fields and Dynamical Systems 17
- 2.1 Basic Definitions and Fundamental Properties . . . 17
- 2.2 Existence and Uniqueness . . . 19
- 2.3 Orbits and Invariant Sets . . . 21
- 2.4 Critical Points . . . 23
  - 2.4.1 Linear Vector Fields . . . 23
  - 2.4.2 General Case . . . 27
- 2.5 Topological Graph . . . 30
- 2.6 Poincaré Index . . . 31
- 2.7 Parameter-dependent Dynamical Systems . . . 33
- 2.8 A Note on Divergence-Free Vector Fields . . . 37
- 2.9 Lyapunov Exponents and Chaos . . . 37
- 2.10 Numerical Treatment . . . 38
  - 2.10.1 Basic Properties . . . 38
  - 2.10.2 Runge-Kutta Methods . . . 40

3 Stream Surfaces 43
3.1 Formal Definition . . . 44
3.2 An Overview of Computational Algorithms . . . 45
3.3 Advancing Front Methods . . . 47
  3.3.1 Basic Principles . . . 47
  3.3.2 Front Resolution Control . . . 48
  3.3.3 Hultquist's Implementation . . . 48
3.4 High-Quality Stream Surface Computation . . . 50
  3.4.1 Improved Streamline Integration . . . 51
  3.4.2 Arc Length Sampling . . . 51
  3.4.3 Front Resolution Control . . . 53
  3.4.4 Implementation . . . 55
  3.4.5 Experiments . . . 55
  3.4.6 Comparison to State Of The Art . . . 57
3.5 Stream Surfaces as Visualization Building Blocks . . . 60
  3.5.1 Enhanced Color Mapping . . . 60
  3.5.2 Geometric Vortex Extraction and Verification . . . 62
  3.5.3 Miscellaneous Applications . . . 65
3.6 Discussion . . . 66

4 Topology-Based Methods for Three-Dimensional Datasets 67
4.1 Motivation . . . 67
4.2 Previous Work . . . 68
4.3 Topology Tracking . . . 68
  4.3.1 Linear Interpolation . . . 69
  4.3.2 Critical Point Paths . . . 70
  4.3.3 Algorithm . . . 72
  4.3.4 Practical Concerns . . . 74
4.4 Tracking Extremal Values of a Scalar Field . . . 74
4.5 Moving Section Planes . . . 75
  4.5.1 Plane Trajectory . . . 75
  4.5.2 Planar Resampling and Projection . . . 76
  4.5.3 Section Plane Tracking . . . 76
  4.5.4 Typical Trajectories . . . 77
  4.5.5 Plane Orientation . . . 78
  4.5.6 In-Context Visualization using Volume Rendering . . . 79
4.6 Visualization of Vortices and Vortex Breakdown . . . 80
  4.6.1 Rotating Lid Cylinder . . . 80
  4.6.2 Delta Wing dataset . . . 82
4.7 Discussion . . . 85

5 Lagrangian Techniques **89**
5.1 The Finite Time Lyapunov Exponent 90
5.2 Lagrangian Coherent Structures 91
5.3 Related Work 92
5.4 Numerical Realization 93
  5.4.1 Incremental Approximation of Maps 93
  5.4.2 Efficient FTLE Computation 96
  5.4.3 Ridge Extraction 97
5.5 Visualization of Coherent Flow Structures 98
  5.5.1 Three-dimensional FTLE 99
  5.5.2 FTLE on Planar Sections 100
5.6 Visualization Results 102
  5.6.1 Choice of Parameters 102
5.7 Performance Analysis and Discussion 105
5.8 Discussion 106

A Pseudocode **111**
A.1 The Dopri5 Integration Scheme 111
A.2 Improved Stream Surface Algorithm 118

Bibliography **127**

Curriculum Vitae **137**

Introduction

Computers are indispensable tools in the scientific process. Their capability to process continually increasing amounts of data has enriched many fields of application. They enable modeling and simulation of complicated situation in increasing detail, and thereby allow to comfortably and cheaply test hypotheses, examine possibilities and verify assumptions. Thereby, they open new horizons and push back the frontiers of our understanding. The field of Scientific Visualization, a very active discipline of Computer Science, takes a key role in the knowledge generation process. It is in charge of generating images that further the understanding and convey meaningful aspects of numerical datasets by exploiting the enormous aptitude of human beings at comprehending and interpreting visual information.

Among the many disciplines that have profited in an enormous fashion from increased computational ability is the field of Computational Fluid Dynamics (CFD). Fluid flows play a very important part in many situations. From the flow of blood in capillary vessels to global weather dynamics, they govern many aspects of our everyday lives. The ability to simulate fluid flow on all scales allows an increased understanding and thereby control of these aspects. However, insight does not follow from the raw numerical data that is the result of such simulations. Flow visualization is hence required to provide essential tools that are tailored to the visualization needs of flow analysis. The approaches devised to extract meaningful structures from a numerical flow description have historically been classified in three main categories. Each has unique advantages and drawbacks.

The first approach centers on the generation of geometric primitives such as lines and surfaces using the trajectories of particles embedded in the flow. Many such conceptions exist, and they are quite intuitive by exploiting natural coherence of particles on the same trajectory or on neighboring trajectories. However, they are not suited to application problems with a high degree of complexity, for two main reasons. First, it is often unclear which trajectories exhibit interesting behavior that allows insightful conclusions. More importantly however, they have so far applied to three-dimensional flows on in a very limited sense. Line-type primitives, while simple and reliable to compute, suffer from issues of visual clarity. Surface primitives on the other hand have great illustrative power, but are difficult to generate numerically due to lack of an algorithm that is applicable to the intricate flow structures that prevail in modern flow applications.

Secondly, Vector Field Topology strives to interpret flows in the language of dynamical systems, thereby leveraging qualitative results from the mathematically deep and rigorous theoretical framework that has been built around them. The significance of the resulting symbolic description is therefore without question. This topological approach has proven a very reliable tool in the case of two-dimensional, stationary flows, but has as of yet failed to demonstrate the same utility in the three-dimensional case. However, the latter is much more relevant from an application point-of-view. Similarly, it has allowed for some limited treatment of time-varying flows in the context of parameter-dependent topology but cannot address typical problems in modern applications, owing to limitations of the theoretical foundation. Especially the last problem has been successfully addressed by the class of Lagrangian approaches that make use of observing the trajectories of neighboring particles. Meaningful structures are hence identified by observing the convergence or divergence of such trajectories. A major hindrance however is the enormous computational cost associated with these techniques that is effectively prohibitive for the analysis of large simulation datasets. Furthermore, the visualization potential of these methods in not fully developed.

Finally, feature-based visualization has taken a more empirical path to insightful images by relying on the identification and depiction of certain types of well-known structures or features that appear in fluid flow. In contrast to geometric and topological approaches, these methods have been applied to three-dimensional datasets successfully. They are not without shortcomings, however, as typically features are defined in terms of specific conditions on physical variables related to fluid flow. Therefore, applicability of such definitions is often limited to specific flow applications. While there are methods known to work well in many settings, successful visualization through the feature-based approach requires more intimate knowledge of the application under consideration and the available feature definitions.

In this context, the contributions of this thesis are the following: First, we develop a state-of-the-art algorithm that enables reliable and fast computation of stream surfaces [GTSS04, TGB*04], enabling surface-based geometric visualization in three-dimensional datasets (Chapter 3). We demonstrate the expressive and illustrative power of the resulting flow visualizations that can be achieved with our algorithm. More importantly we show that when used as a building block it dramatically enhances advanced visualization techniques in both topological and feature-based visualization[GTSS04].

Furthermore, we extend two-dimensional topological analysis to the three-dimensional setting[GTS04] based on a core algorithm for the tracking of the topological transformations undergone by a parametric vector field in arbitrary space dimensions (Chapter 4). We show that our method, which leverages the notion of topological visualization on section planes, achieves a qualitative and quantitative visualization of complex three-dimensional flow structures[TGK*04]. Remarkably, our approach inherits the compactness and high-level abstraction of topological methods but avoids their traditional drawbacks, namely their occlusion and their lack of semantic interpretation. Moreover, we augment the visually sparse topological information by an application of volumetric visualization and show that this combination provides ideal visualization properties. We further demonstrate that application of our algorithm to time-varying three-dimensional flows enables new insights into the evolution of flow structures[TGar, GLT*06, LGD*05].

Next, we propose a novel scheme for the efficient computation of visualizations based on Lagrangian approaches (Chapter 5). Leveraging the increased performance available through our method, we study the visualization aspects of Lagrangian methods on three-dimensional datasets and demonstrate several visualization techniques that reduce the visual complexity of the resulting depictions in certain typical situations while retaining essential information about the prevalent flow structures[GGTH07].

Finally, we apply the proposed methods to challenging state-of-the-art datasets and demonstrate their practical relevance and usefulness well beyond the realm of academic examples. We emphasize the visualization of vortices and vortex breakdown. For the latter we provide a detailed discussion of the new insights gained into this intricate phenomenon through the use of our novel visualization techniques.

Structure of the text. The basic notions of both modeling and simulation of fluid flows are first introduced in Chapter 1. To embed this thesis in a proper context, we discuss fundamental properties and numerical treatment of the underlying Navier-Stokes equation and give an overview of terminology specific to this setting, followed by a description of relevant flow visualization aspects and methods. Chapter 2 then proceeds to sketch the theoretical foundations of dynamical systems that we draw on in the remainder of this thesis. Starting with Chapter 3, we present our new results. First, we present a brief discussion of available stream surface techniques before we introduce our novel stream surface algorithm that is based on arc length integration and refined front control. Before concluding this chapter with a discussion of advanced visualization techniques built on our implementation, we provide a detailed analysis of the benefits of our novel algorithm in comparison to the previous state of the art.

We move on to present our general tracking algorithm and show how it can be applied in various settings in Chapter 4. After briefly discussing the current state of three-dimensional topological visualization, we develop the moving section plane approach and demonstrate its good visualization properties, especially in combination with volumetric methods. We conclude this chapter by using our scheme to examine the temporal evolution of vortex breakdown bubbles on several examples and provide a discussion of the novel insights that were obtained.

Next, Chapter 5 is concerned with Lagrangian approaches. After providing a brief introduction to the topic and prior work, we devise an adaptive approximation scheme and demonstrate that it allows to significantly reduce the computational cost typically associated with this type of method. We go on to discuss general visualization aspects of Lagrangian methods and explore novel visualization approaches based on our efficient algorithm. We conclude each chapter with a brief discussion of possible future work.

Finally, we present pseudocode for some of the algorithms developed in this thesis in Appendix A.

Chapter 1

A Brief Introduction to Flow Visualization

Fluid flows are essential objects of study in a broad range of scientific, engineering, and medical applications. In particular, the optimization of numerous industrial processes requires the precise understanding and the control of flows. Examples include important application areas such as combustion, automotive industry, aeronautics. The complexity of the considered flows has proven a challenge to computational ability, and only recently has increased computing power paved the way for numerical experiments

In this section, we will discuss some of the basic concepts underlying the modeling and simulation of technical flows and discuss a number of technical aspects that are encountered when dealing with simulation output. Furthermore, we present a short introduction to both feature- and topology-based flow visualization, before we conclude this chapter with a description of the example datasets used throughout this thesis.

1.1 Modeling and Simulation

This complexity of fluid flow is an inherent property of the Navier-Stokes equation that is the basis for modeling and studying of flow physics in many application areas. In this chapter, we will provide a brief empirical introduction to flow simulation based on the Navier-Stokes equation. The topic is so broad that we will only touch the surface to embed this thesis into a proper context. For a more comprehensive presentation, we refer the reader to [CM04, Lug96]. Furthermore, we will discuss several practical aspects that appear relevant to the understanding of the topics discussed in this thesis.

1.1.1 The Navier-Stokes Equation

The *Navier-Stokes equations* govern the motion of fluid substances such as liquids or gases in the subsonic regime. Essentially, these equations state that the motion of the fluid is governed by Newton's second law, i.e. the changes in momentum in an infinitesimal flow volume are the sum of viscous forces, changes in pressure, and other forces acting inside the volume.

1.1.2 Basic Equation

In their basic form, the Navier-Stokes equations are partial differential equations. As opposed to classical mechanics, they describe not position but rather velocity. Therefore, solutions to these equations are called *flow fields* or *velocity fields*, and other quantities of interest such as e.g. the path of a particle may be derived.

Assuming that the fluid is a continuum, the equations describing the velocity v take the following form:

$$\rho\frac{Dv}{Dt} = -\nabla p + \nabla \cdot T + f \tag{1.1}$$

The right hand side of this equation is a sum of body forces that contribute to the change of momentum ρv in a volume element moving with the flow, represented by the *substantial* or *convective* derivative

$$\frac{Dv}{Dt} = \frac{\partial v}{\partial t} + v \cdot \nabla v. \tag{1.2}$$

The forces acting on a volume element are the pressure gradient ∇p, which induces *normal stress* (acting perpendicular to the boundary of the volume element and inducing compression or tension), the *shear forces* $\nabla \cdot \mathcal{T}$ (acting tangential to the boundary), where $\mathcal{T}$ is a second order tensor, and other forces, such as for example gravitation. Besides conservation of momentum, conservation of mass is imposed by the additional equation

$$\frac{\partial \rho}{\partial t} + \operatorname{div}\,(\rho v) = 0. \tag{1.3}$$

Equations (1.1) and (1.3) are very general and allow a modeling of many fluids. They are however rarely applied in this form. Rather, specific assumptions on a fluid are incorporated. For example, the term $\nabla \cdot \mathcal{T}$ contains too many unknowns. For the class of so-called *Newtonian* fluids, of which water is a representative, it is replaced by a term $\mu \Delta v$. This *viscosity* term is assumed a linear diffusion of momentum due to friction of neighboring fluid elements. μ is the corresponding proportionality constant. Furthermore, the vast majority of simulation of flows is performed under an *incompressible flow* assumption, i.e. ρ is constant. In this case, Equation 1.3 takes the simpler form

$$\operatorname{div}\, v = 0 \tag{1.4}$$

Interestingly, this assumption typically holds even for compressible substances, such as air at room temperature, allowing a treatment of many application problems as incompressible. In addition, further conservation properties are often necessary to correctly model specific flows. For example, if the temperature of a fluid may vary, conservation of energy must be imposed.

The Navier-Stokes equations are nonlinear partial differential equations. This nonlinearity makes most problems difficult or impossible to solve analytically. It is imposed by the *convective acceleration* term $v \cdot \nabla v$, which is an acceleration associated with the change in velocity over position. Therefore, any flow that contains convection is nonlinear. Furthermore, the time-dependent chaotic behavior seen in many fluid flows is called *turbulence*. Its manifestation is a consequence of nonlinearity, and generally related to the inertia of the fluid (the left hand side of Equation 1.1); hence, flows where inertial effects are small tend to be *laminar*, that is, they do no contain turbulence. For any given problem, the appearance of turbulence can be judged by the *Reynolds number* that denotes the ratio of inertial forces to viscous forces. It is typically given as

$$Re = \frac{\rho v_s^2 / L}{\mu v_s / L^2} = \frac{\rho v_s L}{\mu}, \tag{1.5}$$

where ρ is density, v_s is the mean fluid velocity, L is the *characteristic length*, for example the streamwise length of an object embedded in a flow, and μ is the (*dynamic*) viscosity. The quantity $\frac{L}{v_s}$ is also called *characteristic time*. A flow is *laminar* at low Reynolds numbers, where viscous forces dominate, and is characterized by smooth, constant fluid motion. On the other hand, turbulence appears at high Reynolds numbers. It is dominated by inertial forces and contains random eddies, vortices and other fluctuations on short time scales. For typical problems, the transition from laminar to turbulent flow is described by a critical Reynolds value Re_{crit}. For example, for a flow through a cylindric pipe, the critical value is 2300. In the case of the delta wing configuration we will examine later, it is on the order of 10^5 to 10^6. Note that the transition from laminar to turbulent is not sudden but gradual.

In numerical experiment, one usually considers a domain of finite size. In this case, as for any partial differential equation, *boundary conditions* must be prescribed on the boundaries of the domain to fully determine a solution. There are various types of such conditions that apply to the Navier-Stokes equations, and they are usually chosen depending on the problem at hand. Of special significance is the so-called *no-slip* condition, imposed on the boundaries of solid objects in the flow, where it states that the tangential component of the velocity relative to the boundary must be zero due to friction. Obviously, there can be no flow through the boundary of a solid object, hence, the velocity vanishes completely at such boundaries.

1.1.3 Numerical Representation

To perform numerical simulation of flows along the lines sketched out above, a discretization of all quantities involved is required. There are many approaches to this purpose with differing properties, see [QV97]. An application of these general principles to flow simulation is called *Computational Fluid Dynamics* (CFD). Common to most methods is the use of a *computational grid*, that is a set of points connected to form a domain partition into cells. Depending on the chosen method, these cells can have any shape. The quantities of interest are then given on a per point, per face or per cell basis. To facilitate a continuous representation, the grid is typically endowed with an interpolant, giving rise to *vector fields* for such variables as velocity, or *scalar fields* for simple scalar variables such as pressure or density.

In this thesis, we will be concerned with *unstructured grids* that are composed of simple geometric cells such as triangles, quadrilaterals, tetrahedra, prisms, pyramids and hexahedra, and the simulation variables such as velocity or pressure are given at the points of the grid. Then, the interpolant is constructed per cell and is linear, bilinear or trilinear, depending on the cell type. We have detailed the construction of such an interpolant for purely tetrahedral grids in Section 4.3.1.

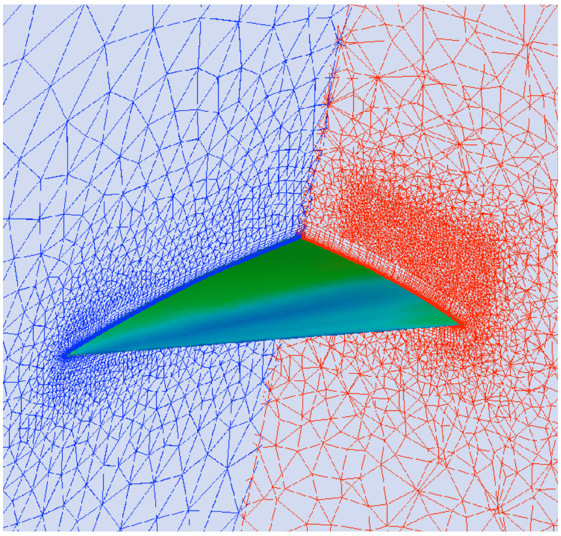


Figure 1.1: Cross-sections through an adaptive unstructured computational grid.

To perform global interpolation on such a discretized domain, it is necessary to locate the grid cell that the interpolation point is contained in. This problem is called *point location*, and several methods exist to solve it efficiently. In this thesis, we make use of the FAnToM visualization system that incorporates the point location algorithm given in [LST03]. The latter is efficient and applicable to large grids and makes use of a hierarchical binary partition of space to quickly locate the correct cell.

In modern simulations, computational grids are typically *adaptive* in resolution to save on computational effort. Instead of fully resolving the domain of interest with a very fine grid, the resolution is locally controlled by several criteria that describe the quality of the solution, which is expected to improve as the resolution is increased. Typical criteria include gradient magnitudes of velocity and pressure. Figure 1.1 illustrates an adaptive grid with a strongly increased resolution towards the wing.

1.1.4 Lagrangian and Eulerian Perspectives

Patterns produced by the movement of fluids are different from static patterns found in many other contexts, as particles do not have a permanent place but move. The paths of the particles form *flow patterns*. Contained in this notion is the idea that fluid motion can be described in two ways.

The first approach of describing moving individual fluid elements by following the paths of individual particles is called *Lagrangian*. Flow properties are interpreted as adherent to fluid elements while they move. On the other hand, the *Eulerian* perspective describes these flow properties in a fixed, pre-imposed reference frame. This duality of descriptions is expressed in the difference between the spatial derivative of a quantity and its convective derivative introduced above (Equation (1.2)). While, for some quantity q,

$$\frac{dq}{dt} = 0$$

implies that q does not change at some point over time,

$$\frac{Dq}{Dt} = 0$$

expresses the entirely different notion that q does not change on the path of a particle. The consequences of this difference in meaning are subtle but important. For example, the concept of *steady* (i.e. unchanging) flow, is quite simple to express in the Eulerian perspective as $\frac{dv}{dt} = 0$, whereas it is meaningless in the Lagrangian view. Conversely, the analysis of time-varying flows is often much simpler in the Lagrangian description.

Historically, the Eulerian perspective was widely preferred in the visualization of simulated flows. Numerical simulations typically prefer the Eulerian description since it is quite natural to discretize a given problem based on a fixed description. Furthermore, in this setting, obtaining the Lagrangian properties of a flow by computing the paths many of particles is difficult.

In this thesis, we will make use of both concepts where appropriate. Chapter 4 is concerned with the topological analysis of stationary flows and is inherently based on the Eulerian viewpoint. Conversely, Chapter 5 makes use of the Lagrangian description to allow an analysis of time-dependent flow fields.

1.1.5 Flow Features

Flow features allow analysis of flows in qualitative and high-level terms, representing a skeleton upon which the entire flow is based. The most prominent examples of features in fluid flow applications include vortices, separation and attachment lines, shock waves and recirculation zones. We will briefly discuss two feature types that we will refer to in this thesis.

Vortices A *vortex* is typically defined as the rotation of fluid elements around a common center (cf. [Lug96]). These fluid elements cover a finite space, hence a vortex is a macroscopic of large-scale flow structure in contrast to the movement of individual fluid elements. The rotating region is typically called *vortex core*, and the center of common rotation is often called *vortex core line*. Vortices exist on many scales, ranging from large vortices such as those encountered in weather systems to smallest-scale short lived eddies occurring in turbulence. While the intuitive conception of a vortex is quite clear, there is a lack of analytic description that accommodates all types of vortices encountered in various situations [Lug79]. However, vortices exhibit a number of properties, such as low pressure in the center of rotation, that are often used to define them empirically (see Section 1.2.1).

In technical flows, vortices are of paramount importance since they are responsible for a great number of both desired and undesired phenomena, such as material transport, mixing, noise, drag and lift. Therefore, they have been of interest in visualization from the start, and we briefly survey visualization methods below in Section 1.2.

The related phenomenon of *vortex breakdown* that designates a sudden loss of coherent vortical motion has received much attention in recent years (see for example [Rüt05]). In the same sense that vortices are important in technical flows, the same is true for vortex breakdown, and its disruptive nature can cause significant problems for such applications that rely on the existence of vortices. As for vortices, no analytic description of vortex breakdown exists per se. We will however describe such a case in detail in Section 1.4.1.

Recirculation The term *recirculation region* or *recirculation zone* typically refers to a flow region that is almost entirely isolated from the rest of the flow. Due to this isolation, particles do not leave such a region easily, hence, particles often circulate throughout it. A particular example is the *vortex ring*, where the circulation pattern takes the form of a ring-shaped, closed vortex.

Separation and Attachment Separation and attachment lines are another major type of feature. They are defined as the lines along which the flow attaches or separates from the surface of an embedded object. This phenomenon is induced by viscous effects that take place in direct proximity of the object. The no-slip boundary condition implies that the velocity magnitude goes to zero as one approaches the surface along a normal direction. Therefore, such lines are not described in terms of the velocity field, but instead rely on the so-called *shear stress* vector field defined on an object boundary. The connection is based on the observation that the shear stress field exhibits the same flow patterns as nearby located streamlines. In particular, flow separation and attachment are reflected in the existence of curves asymptotic particle convergence. The corresponding three-dimensional flow pattern is characterized by the presence of a stream surface (see also Chapter 3) starting or ending along the feature line that, on the other hand, swirls around a nearby located vortex. As a matter of fact, flow separation and vortex genesis are two closely related phenomena. Figure 1.2 provides an example of such a flow configuration.

Due to their skeletonic quality, flow features are ideally suited to visualization purposes, as we will discuss in the next section.

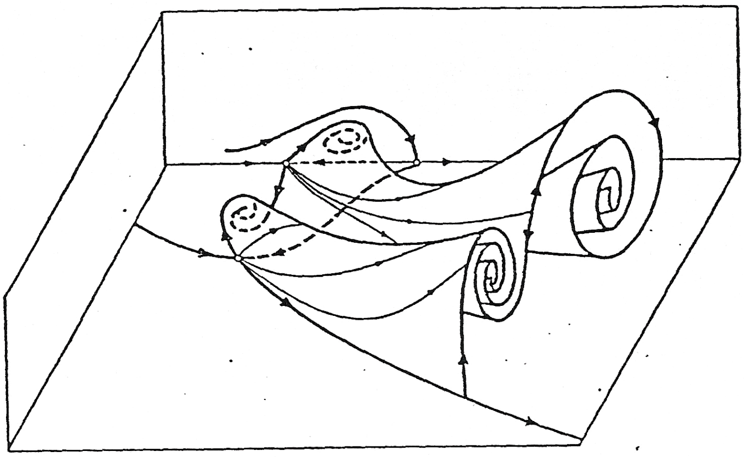


Figure 1.2: Stream surface starting on a separation line and rolling up into a vortex (reproduced from [Dal83a])

1.2 Feature-Based Visualization

The goal of feature-based visualization methods is to generate images that restrict the depiction of complex flow fields to a limited set of points, lines, surfaces and volumes representing features of particular interest for the considered application. This yields schematic pictures that convey significant flow properties in a concise and compact form.

Common to most features is a loosely empiric definition, sometimes even depending on the specific application of study. Concerning visualization, this has resulted in a variety of algorithms available to locate, identify, and visualize them.

This places a feature-based visualization user in the unfortunate position to experimentally determine which method is best suited for the needs of his particular application. Further restrictions on the type of method can be imposed by the size or the structure of the data.

We will next present a brief overview of several commonly used methods for feature-based visualization. For a more general survey of the field, refer to [PVH*03].

1.2.1 Vortices

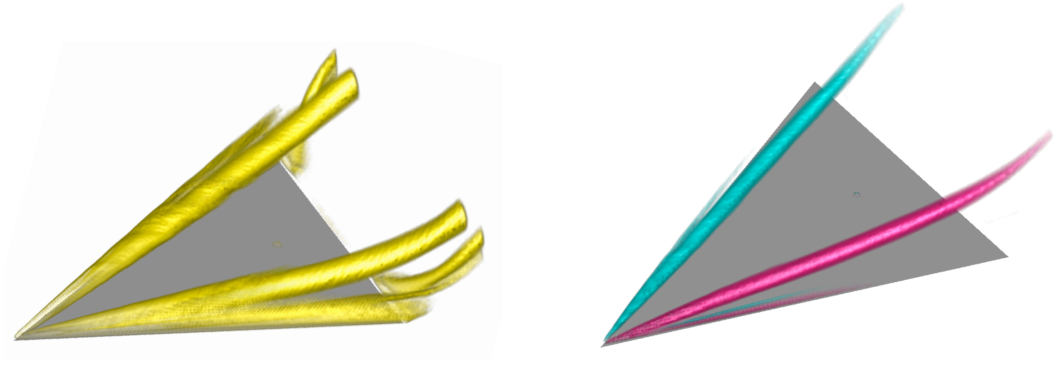
Vortex extraction and visualization methods are essentially characterized by the type of ad-hoc vortex criterion they are built on. These criteria are either region-based (identifying regions of vortical flow behavior) or rely on a line-type description (focusing on the vortical axis or vortex core line), and are typically defined for steady flow. Region definitions include high vorticity ($\operatorname{curl} v$), helicity ($\operatorname{curl} v \cdot v$) and low pressure. Most often used in engineering applications is the λ_2-definition by Jeong and Hussain [JH95]. The physical meaning behind this method is a similarity measure of the local flow structure to that induced by a pressure minimum. The major limitation of λ_2, however, is a failure to isolate closely neighboring individual vortices.

Among the line-type definitions, the approach of Sujudi and Haimes [SH95] is most widely used. The idea here is to perform pattern matching of a rotational motion against the vector field a cell-wise basis and locally extract linear sections of the rotation axis. The resulting line segments can be patched together to approximate the vortex core line. Because of the linear nature of the sought pattern, the method has issues with vortex core lines that are strongly curved. Roth and Peikert proposed a higher-order scheme that can extract curved core lines reliably [RP98]. They also showed in a subsequent paper that this and other similar methods can be formulated in a unified framework involving their *parallel operator* [PR00].

In applying such vortex feature definitions, one is often concerned with *Galilean invariance*, i.e. invariance under a constantly moving frame of reference. While those criteria that rely on derivative quantites are Galilean invariant by definition (see [SWH05]), the criteria by Sujudi/Haimes and Roth/Peikert are not. Some of the difficulties in this context arise from the use of an Eulerian description in most methods. Refer also to the excellent discussion in [Hal05].

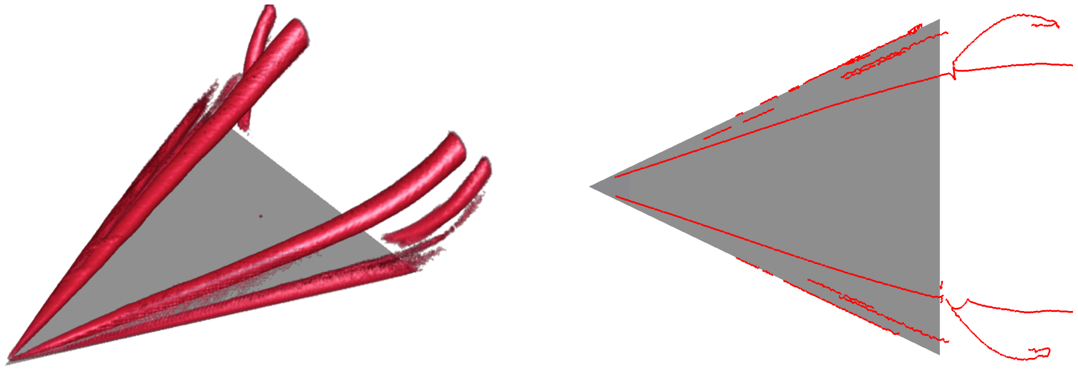
1.2.2 Separation and Attachment Lines

Following the original idea of Sujudi and Haimes for vortex core lines, Kenwright et al. proposed a simple and fast pattern matching method for the extraction of separation and attachment lines directly from the shear stress vector field [Ken98]. Their basic observation is that these feature lines are present in two linear patterns,



(a) curl v

(b) helicity



(c) λ_2

(d) Sujudi-Haimes

Figure 1.3: Vortex feature definitions and resulting visualizations.

namely saddle points and nodes (see section 4). The original method works on a cell-wise basis and extract these pattern within each triangle. Since the shear stress field is hard to compute in general and subject to numerical noise due to numerical derivation, this scheme does not perform well in many cases. Consequently strong pre-smoothing of the data is often necessary which in turn can deform and shift the features. Another approach was proposed earlier by Okada and Kao [OK97] who extend the classical *Line Integral Convolution* (LIC) algorithm [CL93] by color coding the flow direction so as to highlight the fast changes in flow direction that occur as streamlines approach separation respectively attachment lines. The weakness of this approach lies in the heavy computation associated with LIC on one hand, and in the fact that the geometry of the feature lines is not extracted. Instead, the method computes a density function that indicates the proximity / likelihood of these feature lines.

Using a different approach, Tricoche et al. recently proposed a scheme [TGS05] designed to overcome the restrictions imposed by the purely local analysis used in the algorithms mentioned previously. Their method is Lagrangian in nature by explicitly monitoring particle convergence in the shear stress vector field.

Recently, Surana et al.[SJH05] have taken a more topological approach to the extraction of separation and attachment lines. It is based on an identification of such lines as stable and unstable manifolds of shear stress critical points (see also Section 2).

1.3 Topology-Based Visualization

Vector field topology is a powerful approach for the visualization of planar flows. Topology-based methods leverage the basic results of the qualitative theory of dynamical systems to generate effective depictions characterized by a high level of abstraction and an accurate segmentation of the domain in regions where the flow exhibit a uniform behavior. Formally, this classification is defined with respect to the limit sets of the streamlines. Additionally, parameter-dependent topology and the notion of bifurcation can be used to extend this technique to time-dependent flows and account for the structural transformations that their topology undergoes over time. We study the theoretical concepts underlying these methods in Chapter 4. Furthermore, for a comprehensive survey of topological visualization methods, we point the reader at Handbook in [LHZP05].

Unfortunately, the application of this methodology to three-dimensional problems has not demonstrated the same usefulness in visualization applications so far. One explanation is the intricacy of the resulting pictures: the topology of volume flows involve stream surfaces that are plagued by self-occlusion and visual clutter. Another problem concerns the lack of intuitive connection between topological structures and major features of interest in fluid dynamics problems, as described in the previous section. Neither vortices nor separation lines are topological elements of the flow velocity vector field. Thus topology-based methods fail to extract them properly. We will further discuss these issues in Chapter 4 where they overlap with our own work.

1.4 Datasets

In the following, we will present the datasets that were used test and exemplify the visualization methods developed throughout this thesis.

The delta wing datasets, as well as the rotating lid cylinder and the high-speed train were made available to us by Markus Rütten at DLR Göttingen.

1.4.1 Delta Wing

The so-called delta wing dataset focuses on the study of the transient flow above a delta wing (with a triangular wing shape) at low speeds and increasing angle of attack. The flow over the delta wing is a vortex-dominated flow field. The wing owes much of the lift that it is able to generate at subsonic speeds to the fact that the flow separates at the leading edge of the wing, and this separated shear layer rolls into a large *primary* vortex over the leeward side of the wing (cf. Figure 3.7 on p. 56). These primary vortices exhibit a large axial flow component, resulting in a region of low pressure that generates lift on the wing. At high angles of attack, vortex breakdown occurs results in the destruction of the coherent primary vortex. The core becomes highly turbulent, the diameter of the vortex core increases, and the high axial velocity in the core ceases to exist. It is this last item that is especially important, because with the loss of axial velocity the wing loses lift. This is especially troubling if the breakdown is asymmetric, i.e. occurring on one side of the wing only, as the resulting force imbalance between the two wings can cause strong roll movements. Even if the breakdown is symmetric, the loss of a large portion of the aircrafts causes control problems. For these reasons vortex breakdown is being studied intensively. Several theories exist as to the cause of vortex breakdown, however, none has been confirmed to this point[Rüt05].

The main goal of the simulation was to numerically investigate the cause of vortex breakdown that the primary vortices exhibit. The simulation features a time-varying adaptive grid. Unfortunately, due to its enormous size, only three non-successive time steps were available to us. Therefore, we are limited to stationary considerations. The computational grid consists of about 15 million unstructured cells and poses a significant challenge from a visualization perspective.

1.4.2 Rotating Lid Cylinder

The cylinder dataset is a model dataset for vortex breakdown analysis. It consists of a closed cylinder with a spinning bottom lid that generates a vortex on the central axis of the cylinder through viscous effects. In such configurations, the flow behavior is determined by two parameters: the heigh-to-radius ratio $\frac{H}{R}$, and the rotation Reynolds number $\frac{\omega R^2}{\nu}$, where ω denotes the angular velocity of the rotating lid. Escudier[Esc84] derived an analysis from physical experiment that describes several possible configurations of vortex breakdown in the steady state depending on the Reynolds number (see Figure 1.4).

The time-varying boundary condition for the rotating lid was chosen such that the Reynolds number increases from a value of 1200 to 3100, and several breakdown bubbles appear and disappear during the course of the time-varying simulation in accordance with Figure 1.4. The dataset is very well resolved. The computational grid consists of 752.000 cells, and the temporal evolution encompasses 5000 timesteps.

1.4.3 High-Speed Train

This dataset is the result of a stationary simulation of a high-speed train traveling at at velocity of 250 km/h with wind blowing form the side at an angle of 30 degrees. Empirically, the wind causes vortices to form on the leeward side of

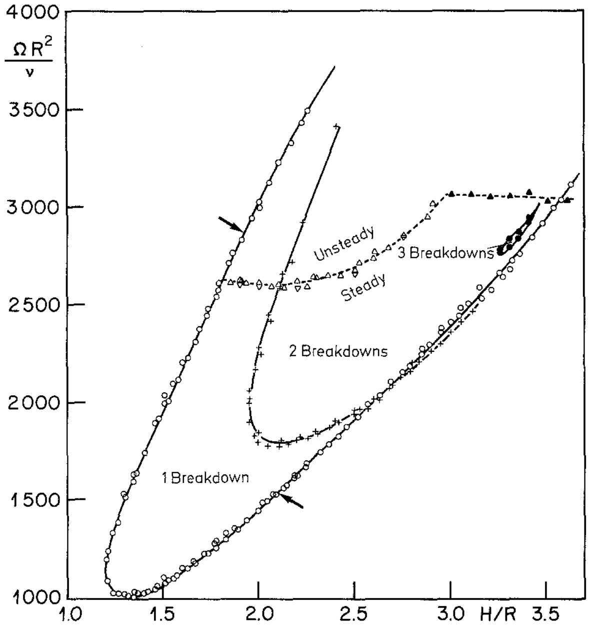


Figure 1.4: Vortex breakdown configurations in the rotating lid cylinder (reproduced from [Esc84]).

the train. The resulting drop of pressure on this side in combination with the increased pressure on the windward side create a pressure differential that adversely affects the train's track holding. The computational grid of contains 2.6 million unstructured elements.

1.4.4 Kármán vortex street

The *Kármán Vortex Street* (cf. [Lug96]) is one of the most widely known patterns in fluid mechanics. It consists of a vortex street behind a cylinder and is a special case of unsteady flow separation from bluff bodies embedded in the flow. It is quite well understood and therefore an ideal test case for many applications. We have computed this dataset for test purposes using the *Gerris Flow Solver* (described in [Pop03]).

Chapter 2

Vector Fields and Dynamical Systems

This chapter is devoted to the theoretical foundations of vector field topology as it is defined and used in Scientific Visualization, and particular aspects of its numerical treatment.

Vector field topology is based on the notion of phase portraits of a dynamical system and enables a geometric interpretation of vector fields. Critical points, closed orbits and their connecting manifolds play a key role. Therefore, we present a qualitative analysis and study some of their properties. Dependence of the studied vector field on a parameter induces further concepts of structural stability and bifurcations that naturally describe qualitative changes with respect to a change of parameter.

In the following, we will introduce and discuss those concepts that are relevant to the remainder of this thesis, and point out important results. We assume that the reader is familiar with basic concepts of geometry and analysis of several variables. For a broad treatment of these topics, including proofs, we refer the reader to [And73, GH83, ZDHD92]. Furthermore, we will broach the subject of numerical treatment of dynamical systems. We limit our presentation to Euclidean space $\mathbb{R}^n$, since it is the case that applies to the remainder of this thesis, but remark that most of the concepts below generalize to the manifold case.

2.1 Basic Definitions and Fundamental Properties

In this section, the notions of dynamical system, flow and phase portrait related to a vector field are defined and the fundamental theorems ensuring the existence and uniqueness of the solution to the Cauchy problem are given. In the following, let $I \subset \mathbb{R}$ and $\Omega \subset \mathbb{R}^n$ open.

Definition 2.1. *A* vector field *is a vector-valued function*

$$v(t,x) : I \times \Omega \longrightarrow I\!R^n.$$

The vector field is called continuous resp. differentiable of order k) if v is continuous resp. of class $C^k(I \times \Omega)$.

We label a vector field *stationary* if v does not depend on t, or *time-dependent* to explicitly state a dependence on t. In the first case, we will simply write $v(x)$.

Definition 2.2. *An* ordinary differential equation *of first order associated with a* vector field v *is a vector-valued function y that satisfies*

$$\frac{dy}{dt} = v(t, y(t)). \tag{2.1}$$

y is of the same dimension as v.

Introducing an initial condition to Equation (2.1) leads to the *Cauchy problem*:

Definition 2.3. *A differentiable function y is a solution of the* Cauchy initial value problem *if it fulfills Equation* (2.1) *and additionally satisfies the* initial condition

$$y(t_0) = y_0. \tag{2.2}$$

for $(t_0, y_0) \in I \times \Omega$.

The evolution of points in Ω as solutions of Equation (2.1) is described by the flow generated by v.

Definition 2.4. *Let $t \in I\!R$, then the vector field v generates a* flow, *that is a differentiable function*

$$\phi_t(t_0, x_0) : I \times \Omega \longrightarrow \Omega,$$

satisfying

$$\left. \frac{d}{dt}\phi_t(t_0, x_0) \right|_{t=\tau} = v(\tau, \phi_\tau(t_0, x_0)) \qquad \forall (t_0, x_0) \in I \times \Omega \tag{2.3}$$

ϕ has an intuitive interpretation: $\phi(t,x)$ is the new position of a particle located at $(t,x) \in I \times \Omega$ after time t subject to the flow ϕ.

Definition 2.5. *A* dynamical system *is a group ϕ acting on a space Ω. That is, there is a set of maps*

$$\phi_t : I \times \Omega \to \Omega, \qquad t \in I\!R$$

such that ϕ_t is a continuous function and the group properties are fulfilled:

1. $\phi_0(t_0, x_0) = x_0 \quad \forall (t_0, x_0) \in I \times \Omega$

2. *If* $t_0{+}t, t_0{+}s, t_0{+}t{+}s \in I$, *then* $\phi_t(t_0{+}s, \phi_s(t_0, x_0)) = \phi_{t+s}(t_0, x_0) \quad \forall (t_0, x_0) \in \Omega$.

The set Ω is called the *phase space* of the dynamical system. It is immediate that a vector field v induces a dynamical system ϕ by means of its flow. Conversely, if ϕ is well-defined, then v is obtained by Equation (2.4).

If $\phi \sim \mathbb{R}$, then the dynamical system is called *continuous*, or *discrete* if $\phi \sim \mathbb{Z}$. Furthermore, a dynamical system is called *autonomous* if $v(t, x)$ is stationary (or equivalently constant in t), and *non-autonomous* otherwise. Finally, it is called *global* if t can vary in $\mathbb{R}$.

Remark. *For an autonomous dynamical system,* $\phi_t(t_0, x_0)$ *does not depend on* t_0*. We will therefore abbreviate* $\phi_t(x_0)$ *in such cases.*

With Definition 2.3, the map

$$\phi_t(t_0, x_0) : t \mapsto \phi(x, t)$$

is a solution of the Cauchy problem with initial condition

$$\phi_x(0) = x.$$

This solution is called *integral curve* or *trajectory* through x. The set of points

$$\gamma_x := \{\phi_x(t) : t \in \mathbb{R}\}$$

is called *orbit* of x in the phase space Ω. Remark that these definitions are often applied directly in the context of vector fields, where the dynamical system is implied by the vector field.

Definition 2.6. *The set of all integral curves of Equation* (2.1) *as subsets of* $I \times \Omega$ *is called* phase portrait *of the dynamical system.*

We will now investigate under which conditions a solution to the Cauchy problem for ϕ_t can be found.

2.2 Existence and Uniqueness

The necessary conditions for existence and uniqueness of ϕ_t relies on the notion of Lipschitz continuity. In the following, let E and F metric vector spaces with associated norms $||\cdot||_E$ and $||\cdot||_F$.

Definition 2.7. *A function* $f : V \subseteq E \to F$ *is* Lipschitz *on* V *if there exists a positive constant* $K > 0$ *such that*

$$||f(x) - f(y)||_F \leq K||x - y||_E \qquad \forall x, y \in \Omega$$

K *is called* Lipschitz constant *of* f.

Essentially, a Lipschitz function is limited in how fast it can change; a line joining any two points on the graph of this function will never have a slope steeper than the Lipschitz constant.

Definition 2.8. *The function $f : V \subseteq E \longrightarrow F$, is* locally Lipschitz *for every $x \in V$ if there exists a neighborhood $U(x)$ such that the restriction $f|_U$ is Lipschitz.*

If f is (locally) Lipschitz, it immediately follows that f is continuous. We are now equipped to state the fundamental theorem on the local existence and uniqueness of the Cauchy problem.

Theorem 2.9. *Let v a vector field and $(t, x) \in I \times \Omega$. If v is Lipschitz with respect to its second argument there exist $\varepsilon > 0$ and a unique solution*

$$y : \ (t - \varepsilon, t + \varepsilon) \ \longrightarrow \ \Omega \tag{2.4}$$

to the Cauchy problem with initial condition $y(t) = x$.

Theorem 2.9 states the existence of a local neighborhood around t on which the trajectory starting at (t, x) exists and is uniquely determined. Under specific circumstances, they can be continued to a larger open interval.

Theorem 2.10. (Picard-Lindelöf) *Let v as before and suppose that v is continuous and locally Lipschitz with respect to its second argument on Ω. Then the local solution Equation* (2.4) *of the Cauchy problem can be uniquely extended to the boundary of $I \times \Omega$.*

Corollary 2.11. *Let $M \subset \Omega$ a compact set and $(v, \mathbb{R} \times \Omega)$ a continuous vector field. If v is locally Lipschitz with respect to its second argument on $\mathbb{R}^n$, then either the solution to the Cauchy problem Equation* (2.4) *is unbounded, or it exists on the interval $(-\infty, +\infty)$.*

In other words, in a global dynamical system a trajectory leaves M or stays in M forever. Thus, the Cauchy problem is uniquely solvable for vector fields defined on compact sets, which is a typical case in practice, and existence of a maximal solution is assured. The next theorem is concerned with the dependence of trajectories on the initial condition.

Theorem 2.12. *Let v a continuous vector field that satisfies a Lipschitz condition with respect to its second argument on Ω. Then there exists $\varepsilon > 0$ such that $\phi_t(x)$ exists uniquely for $t \in (-\varepsilon, \varepsilon)$ and is continuous with respect to x.*

Remark. *In general, ϕ_t is not continuous with respect to x for arbitrary t.*

2.3 Orbits and Invariant Sets

From this point on, we will only consider autonomous systems, since some of the concepts we will define next are not applicable to non-autonomous systems.

Definition 2.13. *Let $x \in \Omega$, then define the set of points*

$$\gamma(x) := \{\phi_t(x) : t \in \mathbb{R}\} \subseteq \Omega$$

as the orbit of x. Furthermore, define the positive and negative semi-orbit *by*

$$\gamma_+(x) := \left\{\phi_t(x) : t \in \mathbb{R}^+\right\} \subseteq \Omega$$

and

$$\gamma_-(x) := \left\{\phi_t(x) : t \in \mathbb{R}^-\right\} \subseteq \Omega$$

Thus, an orbit is the set of points that are visited by the trajectory through x. Remark that $y \in \gamma(x)$ implies $y = \phi_t(x)$ for some t and hence $\gamma(x) = \gamma(y)$. In particular, different orbits are disjoint.

Orbits are classified using the following definitions into fixed, periodic, and non-periodic orbits.

Definition 2.14. *If $\gamma(x) = \{x\}$, then x is called a* fixed point *of ϕ.*

Equivalently, we find the following definition for the vector field v associated to ϕ.

Definition 2.15. *A* critical point *x_0 of a vector field v is characterized by*

$$v(x_0) = 0$$

It is also called singular point*,* singularity *or simply* zero*.*

Definition 2.16. *x is a* periodic point *of ϕ if there is some $t > 0$ such that $\phi(t, x) = x$. The lower bound of such t is called the* period *of x. The orbit $\gamma(x)$ is called* periodic orbit*.*

It is easy to see that for any periodic orbit, $\gamma_+(x) \cap \gamma_-(x) \neq \emptyset$. Therefore, they are also called *closed orbits*.

Definition 2.17. *An orbit $\gamma(x)$ is called* non-periodic *if x is neither a fixed point nor a periodic point. x is then called* regular point*.*

Definition 2.18. *The ω-*limit set *$\omega(x)$ of a point $x \in \Omega$ is a subset $U \subset \Omega$ such that for each $y \in U$ there is a sequence $t_n \to \infty$ with $\phi(t_n, x) \to y$.*

*Furthermore, the α-*limit set *$\alpha(x)$ of a point $x \in \Omega$ is a subset $U \subset \Omega$ such that for each $y \in U$ there is a sequence $t_n \to -\infty$ with $\phi(t_n, x) \to y$.*

Observe that limit sets for points on the same orbit are identical. Naturally, both fixed points and periodic orbits are limit sets.

Definition 2.19. *A set $U \subset I\!R^n$ is called* invariant *if $\gamma(x) \subseteq U$ for all $x \in U$.*

Lemma 2.20. *Let $x \in \Omega$. Then the limit sets $\alpha(x)$ and $\omega(x)$ are both closed and invariant.*

When considering limit sets, one is interested in the qualitative behavior of integral curves in a small neighborhood, which is characterized by the concept of hyperbolicity, given in the next definition.

Definition 2.21. *A compact invariant subset $C \subset \Omega$ is called a* hyperbolic set *of the dynamical system ϕ if for each point $x \in C$ there exist non-trivial subspaces of E^s_x and E^u_x for which the following properties hold:*

1. *There are constants $a, b, c > 0$ such that the inequalities*

   $$\begin{array}{lll} ||D\phi_t(x)x^s|| \leq a||x^s||e^{-ct} \quad \textit{and} \quad ||D\phi_t(x)x^s|| \geq b||x_u||e^{ct} & \textit{for } t \geq 0 \\ ||D\phi_t(x)x^s|| \geq a||x^s||e^{ct} \quad \textit{and} \quad ||D\phi_t(x)x^s|| \leq b||x_u||e^{-ct} & \textit{for } t \leq 0 \end{array}$$

   hold for any $x^s \in E^s_x$ and $x^u \in E^u_x$.

2. *The local tangent space $I\!R^n$ allows a decomposition*

   $$I\!R^n = E^s_x \oplus E^u_x \oplus E^n_x,$$

   where E^n_x is a one-dimensional subspace spanned by the vector $\frac{d\phi_t}{dt}(x)$ if it does not vanish, and $\{0\}$ otherwise.

E^s_x and E^u_x are called stable *and* unstable subspace*, respectively.*

That is, around any point of the invariant set, the behavior of trajectories parallel to x can be decomposed into convergent and divergent parts.

Definition 2.22. *A fixed point $x_0 \in \Omega$ is called* hyperbolic *if the Jacobian $D\phi_t(x_0)$ does not have eigenvalues of unit modulus.*

Remark. *The definition of hyperbolicity for fixed points is often given by mandating non-vanishing real part of the eigenvalues of $Dv(x_0)$. Since*

$$D\phi_t(x_0) = e^{tDv(x_0)},$$

these definitions coincide.

Furthermore, one introduces the concept of stability to study the long-term behavior of trajectories near limit sets. To keep consistent with the Scientific Visualization literature, we will make use of α- and ω-stability as introduced in [Sch99].

Definition 2.23. *A compact invariant set $C \subset \Omega$ is called* ω-stable *(resp.* α-stable*) if for every open set $U \supset C$ there exists an open set $U' \supset U$ such that $\gamma_+(U') \subset U$ (resp. $\gamma_-(U')$). Furthermore, if for any $x \in U'$*

$$\inf_{y \in \gamma_+(x), y' \in C} ||y - y'|| = 0 \qquad \textit{(resp.} \inf_{y \in \gamma_-(x), y' \in C} ||y - y'|| = 0 \textit{)}$$

then C is called ω-asymptotically stable *(resp.* α-asymptotically stable*).*

Stability in the sense of Definition 2.23 implies that initially close trajectories stay close to the limit set, and even converge to it under asymptotic stability. In the next section, we will see how these ideas are applied in a systematic study of fixed points of dynamical systems.

2.4 Critical Points

We deal in the following local phase portrait of fixed points. A first characterization of critical points is achieved by the following definition

Definition 2.24. *Let x_0 a critical point of a vector field v. Then x_0 is said to be of* first order *if the Jacobian matrix $Dv(x_0)$ has full rank. Otherwise, it is said to be of* higher order *or* non-linear*.*

We will consider the purely linear case first, and we will later show how the study of general first order critical points can be reduced to this case.

2.4.1 Linear Vector Fields

Linear vector fields provide a fundamental special case in that their critical points can be completely classified. The following results are taken from [HS74, pp. 82 - 96].

Definition 2.25. *A vector field v is* (affine) linear *if there exists a matrix $A \in I\!R^{n \times n}$ and a vector $b \in I\!R^n$ such that*

$$v(x) = Ax + b \quad \forall x \in I\!R^n.$$

If $b = 0$, v is called homogeneous linear *and 0 is a critical point.*

If A is invertible (i.e. $\det A \neq 0$), letting $y := A^{-1}b$, we set

$$v'(x) := v(x - y) \equiv Ax.$$

Hence, a linear vector field v is equivalent to a homogeneous linear vector field v' by a translation of the origin. Furthermore, 0 is the only critical point of v'. Since the notions of stability introduced above are topological in nature, they are not affected by a coordinate change. Without loss of generality we will limit ourselves to the homogenous linear case.

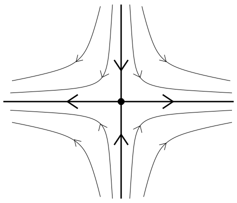
Property 2.26. *The critical points of homogeneous vector fields are characterized by the eigenvalues of their matrix* A*.*

We go on to briefly discuss the possible cases for linear vector fields on $\mathbb{R}^2$ and $\mathbb{R}^3$ as they typically appear in the Scientific Visualization literature.

Classification in $\mathbb{R}^2$

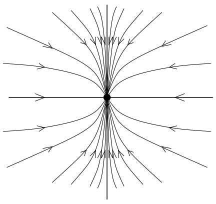
In the two-dimensional case, the matrix A has two eigenvalues $\lambda_1, \lambda_2 \in \mathbb{C}$. The possible cases are enumerated next.

- **Case 1.** A has real eigenvalues with opposite signs. The zero is called a **saddle point**.

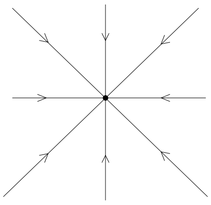


Saddle point

- **Case 2.** Both eigenvalues have negative real parts. The zero is called a *sink*, because any integral curve tends toward O for $t \longrightarrow \infty$.
  - **Case 2a.** A is diagonalizable and its eigenvalues are different. The zero is called a *node sink*. The eigenvector related to the eigenvalue with largest (resp. smallest) modulus corresponds to the direction of "fast" (resp. "slow") convergence. The special case where the eigenvalues are equal is called a *focus sink*.

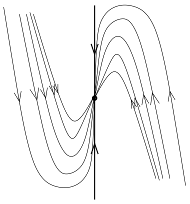


Node sink

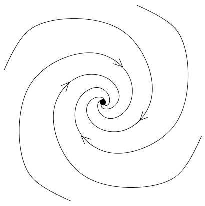


Focus sink

  - **Case 2b.** A is not diagonalizable but has one real negative eigenvalue. The zero is called an *improper node sink*.
  - **Case 2c.** A has two complex conjugate eigenvalues with negative real parts. The zero is called a *spiral sink*.

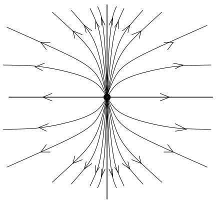


Improper node sink

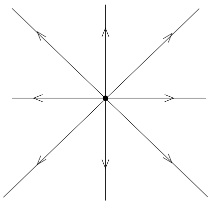


Spiral sink

- **Case 3.** Both eigenvalues have positive real parts. The zero is called a *source*, because any integral curve tends toward it for $t \longrightarrow -\infty$.
  - **Case 3a.** A is diagonalizable and its eigenvalues are different. The zero is a *node source*. If both eigenvalues are equal, the zero is a *focus source.*

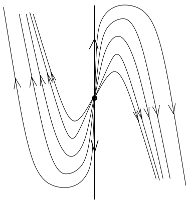


Node source

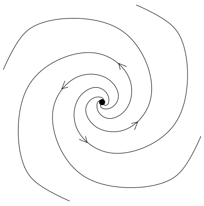


Focus Source

  - **Case 3b.** A is not diagonalizable but has one real positive eigenvalue. The zero is called an *improper node source.*
  - **Case 3c.** A has two complex conjugate eigenvalues with positive real parts. The zero is called a *spiral source.*

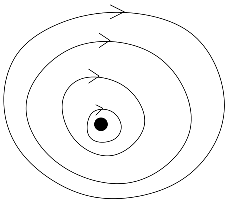


Improper node source



Spiral source

- **Case 4.** A has pure imaginary (conjugate) eigenvalues. The zero is called a *center.*



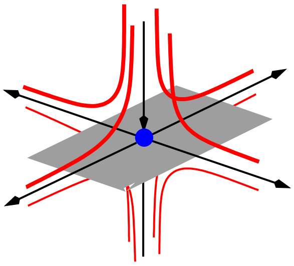
Center

Classification in $\mathbb{R}^3$

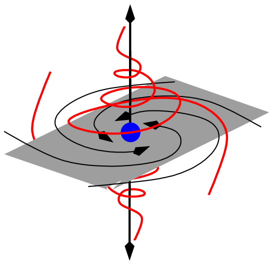
Likewise, the critical point types in $\mathbb{R}^3$ are classified as follows. We again base the classification on the (generalized) eigenvalues of A. Remark that the designations of critical point types in $\mathbb{R}^3$ vary between authors.

- **Case 1.** The real parts of the eigenvalues have mixed sign. The zero is then generally called a *saddle point* and is further labeled by the two-dimensional critical point type that corresponds to a restriction of A to the eigenspaces of the eigenvalues of same sign. The

  Examples:

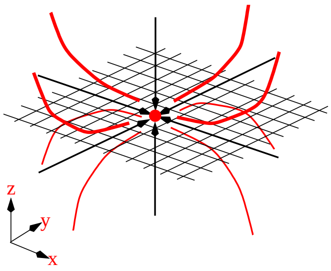


Repelling node saddle

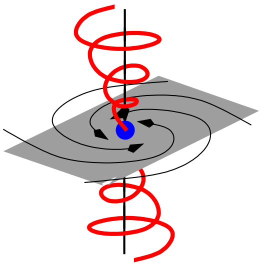


Attracting spiral saddle

- **Case 2.** All eigenvalues have negative real part. The zero is called *sink* since any integral curve approaches the critical point as $t \to \infty$.
  - **Case 2a.** A is diagonalizable. The zero is called a *node sink.*
  - **Case 2b.** A has a complex conjugate eigenpair and a real eigenvalue. The zero is called *spiral sink.*



Node sink

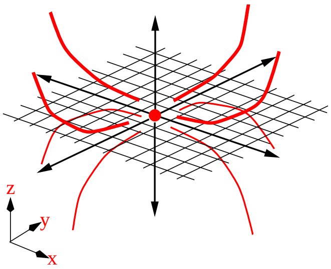


Swirl sink

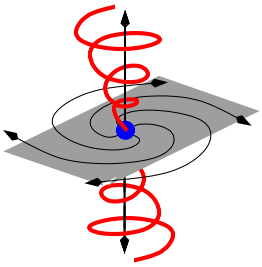
  - **Case 2c.** A is not diagonalizable. The zero is called *improper node sink*

- **Case 3.** All eigenvalues have positive real part. The zero is called *source* since any integral curve approaches the critical point as $t \to -\infty$.
  - **Case 3a.** A is diagonalizable. The zero is called a *node sink.*

- **Case 3b.** A has a complex conjugate eigenpair and a real eigenvalue. The zero is called *spiral source.*



Node source



Swirl source

- **Case 3c.** A is not diagonalizable. The zero is called *improper node source*

2.4.2 General Case

Moving away from the special case of purely linear vector fields, the study of critical points concentrates on the local behavior of the phase portrait in a small neighborhood around the considered critical point. The linearization of a vector field v around a critical point x_0 is a linear vector field

$$v'(x) := Dv(x_0)x.$$

We recall Definition 2.24 and proceed to investigate first-order critical points first. Among all possible types of such critical points, special attention is paid to sinks and sources. Their definition is based on the property of the Jacobian matrix at their location. They generalize the linear sinks and sources encountered previously.

Definition 2.27. *If all eigenvalues of the Jacobian matrix $Dv(x_0)$ have negative real parts, then the critical point x_0 is called a* sink. *Conversely, if all eigenvalues have positive real parts, x_0 is called a* source.

The intuitive meaning of this classification is stated more precisely by the following theorem.

Theorem 2.28. *Let x_0 be a sink of the vector field v with corresponding flow ϕ. Suppose there exists $c < 0$ such that every eigenvalue of $Dv(x_0)$ has real part less than c. Then there is a neighborhood U of x_0 such that*

$$||\phi(x,t) - x_0|| \leq \exp^{-tc} ||x - x_0|| \qquad \forall x \in U, t \geq 0.$$

Corollary 2.29. *With the definitions above, a sink (resp. source) is a ω- (resp. α-) stable critical point. If furthermore $Dv(x_0)$ has no eigenvalue with zero real part, it is ω- (resp. α-) asymptotically stable. Furthermore, both sinks and sources are hyperbolic in the sense of Definition 2.22.*

The following theorem provides the critical connection between the general and linear cases. It indicates the fundamental relation between trajectories of a vector field and those of its linearization.

Theorem 2.30. (Hartman-Grobman) *If a critical point x_0 of a vector field v is hyperbolic then there is a neighborhood $U \ni x_0$ and a homeomorphism $H : U \to \mathbb{R}^n$ that locally maps the integral curves of the non-linear flow ϕ induced by v to those of the corresponding linear flow. The homeomorphism preserves the direction of integral curves and can be chosen to preserve parameterization of trajectories.*

Thus, for hyperbolic critical points, the phase portrait of the integral curves in their neighborhood will be similar to the linear case. Fig. 2.1 provides an idea of this correspondence for a two-dimensional vector field. The general classification of critical points in $\mathbb{R}^2$ distinguishes two types of critical points: center and non-center types.

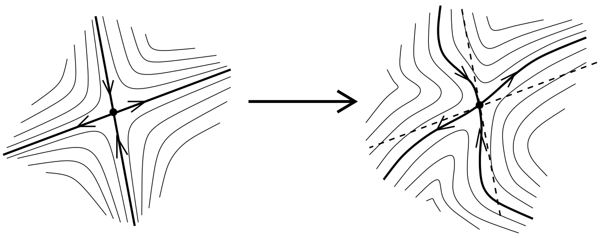


Figure 2.1: Relation between a linear and a non-linear saddle point in $\mathbb{R}^2$.

Definition 2.31. *A critical point in the plane that is not approached by any integral curve is said to be of* center *type. If on the contrary, at least one integral curve converges to it, it is of* non-center *type (see Figure 2.2).*

In the non-center case, the integral curves converging to the critical point determine so-called *sectors*. These notions are illustrated in Fig. 2.3 for two-dimensional situations. Sectors allow a classification of local integral curve behavior into three types.

Definition 2.32. *The sectors of a non-center critical point x_0 can be of three different types. Let $\varepsilon > 0$ and $x \in \partial B_\varepsilon(x_0)$, and let $\phi_t(x)$ the integral curve through x. Let*

$$x^+ := \lim_{t\to\infty} \phi_t(x) \qquad \textit{and} \qquad x^- := \lim_{t\to-\infty} \phi_t(x)$$

1. *If $x^+ \neq x_0$ and $x^- \neq x_0$, then x is contained in a* hyperbolic *or* saddle sector*. In this case, integral curves through x belong to* separatrices *of x_0.*
2. *If $x^+ = x_0$ and $x^- = x_0$, then x is contained in a* elliptic *sector.*

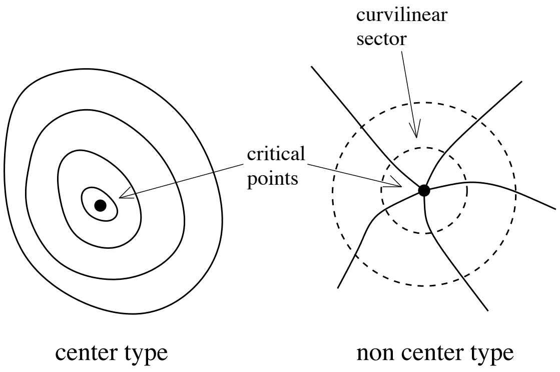
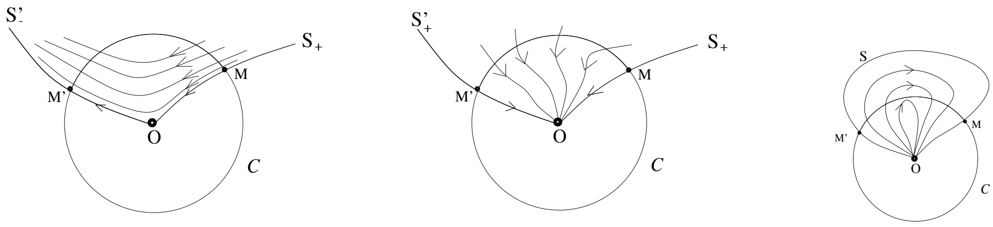


Figure 2.2: Center and non-center types in $I\!R^2$

3. *If* $x^+ = x_0$ *and* $x^- \neq x_0$ *or* $x^+ \neq x_0$ *and* $x^- = x_0$*, then* x *belongs to a* parabolic *sector.*

An illustration of the different sector types for the two-dimensional case is given in Fig. 2.3.



(a) Hyperbolic (b) Parabolic (c) Elliptic

Figure 2.3: Sector types

By considering a neighborhood with a small enough radius, one can obtain a decomposition into the sector types defined above (analytical case). The boundaries of an elliptic sector cannot be determined locally. If one restricts the study of an elliptic sector to a neighborhood of the critical point, this sector is always bounded on both sides by parabolic sectors. The set of integral curves bounding a hyperbolic sector is called *separatrix* because it separates two sets of integral curves that diverge from one another as $t \to \infty$ or $t \to -\infty$. Consequently, any singular point may be characterized by the type and geometry of its sectors. The precise meaning of this characterization is the following.

Remark. *In the special case of linear vector fields, the only critical point type presenting hyperbolic sectors and thus separatrices is the saddle point.*

Theorem 2.33. *If the structures of two singular points are related through a one-to-one correspondence between their respective separatrices converging for* $t \to \infty$*, separatrices converging for* $t \to -\infty$ *and elliptic regions then there exists a curve-preserving topological mapping of a neighborhood of the first onto a neighborhood of the second preserving orientation and direction of* t*.*

Moving away from the vicinity of a critical point to consider its influence on the global structure of the phase portrait, the notion of a basin is introduced.

Definition 2.34. *The union of all trajectories that tend toward a critical point* x_0 *as* $t \to \infty$ *is called the* ω*-basin of* x_0*. Conversely, the union of all integral curves tending toward* x_0 *as* $t \to -\infty$ *is called the* ω*-basin.*

Remark. *The* ω*-basin of a source is reduced to the source itself and the* α*-basin of a sink is reduced to the sink itself.*

Remark. *In the case of a saddle point,* α*- and* ω*-basins form manifolds that are called* unstable *and* stable *manifold, respectively, and coincide with the separatrices. In* $\mathbb{R}^2$*, both manifolds have dimension* 1*. In* $\mathbb{R}^3$*, one of these manifolds has dimension* 2 *and is frequently called a* separation surface *in Scientific Visualization.*

2.5 Topological Graph

The concepts introduced so far are the constituent parts of the structure of the phase portrait of a dynamical system, also called topological graph or simply topology. By extension, one calls it topology of the corresponding vector field. The definition used in the following is given next.

Definition 2.35. *The* topology *of a vector field* v *consists of all limit sets and separatrices of* v*.*

The essential property of a separatrix is to separate groups of integral curves that have different asymptotic behaviors. In other words, a separatrix locally divides the domain of definition of a vector field into two subdomains inside which all orbits have the same $\alpha-$ and $\omega-$limit sets. An equivalent definition consists in considering a separatrix as the intersection of the closure of two basins. An interesting problem arises when dealing with vector fields defined on compact sets. In this case, the asymptotic behavior of all integral curves is not only determined by the limit sets as subsets of the domain but also by the restriction of the vector field to the boundary of the domain: This boundary can locally act as source (where the vector field is directed inwards), sink (where the vector field is directed outwards) or saddle (separating the former two). Therefore, the generalized notion of separatrix also includes the integral curves starting at the boundary saddles. A presentation of this topic from the visualization viewpoint can be found in [SHJK98]. The previous notions are illustrated in Fig. 2.4. Remark that this topology contains no separatrix emanating from critical points.

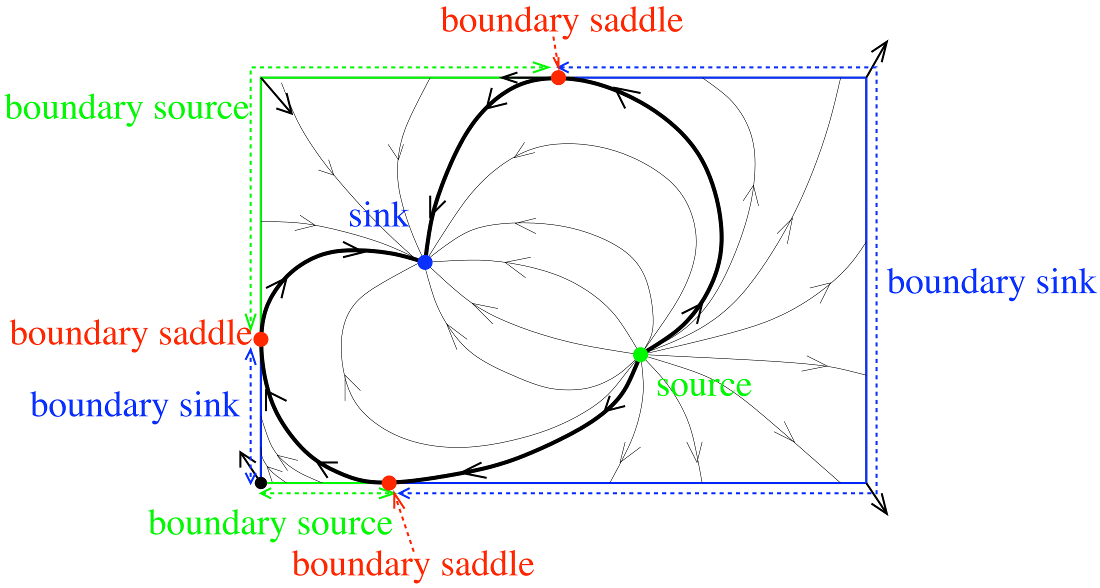


Figure 2.4: Topology of a vector field over a bounded domain

2.6 Poincaré Index

The Poincaré index is a fundamental concept that has been introduced by Poincaré himself[Poi] in the qualitative theory of dynamical systems. This notion has many theoretical and practical applications, and the basic idea behind it is the answer to the question of how many times a vector field "rotates" in the neighborhood of a point. The definitions are given next as well as fundamental theorems that will prove useful in the following. The following results are taken from [HS84] where they are presented in the language of Geometry Algebra.

In the following, let $v : \Omega \to \mathbb{R}^n$ a continuous vector field that contains only isolated critical points and C_v the set of isolated critical points of v.

Definition 2.36. *The map*

$$\Gamma : \mathbb{R}^n \backslash \{0\} \to S^{n-1}, \quad x \mapsto \frac{x}{||x||}$$

is called the Gauss map.

Simply put, Γ maps any vector to its direction. Note that Γ is idempotent, i.e. $\Gamma \circ \Gamma = \Gamma$. The winding number and index concept for hypersurfaces is defined next.

Definition 2.37. *Let $S \subset \Omega$ a closed orientable hypersurface. Define* winding numnber *of S with respect to a point $x \notin S$ by*

$$\#_x(S) := \frac{1}{vol(S^{n-1})} \int_S \Gamma(x-y) dS(y). \tag{2.5}$$

In other words, the winding number is a measure of the signed area on the unit sphere covered by the point $\Gamma(x-y)$ as y moves over all of S, relative to the area of the unit sphere, and is always an integer. Furthermore, it is immediate that the winding number is homotopically invariant, i.e. it retains its value under a continuous deformation of S.

Definition 2.38. *Let $S \subset \Omega$ a closed orientable hypersurface that does not contain a critical point of v, i.e. $S \cap C_v = \emptyset$. Then the* index *of S relative to v is defined as*

$$\operatorname{ind}_v(S) := \#_0\left(\Gamma(v|_S)\right). \tag{2.6}$$

Informally speaking, the index measures how often a vector field rotates around the origin if it is sampled on all of S is traversed. The following fundamental theorems give deeper insight into the concept.

Theorem 2.39. *Let S a closed oriented hypersurface as in the definition above. If the interior of S does not contain a critical point of v, then* $\operatorname{ind}_v(S) = 0$.

Theorem 2.40. *Let $S_1, \dots, S_m$ closed oriented hypersurfaces as above and S a closed oriented hypersurface that encloses all S_i. Moreover, S does not enclose critical points other than those enclosed by one of the S_i. Then*

$$\operatorname{ind}_v(S) = \operatorname{ind}_v(S_1) + \cdots + \operatorname{ind}_v(S_m).$$

Finally, we are able to define the index of a critical point.

Definition 2.41. *Let x_0 a critical point of v and ε and choose a closed oriented hypersurface S such that the closure of S contains only the critical point x_0, i.e. $\bar{S} \cap C_v = \{x_0\}$. Then define*

$$\operatorname{ind}_v(x_0) := \operatorname{ind}_v(S).$$

In essence, the index measures how often the vector field covers a sphere in the vicinity of a critical point. Figure 2.5 illustrates this for the two dimensional case, where S takes the form of an orientable closed curve or path. An example for a vector field on $\mathbb{R}^3$ is given in Figure 2.6.

While this conception of the index in terms of closed oriented surfaces is quite geometric in nature, a deep connection exists between the index and the degree of a map. More specifically, the wrapping number is provably equal to the degree of the map $(\Gamma(v|_S)$. The following statement is a result of this connection.

Theorem 2.42. *The index of a first-order critical point is ± 1. More specifically,*

$$\operatorname{ind}_v(x_0) = \operatorname{sign}(\det Dv(x_0)).$$

By extension, it follows that the index is equal to the sign of the product of the eigenvalues of the Jacobian eigenvalues. This immediately leads to

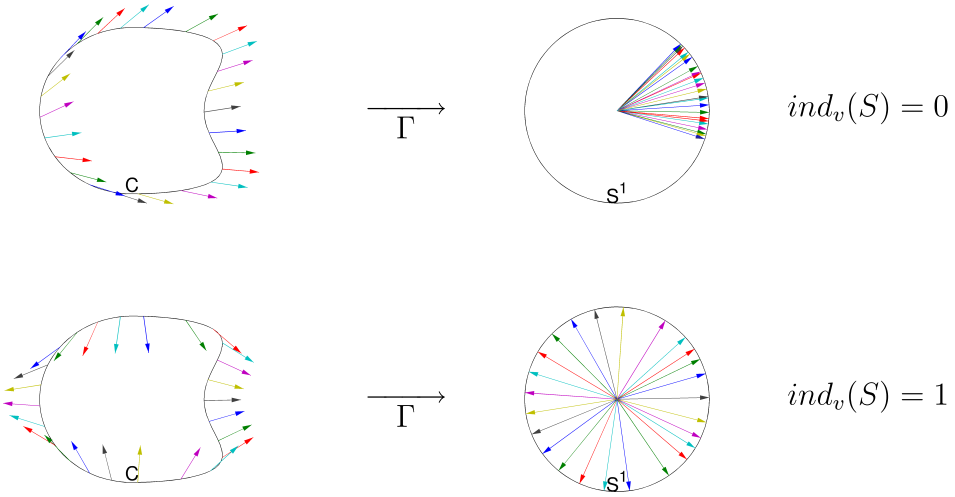


Figure 2.5: Poincaré index in $I\!R^2$

Corollary 2.43. *In $I\!R^2$ a saddle has index -1. Furthermore, a first-order critical point with an index $+1$ either a source or a sink.*

In three dimensions, this is more complex.

Corollary 2.44. *In $I\!R^3$ a saddle point has index $+1$ or -1. Furthermore, a sink has index -1 and a source has index $+1$.*

2.7 Parameter-dependent Dynamical Systems

The previous sections focused on autonomous dynamical systems. Now, if one considers a parametric family of vector fields (inducing a parametric family of dynamical systems), the structure of the phase portrait may change as the value of this parameter evolves. Therefore, the analysis of the corresponding systems is concerned with the essential question of structural stability of the phase portrait, i.e. the ability of a given topology to maintain its qualitative nature under small changes of the parameter value. This section introduces the notions required to precisely define structural stability.

The first definition introduces the notion of small changes to a vector field.

Definition 2.45. *Let f be a a function of class $C^r(I\!R^n)$, $r \geq 1$ and $\varepsilon > 0$. Then a function g is a C^1* ε-perturbation *if there exists a compact subset $K \subset I\!R^n$ such that $f = g$ on $I\!R^n \backslash K$ and for all $i \in \{0, .., n-1\}$, one has*

$$\left\| \frac{\partial}{\partial x_i}(f - g) \right\| < \varepsilon$$

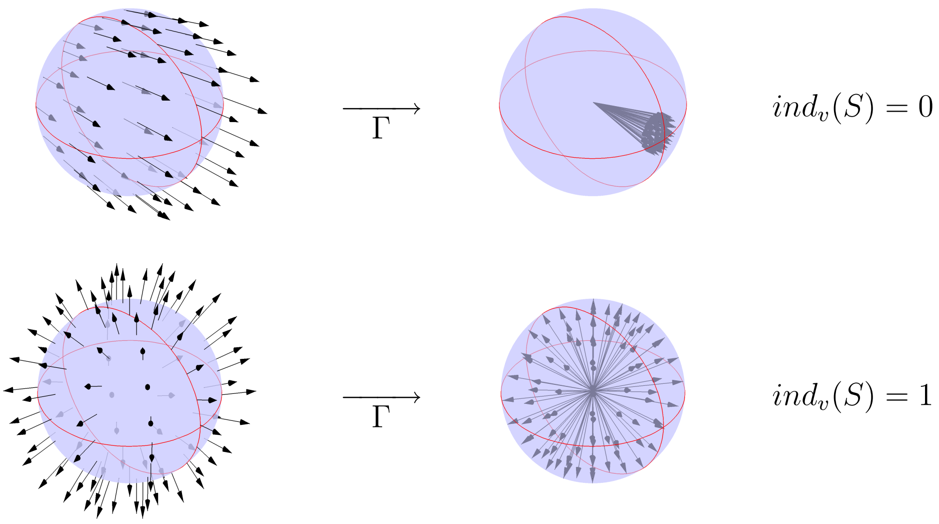


Figure 2.6: Poincaré index in $I\!R^3$

The preservation of the qualitative nature of a dynamical system is intimately related to the notion of equivalence as defined next.

Definition 2.46. *Two C^r vector fields v and $'$ are said to be C^k-*equivalent *($k \leq r$) if their exists a C^k-diffeomorphism H which takes orbits $\phi^v_t(x)$ of v to orbits $\phi^{v'}_t(x)$ of g, preserving sense but not necessarily parameterization by time. If furthermore h does preserve parameterization by time, then H is called* conjugacy *and v and v' are* conjugate *or topologically equivalent in the case $k = 0$.*

Remark. *This definition implies that for any x and t, there is a t' such that*

$$H(\phi^v_t(x)) = \phi^{v'}_{t'}(H(x))$$

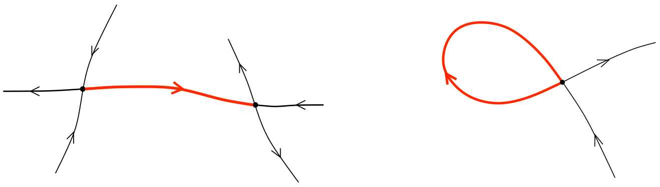
Structural stability is now defined as follows.

Definition 2.47. *A C^r vector field v is* structurally stable *if there is an $\varepsilon > 0$ such that all C^1 ε-perturbations of v are topologically equivalent to v.*

The focus is again on planar vector fields. One first needs to introduce special cases of separatrices of first order critical points.

Definition 2.48. *A separatrix connecting two saddle points is called a* heteroclinic connection. *A closed separatrix connecting a saddle point with itself is called a* homoclinic connection.

Saddle connections are shown in Fig. 2.7 for the two-dimensional case.
For two-dimensional vector fields defined on a compact Euclidean domain, the fundamental Peixoto theorem[Pei62] describes the circumstances that result in structural stability.



(a) Heteroclinic

(b) Homoclinic

Figure 2.7: Saddle connections

Theorem 2.49. (Peixoto) *A C^r vector field v on a two-dimensional compact planar domain of $\mathbb{R}^2$ is structurally stable if and only if:*

1. *the number of fixed points and closed orbit of v is finite and each is hyperbolic;*
2. *there are no orbits connecting saddle points (heteroclinic or homoclinic).*

Practically, Peixoto's theorem implies that a planar vector field typically presents saddle points, sinks and sources as well as attracting or repelling closed orbits. Furthermore, it asserts that non-hyperbolic critical points or closed orbits are unstable because small perturbations can make them hyperbolic. Saddle connections, as far as they are concerned, can be broken by small perturbations as well.

For three-dimensional systems, no such conjecture exists, although several ideas have been formulated (see [Sma67]). In similarity to the two-dimensional case, non-hyperbolicity and absence of saddle connections seem to play a key role in the appearance of structural instabilities.

Now that we have defined the concept of structural stability, we will next be concerned with structural transitions, called bifurcations. The term bifurcation was originally used in the literature to describe the splitting of equilibrium points in a parameter-dependent dynamical system, as the value of this parameter comes to change over its domain of definition: If one depicts the curve describing the successive positions of the equilibrium over the space embedding euclidean and parameter space, one notices for a particular parameter value the presence of a fork that leads to several alternative equilibria. This basic idea is illustrated in Fig. 2.8 for the simplest case of one-dimensional Euclidean space and one-dimensional parameter space (variable μ).

The structure associated with the parameter value where the fork occurs is thus unstable for slight changes of this value can lead to another different structure (non-equivalent in the sense of Definition 2.46). Therefore, the following definition is imminent.

Definition 2.50. *Let v_μ a parametric family of vector fields. A value μ_0 of the parameter μ for which the induced flow ϕ^μ is not structurally stable is called a* bifurcation value *of μ.*

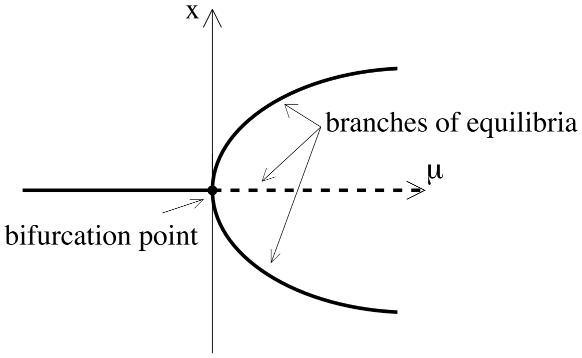


Figure 2.8: Bifurcation diagram

In the definition above, the notion of bifurcation is restricted to a one-dimensional parameter space. This is motivated by the fact that the approach of structural stability employed in this thesis is limited to one-parameter families of vector fields. However, bifurcation problems go far beyond this restriction and actually apply to any n-dimensional parameter space. For a further treatment, see [GH83].

The mathematical branch of *bifurcation theory* is concerned with the structural transitions that occur at the bifurcation values of the parameter. These transitions may be very complicated and an exhaustive classification is impossible, even in the simple case treated here. Yet, two categories exist: On one hand, some bifurcations only affect the nature of a critical point or a closed orbit, and the corresponding new stable state (reached after transition) is to be found in a neighborhood. These bifurcations are called *local bifurcations*. On the other hand, bifurcations that change the global structure of the flow and cannot be deduced from local information are called *global bifurcations*. The planar vector field case has been treated extensively in the literature, see e.g. [Tri02].

In the local case, recalling Definition 2.22, the non-hyperbolicity of a critical point is due to the fact that the Jacobian matrix at the corresponding position has at least one eigenvalue with vanishing real part. An additional constraint is posed by the continuity of the Poincaré index that dictates that the index in a local region around the bifurcation remains unchanged. For specific classes of vector fields, such as piecewise linear fields that vary linearly with a parameter, these constraints result in a simplification of possible cases. As we apply the previously defined concepts in this context, we will provide further discussion (cf. Section 4.3).

The analysis of global bifurcations cannot be reduced to the neighborhood of a critical point. Their common characteristic is that they involve saddle connections, and they are unstable in the sense of Definition 2.47.

2.8 A Note on Divergence-Free Vector Fields

In practical applications, and especially in many cases involving fluid dynamics, divergence-free vector fields are quite common, i.e.

$$\operatorname{div} v = 0 \qquad \text{or equivalently} \qquad \operatorname{trace} Dv(x) = 0.$$

Hence, there are limitation on the nature of critical points that appear in these vector fields. Considering that the eigenvalues must sum to zero, one finds

Property 2.51. *If v is a divergence-free vector field on $\Omega \subset I\!R^2$, all critical points are of type saddle or center. If v is defined on $\Omega \subset I\!R^3$, all critical points are saddles.*

Thus, heteroclinic and homoclinic connections are prevalent in divergence-free vector fields.

2.9 Lyapunov Exponents and Chaos

Although there is still no unanimously accepted mathematical definition of a chaotic map, the term *chaos* in the context of a dynamical system refers to a sensitive dependence on initial conditions. In other words, as Theorem 2.12 holds, orbits starting together will stay together for a while. Stability is a different notion: orbits starting together will stay together, forever. In essence, while sensitive dependence on initial conditions does not contradict continuity, it is at odds with the stability property.

Among the tools developed to study chaotic behavior in dynamical systems, the Lyapunov exponent is most prominent. It exploits the fact that arbitrarily close initial conditions may lead to evolutions that diverge exponentially fast with time. We return to the non-autonomous case of dynamical systems to define it in the following.

Definition 2.52. *Let ϕ a dynamical system on $I\!R^n$ and $x_0 \in I\!R^n$. Consider two trajectories $\phi_x(t)$ and $\phi_{x'}(t)$, starting at $x, x' \in \Omega$, and denote*

$$\delta x := x - x' \qquad \text{and} \qquad \delta\phi(t) := \phi_{x'}(t) - \phi_x(t).$$

The Lyapunov Exponent *is then given by*

$$\lambda(x) := \lim_{x' \to x} \lim_{t \to \infty} \frac{1}{t} \ln \frac{||\delta\phi(t)||}{||\delta x||}.$$

Roughly speaking, the Lyapunov exponent measures the mean exponential separation rate

$$||\delta\phi(t)|| \approx e^{\lambda t} ||\delta x||$$

between trajectories that start infinitesimally close. Informally, it allows a brief classification of orbits as follows.

$\lambda(x) \leq 0$: The orbit attracts to a stable fixed point or closed orbit. In the case $\lambda(x) < 0$, the fixed point or closed orbit is asymptotically stable.

$\lambda(x) > 0$: The orbit is unstable and chaotic. Nearby points result in exponentially diverging trajectories.

We will require this concept in Chapter 5 where we will replace it with a finite-time analogue, the *Finite Time Lyapunov Exponent*, to study the separation characteristics of time-dependent flow fields. Note that the Lyapunov exponent can also be defined for discrete dynamical systems by replacing time t with iteration number n.

2.10 Numerical Treatment

In the following, we will briefly present a short overview over the most important aspects of the numerical solution of ordinary differential equations such as Equation (2.1). As the topic is very broad, our discussion will be limited to the specific requirements of this thesis. For an extensive exposition, see [HNW93].

Many differential equations cannot be solved analytically, including such important cases as interpolated data from numerical simulations. It is the purpose of *numerical integration* to provide an approximation to the real solution in this case. In the following, we will call solutions to this approximation problem *integration methods* or *numerical integrators*.

The basic object of study is the Cauchy problem (Def. 2.3), in this context often termed *initial-value problem* (*IVP*). Theorem 2.10 states that a global solution is uniquely determined under the condition that the vector field underlying Equation (2.1) is Lipschitz. The fundamental principle applied by most integration methods is a piecewise construction of the solution of an ordinary differential equation along the lines of Theorem 2.9.

2.10.1 Basic Properties

To illustrate the subject and introduce notation, we will first discuss two simple methods. In the following, let v a Lipschitz continuous vector field defined on $[0, T] \times I\!R^n$, $T > 0$. Replacing the derivative $y'(t) := \frac{dy}{dt}$ by a finite-difference approximation in Equation (2.1)

$$y'(t) \approx \frac{y(t+h)) - y(t)}{h} \tag{2.7}$$

and solving for $y(t+h)$ yields the following formula:

$$y(t+h) \approx y(t) + hv(t, y(t)). \tag{2.8}$$

Choosing a *step size* $h > 0$, one can construct a sequence $(t_n) := nh$. We denote by y_n a numerical estimate of the exact solution $y(t_n)$. By recursive application of Equation (2.8), we obtain

$$y_{n+1} = y_n + hv(t_n, y_n).$$

This scheme is called the *forward* or *explicit Euler method.* It is the simplest example of the class of *explicit* integration methods. The nomenclature signifies an explicit dependence of y_{n+1} on y_n. In contrast, solving for $y(t)$ in Equation (2.7) results in the *backward* or *implicit Euler method*:

$$y_{n+1} = y_n + hv(t_{n+1}, y_{n+1}).$$

In the latter, an equation must be solved to find y_{n+1}. Since v is typically non-linear, some form of iteration must be employed to accomplish this.

Two basic concepts are used in the analysis of such methods, given next.

Definition 2.53. *A numerical method is said to be* convergent *if the numerical solution y_n approaches the exact solution $y(t_n)$ as $h \to 0$. More precisely, we require that*

$$\lim_{h\to 0+} ||y_n - y(t_n)|| = 0.$$

Both forward and backward Euler schemes are convergent. In fact, it is a required condition for any integration method.

Definition 2.54. *If the scheme is of the form*

$$y_{n+k} = F(t_{n+k}; y_n, y_{n+1}, \ldots, y_{n+k}; h)$$

for a function F, then the local approximation error *is the error introduced by one step of the scheme, i.e.*

$$\delta^h_{n+k} : F(t_{n+k}; y_n, y_{n+1}, \ldots, y_{n+k}; h) - y(t_{n+k}).$$

The scheme is called consistent of order p *if*

$$\delta^h_{n+k} = O(h^{p+1}) \qquad \text{as } h \to 0,$$

or simply consistent *if* $p \geq 1$.

Again, both Euler methods are consistent of order 1. Typically, the design goal of many methods is to attain a higher order. Remark that consistency is a necessary condition for convergence.

Definition 2.55. *Consider the ordinary differential equation $y' = ky$ for $k < 0$. The solution*

$$y(t) = e^{kt} \tag{2.9}$$

approaches zero as $t \to \infty$. If the numerical method also exhibits this behavior, it is called A-Stable *or simply* stable.

While the explicit Euler method is not stable, the implicit Euler method is. The importance of this last criterion is based on the fact that many typical problems from applications include such exponential behavior (often called *stiffness*). Therefore, stability is a required property for such stiff systems.

To present an overview of integration schemes, or even classes of integration schemes, is an exhaustive task. There are many specialized solutions and methods that work well for one type of problem break down completely when applied to another. In the following, we will focus on a class of integration methods that have been traditionally used in Scientific Visualization to compute solutions of Equation (2.1) for flow vector fields.

2.10.2 Runge-Kutta Methods

Runge-Kutta methods are available in both implicit and explicit flavors. We will limit our presentation to explicit schemes. A Runge-Kutta method is of the general form

$$y_{n+1} = y_n + \sum_{i=1}^{s} b_i k_i,$$

where

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f(t_n + c_2 h, y_n + a_{21} h k_1) \\ k_3 &= f(t_n + c_3 h, y_n + a_{31} h k_1 + a_{32} h k_2) \\ &\vdots \\ k_s &= f(t_n + c_s h, y_n + h \sum_{i=0}^{s-1} a_{s,i} k_i). \end{aligned}$$

A particular method is thus specified by the integer s (called number of *stages*) and the coefficients a_{ij} for $1 \leq j < i \leq s$, b_i for $i = 1, \dots, s$ and c_i for $i = 2, \dots, s$.

One of the interesting aspects of Runge-Kutta methods is that their properties such as convergence and consistency order may be read off their coefficients[HNW93] in the form of products and sums. One can show that the order of a Runge-Kutta scheme can be increased linearly with each stage, up to order 5, which already requires 6 stages. For additional orders, the number of required stages increases superlinearly, therefore, order 5 is quite common as a good compromise between high consistency order and limited number of function evaluations. We will go on to describe two Runge-Kutta schemes that or of interest to this thesis.

Fehlberg's method is quite interesting in that it offers an automatic, adaptive choice of step size h. This is achieved by using six stages, and combining them differently to simultaneously obtain two approximations of order 4 and 5. Runge-Kutta formulas of this type are called *embedded*. From a comparison of both, an estimate of the local approximation error of the fourth-order method can be derived, and the step size is controlled such that a specified bound is not surpassed. The method is often abbreviated RKF. While the overall accuracy benefits from an optimal choice of step size, it is now no longer necessary to choose the step size overly conservative to ensure that the solution is correctly approximated. The performance increase gained by taking larger steps where possible is substantial for most applications.

In the spirit of Fehlberg's method, many embedded Runge-Kutta formulas have been found. The method by Cash and Karp[CK90] (RKCK or RK45) is the de-facto standard in many applications. Its popularity is due to its range of applicability for generally non-stiff problems, and it is the integration method given in the widespread *Numerical Recipes* series[PFTV92]. It is available for example in the visualization system VTK.

However, application of these methods in Scientific Visualization is not without problems. There, it is often required to represent integral curves graphically. Hence, the sequence of output points y_n must be transformed to a continuous curve that is then depicted. Lacking further information about the solution in between the y_n, approximating the solution curve by a piecewise linear function is the only possibility. This often leads to severe visual artifacts and is unsatisfactory in general. Moreover, step size control tends to increase these artifacts. By its design goal, it ensures that the solution is correctly approximated by a non-linear polynomial.

In contrast, we make use of the method of *Dormand* and *Prince* (DOPRI5) that offers all benefits of an embedded Runge-Kutta pair of orders 4 and 5, but is additionally capable of *dense output*. That is, for every step taken there exists a polynomial p_n of order 5 with

$$p_n(0) = y_n \qquad \text{and} \qquad p_n(1) = y_{n+1}.$$

Moreover, if $u_n(t)$ is the local solution of Equation (2.1) with initial condition $u_n(0) = y_n$, then p_n enjoys the local approximation property

$$p_n(s) - u_n(sh) \approx O(h^4).$$

The curve described by joining these polynomials over the sequence (t_n) is globally C^1.

There is one slight drawback in that dense output for the Dormand-Prince scheme requires an additional stage, resulting in a total of seven stages. However, DOPRI5 is an *FSAL* (first same as last) scheme, meaning that the last stage of a step is the first stage of the next one. Therefore, in practice, this does not result in a loss of performance with respect to e.g. RKCK. The interpolation polynomial is given directly as a combination of the stages $k_1, \ldots, k_7$ and is therefore available at no additional cost. For a detailed treatment, we refer the interested reader to [PD81, Sha85, GSBB87, Hig91]. As to our knowledge this type of integration method has not been used before in Scientific Visualization, we have provided a documented sample implementation in Appendix A.1 for completeness and reproducibility. Note that the Dopri5 method was chosen as the default integration method for non-stiff problems in the ubiquitous mathematical programming environment Matlab for reasons of overall good performance and applicability.

Chapter 3

Stream Surfaces

Stream surfaces, as a natural generalization of streamlines, are a powerful tool for insightful flow visualization. Essentially a continuum of streamlines, they constitute a surface and allow the application of shading techniques. Hence, in comparison to streamlines or other line-based techniques, they support depth perception and greatly facilitate the visual understanding of three-dimensional structures.

Stream surfaces appear quite naturally in flow visualization. The groundbreaking drawing work of Dallmann [Dal83b] has shown that flow structures can be well understood in terms of *flow sheets* (see Figure 1.2 on p. 11). These sheets represent precisely chosen stream surfaces that emanate from specific separation and attachment lines on the surface of objects embedded in a flow (cf. Section 1.1.5), and often take on the role of flow separators that divide regions of differing behavior. In the same sense, stream surfaces appear as two-dimensional separatrices in three-dimensional vector field topology, where they describe two-dimensional stable or unstable manifolds of critical points of a vector field. Even if not coupled to such specific meanings, stream surfaces have great illustrative power.

In recent years, several approaches have been presented for the computation and graphical representation of stream surfaces in CFD datasets. However, there has been lack of an algorithm able to deal with the complicated flow structures contained in state-of-the-art datasets, owing to the difficulties posed by complex object geometries and associated flow fields. In this chapter, we will present a novel algorithm for the computation of stream surfaces in steady vector fields. Our algorithm borrows the basic principle of adaptive front advection first introduced by Hultquist in [Hul92] and and incorporates accurate arc length integration, exact sampling, and highly adaptive front control to effectively tackle the challenges posed by complex, realistic flow data.

We will first discuss the formal setting of stream surface computation and briefly survey available algorithms. Then, we proceed with an in-depth discussion of the characteristics of advancing front methods and present our novel approach based on these principles. Finally, we demonstrate several visualization applications enabled by the presented algorithm.

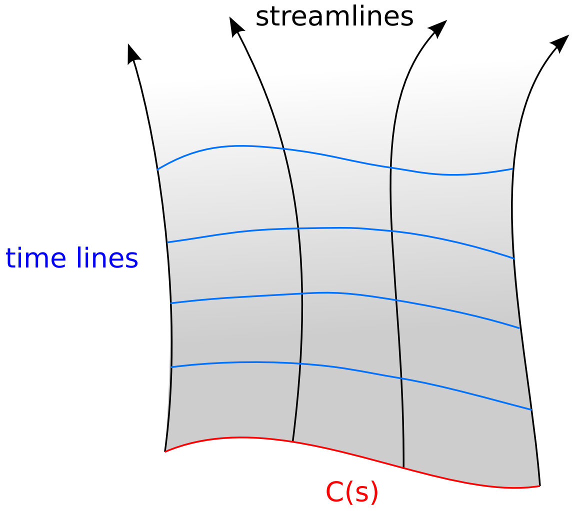


Figure 3.1: A stream surface.

3.1 Formal Definition

In the following, let $T > 0$, $\Omega \subset \mathbb{R}^3$, and $v : \Omega \to \mathbb{R}^3$ a stationary vector field. Furthermore, let $x \in \Omega$.

Definition 3.1. *A* streamline $S(t,x)$ *is an integral curve through* $x \in \Omega$ *in the dynamical system* ϕ *generated by* v *(Definition 2.5).*

The second argument x to S denotes the starting point of the streamline, while the first argument t parameterizes the curve that is the stream line. The intuitive understanding associated with streamlines is that of massless particles that are advected through a domain by a stationary vector field as time t increases. Now, let $C : [0,1] \to \Omega$ a space curve, parameterized by s.

Definition 3.2. *A* stream surface *is a two-dimensional surface* $\mathcal{S} : [0,T] \times [0,1] \to \Omega$ *defined by*

$$\mathcal{S}(t,s) := S(t, C(s)). \tag{3.1}$$

In other words, $\mathcal{S}$ is a continuum of stream lines emanating from C. If $t \in [0,T]$ is fixed, the set $S(\cdot,t)$ is called *time line* of $\mathcal{S}$ at t. The graphical analogy is obvious: as t increases from 0 to T, the time line $S(\cdot,t)$ traverses all of $\mathcal{S}$. Conversely, fixing s results in individual streamlines. Figure 3.1 provides a more graphical explanation of these terms.

Since v is continuous, Equation (2.1) implies that $\mathcal{S}$ is differentiable with respect to t. Furthermore, it can be easily derived that $\mathcal{S}$ is (piecewise) differentiable in s if v is (piecewise) differentiable. Even in the presence of hyperbolic limit sets that may induce an arbitrary divergence of neighboring streamlines (refer also to the discussion in Section 5.1), the surface is discontinuous only in the limit $T \to \infty$. We exclude this as an unreasonable case in good conscience. Hence, $\mathcal{S}$ is a piecewise regular surface and possesses well-defined normals on each s-piece. Therefore, construction of a high-quality graphical representation is a feasible task.

We note that $\mathcal{S}$ has a natural parameterization in the form of (s, t)-coordinates. Every point on the surface, given its parametric coordinates (s, t), can be computed by propagating the unique streamline starting at $C(s)$ through the application of a integration method until it reaches t. The goal of any stream surface algorithm is then to construct a geometric approximation to Equation 3.1. We proceed to discuss several approaches that have been proposed in previous work.

3.2 An Overview of Computational Algorithms

The most commonly used algorithm was introduced by Hultquist[Hul92] in 1992 and is based on an advancing front of discrete streamlines spanning the stream surface. Streamline computation is delegated to a numerical integration method. Adaptive front resolution is used to handle converging and diverging behavior of the flow by a simple insertion and merging heuristic that controls the number of front streamlines. The scheme is straightforward to implement and fast, but performs well only for simple flows. Stalling [Sta98] augmented Hultquist's scheme by incorporating topological information into the triangulation process and slightly modifying the refinement criteria.

Adopting the advancing front idea of Hultquist, Scheuermann et al.[SBH*01] exploit the existence of an analytic flow solution for tetrahedral grids endowed with linear interpolation to compute a stream surface on a per-tetrahedron basis. Due to the linear nature of the vector field inside every grid cell, streamline paths are available as a closed-form solution to Eq. 2.1. Therefore, the use of a numerical integration scheme is not required. Assuming the intersection of an existing stream surface piece with the faces of a tetrahedron is given, the surface can be continued analytically inside the tetrahedron. After computation of the intersection curves on the remaining faces, the mesh adjacency structure is used to recursively advance the surface through neighboring tetrahedra, until the full surface is formed. Hence, the algorithm has characteristics of a *marching* method. Moreover, the restriction to piecewise-linearly interpolated vector fields allows a determination of the topological configuration of the vector field in each tetrahedron that is traversed by the surface. This information is then used to provide stable and deterministic behavior in the presence of critical points, which account for surface discontinuities and often pose a difficulty for other stream surface algorithms.

In spite of these positive characteristics, the algorithm is rarely used in practice. To produce computationally feasible implementations, some simplifications must be introduced. Typically, the intersections of the stream surface with mesh faces are assumed straight lines, restricting the surface inside a cell to a ruled surface. Moreover, the resolution of the resulting surface is closely tied to that of the underlying mesh, since at least one triangles is generated for every cell traversed. Frequently, this causes large triangle sets even for geometrically simple surfaces. Last, the limitation to tetrahedral grids is a serious impediment to applications that base model computations on higher-order interpolation schemes.

In contrast to such methods that compute the surface geometry explicitly, van Wijk[vW93] uses a global approach that implicitly represents a family of stream surfaces. Through advection of a scalar field from the domain boundary through the flow domain, the computation of a particular stream surface is transformed into the extraction of an iso-surface. The latter problem is well understood, and many reliable methods are available[LC87, Nie04]. The generation of the scalar field itself, however, requires extensive processing. Essentially, the domain of interest is discretized, and a scalar field is prescribed on the domain boundary. Then, for every sample point, a streamline is traced until it intersects the domain boundary, and the sample point is assigned the corresponding scalar field value. In this fashion, all data points along a single streamline are assigned similar values, and isosurfaces approximate stream surfaces by representing all streamlines that share a single isovalue. However, selecting stream surfaces is very non-intuitive, and defining a suitable boundary field is hard. Cai and Heng[CH97] have shown that so-called *principal stream surfaces* can automatically select and compute interesting surfaces, but their work is limited to irrotational flows. For state-of-the-art datasets, the algorithm is not applicable, as the required resolution and hence the number of streamline integrations is prohibitive.

Aside from explicit construction of stream surface geometry, methods exist that create the visual impression of a stream surface by using particles, e.g. [vW92]. While simple to implement and broadly applicable, the visual clarity of these schemes is often lacking, as the depth-enhancing quality of shaded surfaces is lost. Furthermore, by forfeiting an explicit representation of the surface geometry, it is impossible to use stream surfaces as building blocks for advanced visualization techniques (cf. Section 3.5).

While all of these methods have different pros and cons, Hultquist's scheme is most often used in practical applications as it is comparatively stable and fast, and applicable to typical vector fields. It should be noted that in principle, Hultquist's method should be able to deal with flows of arbitrary complexity if only the prescribed resolution is high enough since, in the limit, a large enough number of streamlines is able to provide an approximation of a stream surface of arbitrary accuracy. However, implementing this basic idea in practice leads to wasteful computation, and the problem becomes intractable for complex flows.

3.3 Advancing Front Methods

In a nutshell, advancing front methods approach the problem of constructing a two-dimensional surface embedded in a three-dimensional flow field by generating a well-distributed collection of sample points on this surface. From these samples, a polygonal representation is generated.

3.3.1 Basic Principles

Looking back at Eq. 3.1, a very simple approximation scheme consists of a regular uniform sampling of the surface based on the (rectangular) parameter domain. The corresponding finite number of streamlines with parameters (s_i) may be sampled by numerical integration to produce a sequence of points (s_i, t_0) through (s_i, t_N). Here,

$$s_i := i\Delta s \quad \text{and} \quad t_j := j\Delta t,$$

for some choices of Δs and Δt. These points then form a simple quadrilateral description of the surface.

However, typical flow fields exhibit convergent and divergent behavior of streamlines and stream surfaces tend to fold and twist. Typically, a uniform sampling will not result in an adequate discretization of the surface. Therefore, some manner of adaptive sampling is called for, and the local resolution must be based on the geometry of the surface in physical space.

In order to achieve this, an *advancing front* is employed that incrementally traverses the whole surface. This front progresses along a finite set of streamlines, whose cardinality is increased or decreased based on the observed properties of the front segments described by these streamlines. Each of the streamlines is in turn discretized as a number of points. If the current front discretization is found to become inadequate as the front is advanced, modifications to the streamline set are performed appropriately.

Making use of the terminology introduced in [Hul92], the stream surface is described in terms of ribbons that represent the surface area in between two adjacent streamlines (designated as left and right). Through a discretization of the bounding streamlines, each ribbon contains a front segment that connects a point on the ribbon's left streamline to one on the right streamline (cf. Figure 3.2). In this setting, advancing the front is reduced to picking the next point on either the left or right streamline on the ribbon, and to generate a new front segment that reflects this choice. For example, in Figure 3.2, the current front in the leftmost ribbon is the line segment (L_0, R_0). The segments (L_0, R_1) and (L_1, R_0) are then candidates for the new front.

A triangular representation of the ribbon is automatically obtained by observing that a triangle is formed by the old and new front segments, independent of the side that is chosen to advance. To ensure a consistent front representation across multiple ribbons, neighboring ribbons must be advanced in turn until the front

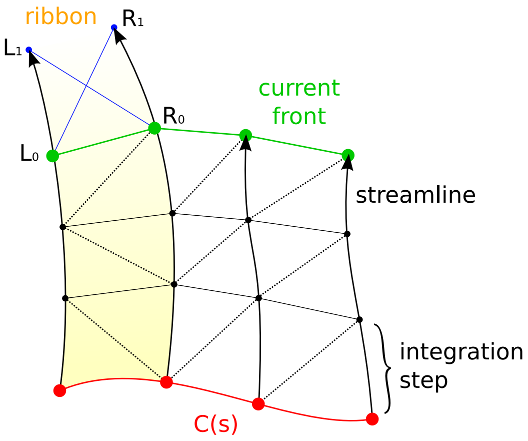


Figure 3.2: An overview of the elements of an advancing front discretization of stream surfaces.

segments are reconnected. Metaphorically speaking, by advancing one ribbon its neighbors are dragged along.

3.3.2 Front Resolution Control

During advancement, the front consists of line segments on the stream surfaces and may be subjected to criteria that determine the quality of the local approximation. If one of these criteria is not met, a ribbon is split in two, increasing the local resolution. Conversely, if the resolution can be reduced by joining neighbor ribbons. In both cases, special triangulation patterns are used (see Figure 3.3).

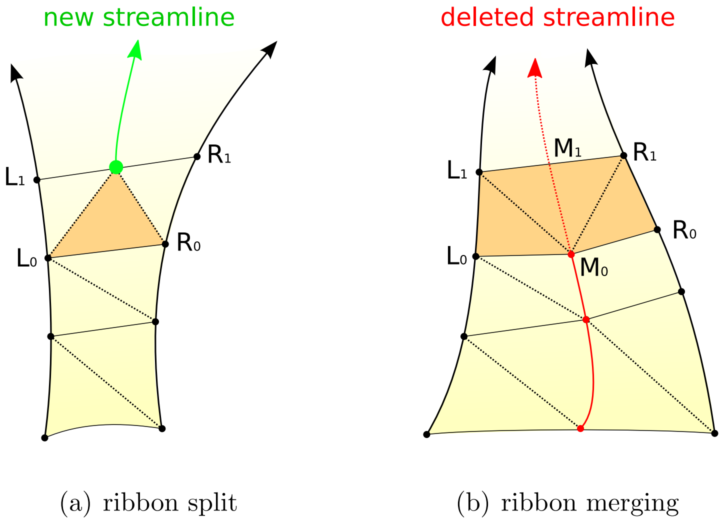
Until now, we have ignored the case that a streamline cannot be integrated further, i.e. a next point is not available. Obviously, if a streamline leaves Ω or reaches integration parameter t, it cannot be integrated further. In such cases, the front becomes discontinuous, and this is reflected algorithmically by splitting it into two separate fronts which are then advanced independently.

3.3.3 Hultquist's Implementation

Hultquist presented the first stream surface algorithm incorporating the above principles [Hul92]. He used a numerical integration method to discretize individual streamlines according to a fixed time step Δt. To generally obtain well conditioned triangles, he employed *greedy minimal tiling* by always choosing the shorter of the two diagonals (L_0, R_1) and (L_1, R_0) as the new front segment (Figure 3.2).

Front resolution is controlled by a simple heuristic:

1. If the width of the quadrilateral at the end of a ribbon between two neighboring streamlines surpasses the height by a factor of two, the ribbon is split.



(a) ribbon split (b) ribbon merging

Figure 3.3: Front resolution control through splitting and merging of ribbons.

A new streamline is inserted into the front at the middle of the line segment (L_1, R_1) (cf. Figure 3.3(a)).

2. If the six points given by the quadrilaterals at the end of two neighboring ribbons $(L_0, M_0, R_0, L_1, M_1, R_1)$ are roughly coplanar and if the merging of ribbons would not violate criterion 1, then the ribbons are joined (cf. Figure 3.3(b)).

3. If the streamlines bounding a ribbon diverge in almost opposite directions, or if one of the streamlines cannot be continued, the ribbon is erased, splitting the front in two.

These criteria are applied after every iteration of advancement. Furthermore, Hultquist found it beneficial to incorporate additional logic that works toward keeping the front locally orthogonal to the flow direction.

Implementation of the algorithm and the corresponding criteria along the lines sketched out in [Hul92] is not entirely trivial, since it is part recursive, part iterative. While the quality of the resulting surfaces is good for vector fields with low complexity, the algorithm in its presented form suffers from a number of difficulties that preclude its application to more complex datasets.

- The primary resolution parameter is Δt and controls streamline discretization. Choosing Δt right for any given dataset is a hard problem, since a careful balance is required between precise approximation and computational effort. However, such a compromise may not exist. If Δt must be chosen small to resolve small structures on a small section of the surface, the

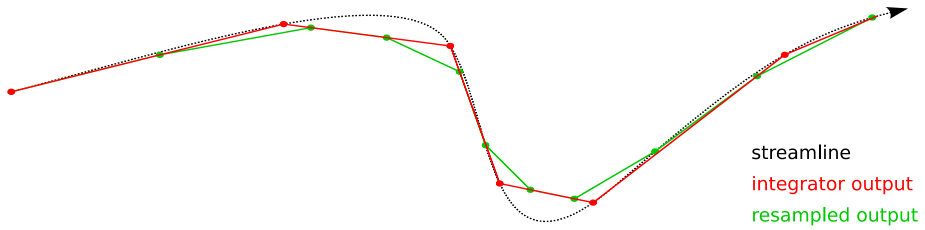


Figure 3.4: Discretization errors introduced by resampling the integration method's output points at fixed time intervals.

effective triangle size is limited for the whole surface, at the expense of an excessive amount of generated triangles. Experience shows that the process of choosing the "right" Δt usually requires a fair amount of trial and error.

- In choosing Δt fixed for the whole surface, the length of triangle edges along streamlines is directly coupled to the magnitude of the traversed vector field. If the magnitude varies strongly between neighboring streamlines, the algorithm generates many skinny triangles, as illustrated in Figure 3.5(a).

- The given refinement criteria are aimed at compensating divergent and convergent flow behavior. Flow sheet folding and twisting are not adequately handled since the curvature of the front is not taken into account as a criterion for refinement. In our experience, this is the most important shortcoming when applying Hultquist's algorithm to complicated vector fields.

- Requiring streamlines to be sampled with fixed Δt-intervals precludes the use of state-of-the-art numerical integrations schemes that select a step size automatically based on the characteristics of the vector field (cf. Section 2.10). Commonly, a benefit in both accuracy and efficiency is achieved. Application of such modern schemes is still possible by way of resampling the resulting irregularly-spaced point sequence at regular intervals. However, this may introduce significant approximation errors (refer to Figure 3.4 for a graphical illustration).

3.4 High-Quality Stream Surface Computation

In this section, we will develop a novel algorithm based on the advancing front principle using [Hul92] as a backbone.

3.4.1 Improved Streamline Integration

For the integration of streamlines, it is typical to make use of adaptive timestepping to integrate the underlying ordinary differential equation for reasons of both accuracy and performance. As discussed in Section 2.10, the sequence of output points of an adaptive integration scheme has irregular spacing with respect to t. However, the advancing front principle forces a more or less regular t-spacing, since large differences in step size between neighboring streamlines lead to ill-conditioned triangulations and aggravate front resolution control.

An adaptive integration scheme can still be used by interpreting the sequence of output points as a piecewise linear approximation. Then, resampling can be employed to generate a t-equidistant sequence. However, this is not optimal since significant approximation errors are incurred, mitigating the positive aspects of the superior integration method (cf. Figure 3.4).

Instead, we employ the numerical integration scheme developed by Dormand and Prince[PD81] (Dopri5 from Section 2.10). It is an adaptive Runge-Kutta scheme[HNW93] and admits dense output, i.e. a differentiable curve instead of a discrete sequence of points. A thorough discussion of this scheme is presented in Appendix 2.10, and a model implementation is given in Appendix A.1

3.4.2 Arc Length Sampling

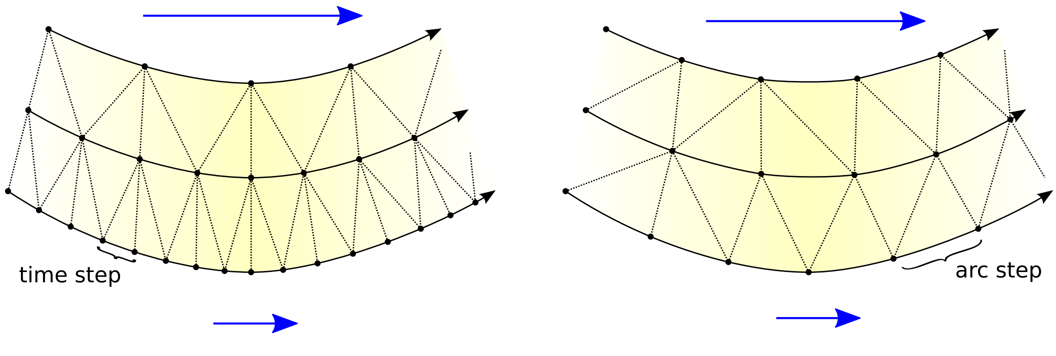
In regions of inhomogeneous flow, sampling streamlines with a fixed timestep can result in a low quality triangulation that contains many skinny triangles, see Figure 3.5(a). On the other hand, we note that sampling at fixed intervals of arc length (Figure 3.5(b)), triangle size is effectively decoupled from vector field magnitude, resulting in an improved triangulation with fewer triangles. We therefore make use of an *arc step length* Δa as the primary streamline discretization parameter in our algorithm.

Although ignoring magnitude information in the discretization process may seem detrimental at first glance, it is quite natural since the definition of a stream surface (Equation. 3.1) does incorporate it.

The arc length of a parametric curve $C(t)$ is a function of the parameter $t \in [0,1]$ and is usually given in integral form

$$\mathbf{a}_C(t) := \int_{t_0}^{t} \|\frac{dC}{dt}(\tau)\| d\tau \qquad \text{for } t \in [0,1]. \tag{3.2}$$

For general parametric curves, it is usually hard to compute arc length at arbitrary values of t because evaluation of the integral must rely on numerical quadrature. To address this problem, we reformulate Equation (3.2) as an ordinary differential



(a) fixed timestep sampling (b) fixed arc length sampling

Figure 3.5: A comparison of triangulations resulting from greedy minimal tiling. Arc length sampling (right) produces more regular triangulations and less skinny triangles than timestep sampling (left).

equation:

$$\mathbf{a}_C(0) = 0 \tag{3.3}$$

$$\frac{d}{dt}\mathbf{a}_C(\tau) = ||\frac{dC}{dt}(\tau)|| \qquad \text{for } \tau > 0. \tag{3.4}$$

For the special case of a streamline, it follows directly that (omitting x for simplicity of notation)

$$\frac{d}{dt}\mathbf{a}_S(\tau) = ||\, v(S(\tau))\, ||. \tag{3.5}$$

Hence, as v is Lipschitz on Ω, $\mathbf{a}_S$ exists and is unique for any given S (Theorem 2.10). Using an integration scheme with dense output as discussed above, a continuous representation of $\mathbf{a}_S$ is easily obtained.

To obtain evenly spaced arc length samples of S, we first note that for any given streamline, $\mathbf{a}_S$ is continuous and strictly monotonically increasing and hence invertible on its image. Therefore, we are able to define

$$\hat{S}(a) := S(\mathbf{a}^{-1}(a)) \qquad \text{for } a \in [0, \mathbf{a}_S(T)].$$

and replace S by $\hat{S}$ to transform the algorithm from time-based sampling to arc-length sampling.

From a numerical point of view, finding $\mathbf{a}$ can be achieved by solving the simultaneous system of equations

$$\left.\begin{pmatrix} S \\ a_S \end{pmatrix}\right|_{t=0} := \begin{pmatrix} x_0 \\ 0 \end{pmatrix} \tag{3.6}$$

and

$$\frac{d}{dt}\begin{pmatrix} S \\ a_S \end{pmatrix}\Bigg|_t := \begin{pmatrix} v(S(t)) \\ ||v(S(t))|| \end{pmatrix}. \tag{3.7}$$

using e.g. the Dopri5 scheme. This entails little additional effort, since both equations rely only on the same values of v. Once both S and $\mathbf{a}_S$ are available, $\hat{S}$ is then evaluated by inverting $\mathbf{a}_S$ using a Newton-type method. While requiring a nonlinear inversion procedure for every sample may seem computationally intensive, we have found its cost to be negligible in our experiments.

Beyond generation of better stream surface triangulations, using an arc length step Δa as the primary unit of sampling resolution allows for a more intuitive understanding of resolution parameters. In contrast to time, arc length has a geometric significance in physical space and is thus ideally suited to compare discretization scale to structure size.

3.4.3 Front Resolution Control

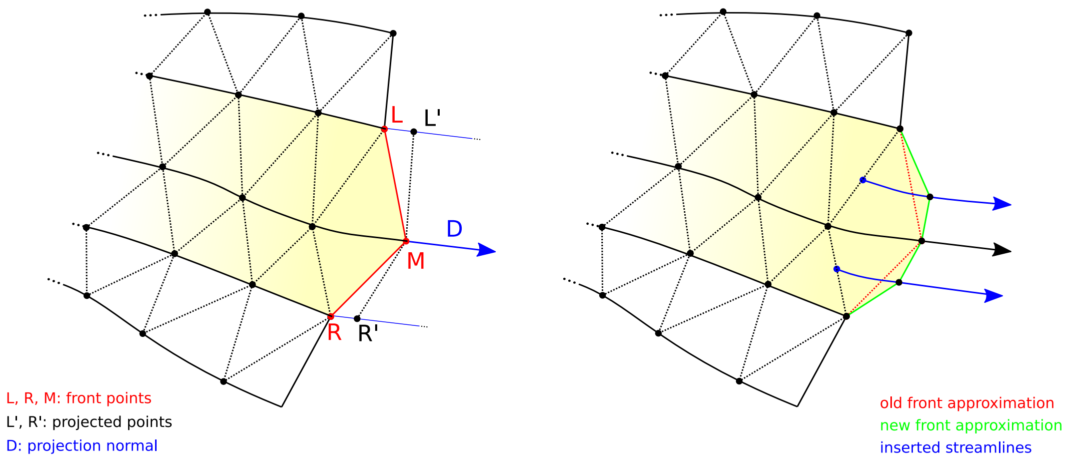
To achieve a good front approximation in the presence of strong distortions, we adopt the front resolution criteria described next.

Divergence / Convergence Monitoring To compensate for convergence and divergence of neighboring streamlines, we implement a simple distance-based criterion by choosing an upper bound $d_{\max}$ on front segment length. If this threshold is passed, a ribbon is split. In similar fashion, if the combined front segment length of two ribbons falls below a prescribed threshold $2 \cdot d_{\min}$, these ribbons are candidates for merging. Naturally, choosing $d_{\min} < d_{\max}$ is sensible.

Curvature Monitoring To control refinement based on front curvature, we introduce the angle α formed by the projection of two adjacent front segments onto the plane orthogonal to the tangent of the common streamline. See Figure3.6(a) for a more detailed explanation. If α exceeds a user-defined threshold $\alpha_{\max}$, the corresponding ribbons are split, and the criterion is recursively reapplied to the new front approximation. Since the front is continuous, the piecewise linear front approximation obtained by successive ribbon splittings converges quickly. Therefore, infinite refinement is not possible.

In order to disallow merging of adjacent ribbons whose front segments exhibit too large an angle, we again make use of a threshold $\alpha_{\min}$ to exclude these ribbons from merging.

Fine-grained Resolution Control To guarantee the optimal application of the presented criteria, checking the front after each advancing step is not sufficient. Rather, the criteria must be applied every time a single ribbon is advanced, and before the generation of output triangles.



(a) curvature measurement (b) refined front

Figure 3.6: Front curvature measurement is performed by projecting the front points L and R from two neighboring ribbons onto a D-orthogonal plane through M (left). If $\alpha = \angle(L', M, R')$ exceeds a threshold $\alpha_{\max}$, the ribbons are split and a new front approximation is obtained (right).

Exact Streamline Termination Conventionally, a streamline is prematurely terminated if it leaves the computational domain. In this case, a next sample point is not available. Through the use of arc length sampling, we have introduced a similar case: While a streamline takes infinite time to reach a critical point, it accumulates only a finite arc length. Again, it is not possible to evaluate the next Δa-sample on the streamline.

Identification of the first case is easily accomplished by checking if the current integration position is contained in Ω. It is in general beneficial to not terminate the streamline right away. Rather, using a reduced step length, a last streamline sample can be computed as close as possible to the boundary of Ω, and the streamline is terminated in the successive step. Thus, the front is split close the boundary instead of some distance away from it. The benefit of this approach lies in a better front approximation close to geometrically complex domain boundaries. The reduced stepsize can be found by using a simple interval bisection technique.

The second case is inferred indirectly by observing the magnitude of $||v||$ and terminating the streamline integration if it drops below a minimal threshold. Since a full step of length Δa cannot be achieved, the step size is again reduced to generate a last streamline sample at the vanishing point of v. Together with the divergence criterion from Section 3.3.2 and front curvature control, this yields improved front approximations right up to the splitting point and allows for a reliable treatment of critical points in the vector field without taking local vector field topology into account explicitly.

Interestingly, the second case is quite common for typical flow simulation data. Objects embedded in the flow are often subject to no-slip boundary conditions (cf. Section 1.1.1), requiring that the flow velocity tends to zero as the boundary is approached. Approaching such objects, streamlines take an infinite time to reach the object boundary as the velocity approaches zero. Hence, these boundaries act similar to critical points concerning the front termination. We have found that correct handling of these cases improves the quality of the front approximation greatly.

3.4.4 Implementation

Improved streamline integration and arc length sampling are incorporated into the advancing front concept in a straightforward fashion. In the simplest form, a corresponding algorithm is easily implemented in a recursive manner based on a ribbon data structure. If a ribbon is advanced, the neighboring ribbons must follow recursively to guarantee a consistent front approximation. However, fine grained resolution control is at odds with this approach. To control curvature, for example, requires incorporation of front segments in neighboring ribbons. If a split is deemed necessary, the triangle that was already generated during the recursive advancement must be deleted, requiring complex bookkeeping to ensure that the triangulation remains intact.

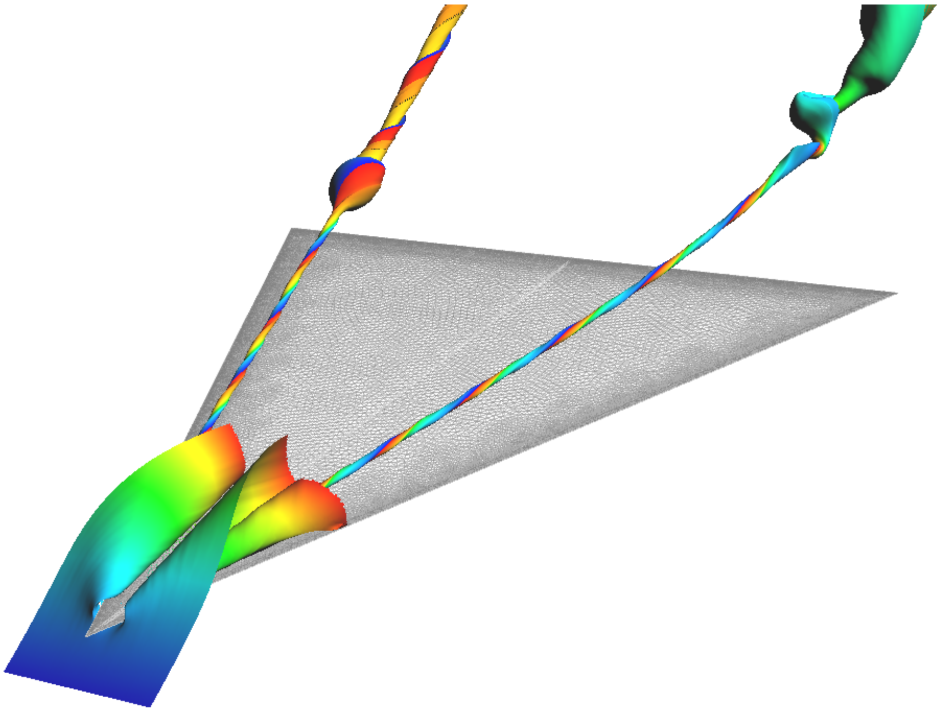
We have chosen the more straightforward approach of reformulating an iterative algorithm to the same purpose. Instead of building on a list of ribbons as the primary data structure, we maintain fronts as list of nodes. Advancement is then performed by iteration over the nodes in the front. By applying simple rules, appropriate actions are performed for every node based on the current state of the front at this node. The reader is directed to Appendix A.2 for a detailed description of the algorithm and data structure details.

Our stream surface algorithm has been implemented in C++ and incorporated into the FAnToM visualization system, through which all of the following experiments were performed.

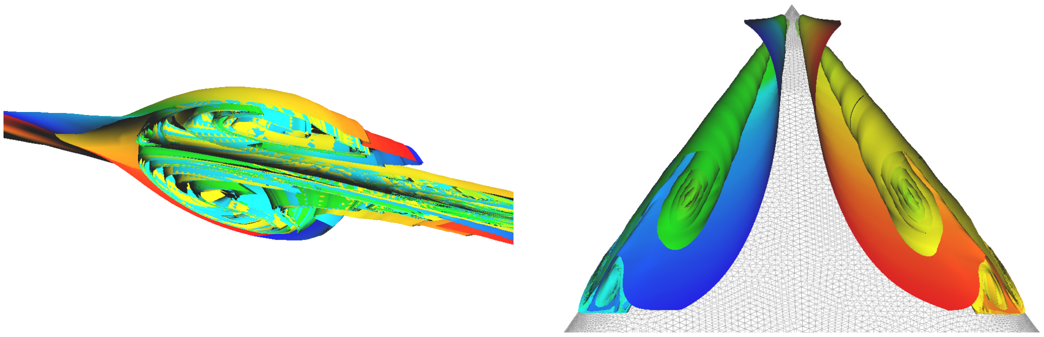
3.4.5 Experiments

We have tested our algorithm on several application datasets (cf. Section 1.4). Figures 3.7 through 3.9 provide an impression of the results achievable. In general, we have found the algorithm to perform remarkably well. It consistently produces high-quality surfaces, even in the presence of strongly distorting flow.

Throughout this thesis, we place special emphasis on the visualization and analysis of vortex breakdown (see Section 1.1.5). Figure 3.7(a) provides a stream surface based overview of the delta wing dataset. The formation of a single breakdown bubble on the right (in direction of flight) vortex is visible. On the left side, there is not a single bubble but instead a more chaotic structure. Although



(a) Overview



(b) Vortex Breakdown Bubble

(c) Vortex Formation at Wing Apex

Figure 3.7: Several stream surfaces computed in the delta wing dataset. (a) shows an overview with three stream surfaces. The surface passing the apex is colormapped according to t. Two surfaces wrapping around the primary vortex core lines are displayed using an s-colormap to illustrate spiraling of streamlines around the vortex cores. Breakdown bubbles are clearly illustrated. (b) Close-up of the breakdown bubble, cut open with a cutting plane, and revealing heavy recirculation. (c) Vortex formation at the apex, viewed from upstream (s-colormap).

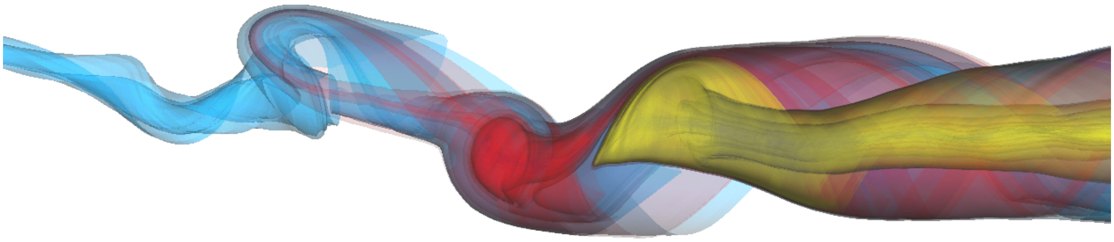


Figure 3.8: Semi-transparent stream surfaces in a topological context as nested separation surfaces of vortex breakdown saddle points.

the right bubble is completely enveloped by the stream surfaces, application of a simple slicing plane allows insight into the bubble and confirms the recirculation nature of the contained flow (Figure 3.7(b)). Remark that this image show only a single stream surface. To further analyze the more irregular left side structure, we have exploited the fact that this region shows several saddle points (cf. Section 4.6.2). Using our stream surface algorithm, we were able to compute the nested separation surfaces of these critical points 3.8.

3.4.6 Comparison to State Of The Art

In the following, we will briefly compare our algorithm to the method of Hultquist, which represents the de facto state of the art in stream surface computation. A direct qualitative comparison in terms of computational complexity cannot be given, since both algorithms rely on both complex heuristics and work on non-trivial input data.

To facilitate a quantitative comparison where performance and accuracy are concerned, we have implemented Hultquist's algorithm and applied both algorithms to several test cases, one of which we will discuss in the following. All datasets employed are introduced in Section 1.4.

Figure 3.10 depicts the result of such a comparison. The dataset under consideration is a high-speed train that is exposed to wind blowing at it from the side. The surfaces are started upstream of the train's nose and terminated at a fixed value of t. This test case is interesting in that the surface travels very close to the object boundary. Furthermore, vortex formation on the side of the train results in high curvature of the stream surface. For both algorithms, we manually chose the resolution parameters as low as possible (fewer triangles, conservative splitting) such that the surface is still correctly reproduced. As Figure 3.10 shows, we were able to achieve a significant reduction in number of triangles required. The original algorithm took 52 seconds to produce a surface of 185K triangles, while the improved method generated 110K triangles in 27 seconds and shows a marked improvement. While the results vary between test cases, both increased perfor-

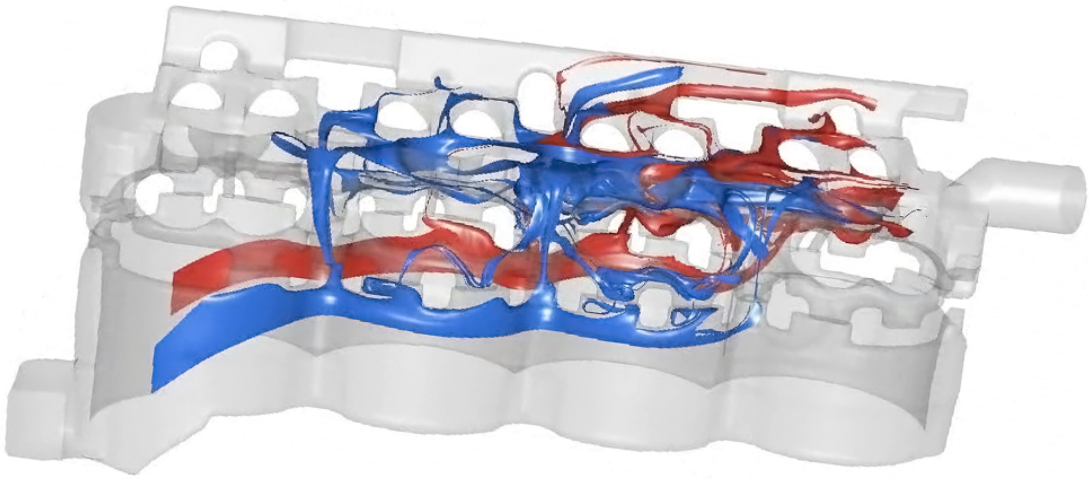


Figure 3.9: Two stream surfaces illustrate flow of water in a cooling jacket. The complex geometry with many holes are an ideal benchmark for our improved stream surface algorithm. While the surfaces are highly convoluted and the fronts are split many times, the result is of very good quality.

mance and visually improved surface approximations are observed when applying both algorithms to the same test cases. In more detail:

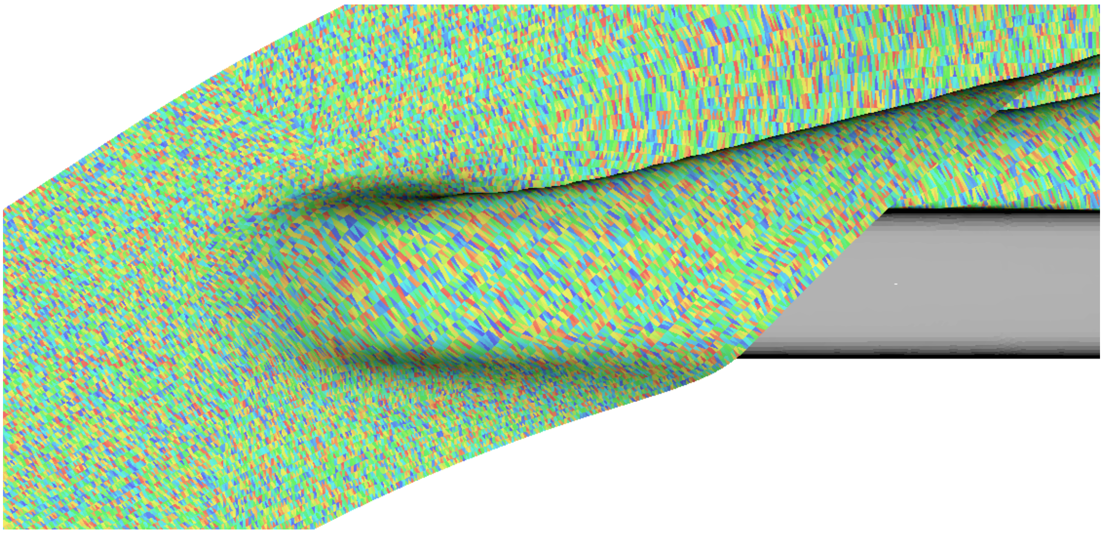
- More complex refinement criteria typically result in an increased number of streamlines that need to be computed. Our algorithm splits more aggressively, and merges less frequently.

- This increased computational cost is compensated by a marked increase in numerical integration performance. In comparison to fixed stepsize integration, adaptive timestepping requires much fewer evaluations of the vector field in typical situations. As computational effort is dominated by the latter, there is a marked increase in performance.

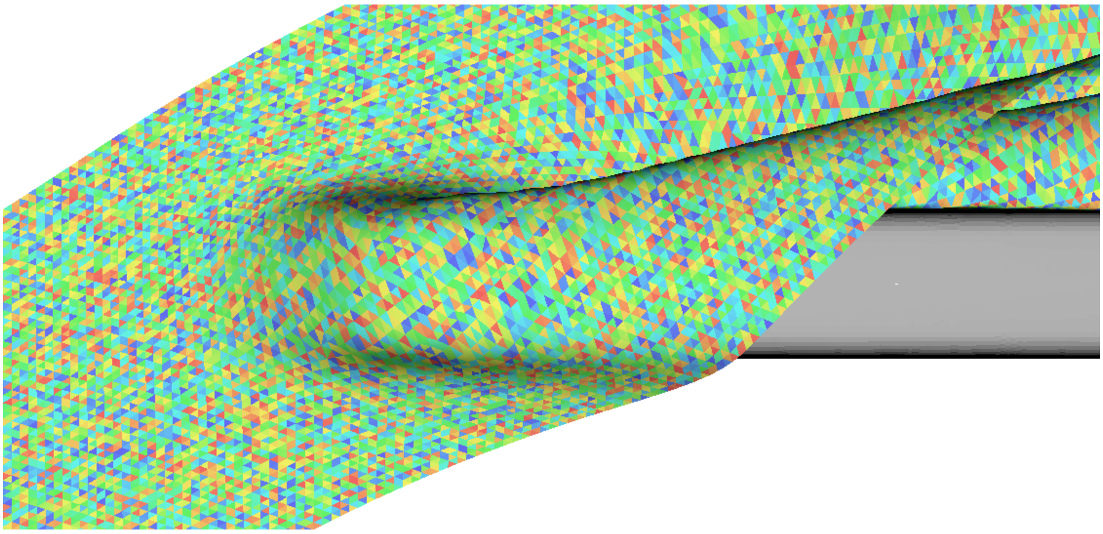
- Especially the curvature criterion results in better surface approximation in regions of strongly curved flow.

- Exact streamline termination enables the treatment of strongly curved flow boundaries or sharp edges. In contrast to Hultquist's algorithm parts of surface passing the closely to an edge are not discarded by our algorithm, at the price of extensive front splitting near the boundary.

Choice of Parameters Finally, some remarks are in order on a general strategy for choosing algorithm parameters. Δa is of greatest significance and should be chosen such that it matches the length scale of the smallest structures that are



(a) Hultquist's algorithm



(b) Our algorithm

Figure 3.10: A comparison of the same surface generated by both original and improved algorithms. Individual surface triangles are colored randomly. In the unmodified algorithm, triangle shape depends on flow magnitude. The improved method generates a much more balanced triangulation and generates much fewer triangles to achieve comparable results.

encountered by the stream surface. While this scale is usually not known a priori, in our experience only few iterations are required to find a good value for Δa.

The front segment lengths are controlled by $d_{\min}$ and $d_{\max}$. Choosing

$$d_{\min} \simeq \Delta a \simeq d_{\max}$$

is a canonical choice that ensures similar discretization behavior in both s- and t-directions. However, to avoid splits that are immediately followed by a merge and vice versa, we introduce hysteresis by letting

$$d_{\max} = 2\Delta a \qquad \text{and} \qquad d_{\min} = \frac{1}{2}\Delta a.$$

The front curvature control parameters are typically not dependent on the dataset under consideration. We typically chose

$$\alpha_{\min} = \cos^{-1}(0.9) \qquad \text{and} \qquad \alpha_{\max} = \cos^{-1}(0.95)$$

to achieve both adequate front refinement and avoid overeager successive splitting and merging.

3.5 Stream Surfaces as Visualization Building Blocks

In this section, we will show how stream surfaces can transcend their shaded surfaces nature in the visualization of complex flow data. In the remainder of this section, we will discuss several techniques that built on stream surface computation to achieve improved visualization results.

3.5.1 Enhanced Color Mapping

Stream surfaces are naturally endowed with a simple yet intuitive (s, t)-parameterization (cf. Section 3.1). While isolines of the s-parameter represent individual streamlines and indicate the direction of the flow on the surface, curves of constant t represent time lines. Section 3.4.5 already demonstrated the effect of corresponding color mappings. Löffelmann et al. [LMGP97] took this one step further and used (s, t)-coordinated for texture mapping purposes. Aside from these simple mapping choices, there are further possibilities facilitated by the role of stream surfaces as natural flow probes.

Field Resampling In engineering practice, it is standard procedure to represent such information by means of a cutting plane and apply a color mapping scheme for certain components of the dataset sampled on that plane. The very same approach is easily generalized to stream surfaces. Technically, a stream surface

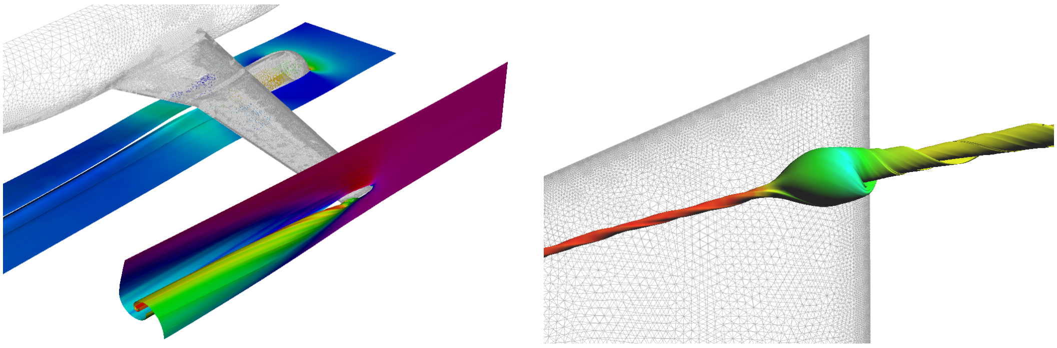

(a) pressure and vorticity colormaps (b) helicity colormap

Figure 3.11: Stream surfaces as a generalization of planar sections. Colormapping of dataset samples enhances the perception of the surfaces in terms of flow behavior.

is computed in the form of a triangle mesh, and quantities of interest can be evaluated at the mesh vertices. For each vertex, a color is then generated and used to interpolate throughout the triangles.

Figure 3.11 provides several examples using dataset attributes such as pressure, helicity and vorticity. In the left image, wake vortex formation is illustrated. The stream surface passing near the wing tip is colored according to vorticity, while second surface is passing through the engine and is split into three fronts. Its colormap illustrates pressure. A color mapping of helicity magnitude (right image) visualizes a strong correlation between increased helicity and the onset of vortex breakdown (see also Section 4.6.2).

Surface Curvature Given in the form of a triangular mesh, the geometry of a stream surface is readily submitted to curvature analysis using discrete curvature operators supplied, e.g. by Meyer[MDSB02]. Strong bending of the stream surface serves as an indicator of inhomogeneous flow behavior, and curvature information is useful in comprehending the three-dimensional flow pattern. Figure 3.12(a) provides an example using Gaussian curvature.

Stretching Making use of the canonical parameterization of the stream surface, we introduce a quantity ϑ that measures the convergence and divergence of streamlines in the surface in terms of length changes in time lines. It is defined as the mixed second derivative

$$\vartheta(s,t) := \frac{\partial}{\partial t}\frac{\partial}{\partial s}\mathcal{S}\Big|_{(s,t)}$$

and is readily computed by applying corresponding differential operators to the discrete surface representation. A high value of ϑ signifies strong divergence.

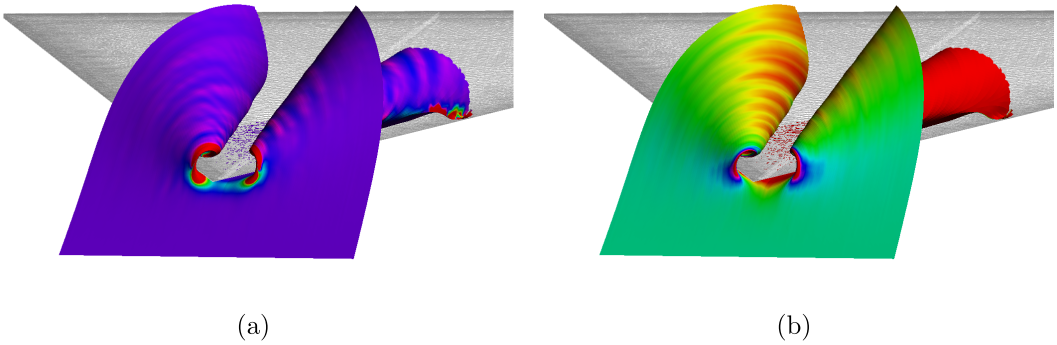

(a) (b)

Figure 3.12: Apex surface in the delta wing dataset, colored according to Gaussian curvature (left) and streamline stretching (right), revealing geometric properties of the stream surface.

Again, Figure 3.12(b) demonstrates how ϑ helps an understanding of the flow sheet represented by the surface.

3.5.2 Geometric Vortex Extraction and Verification

Vortices are at the heart of flow visualization, and many methods have been proposed to extract them from a dataset. We refer the reader to the discussion in Section 1.2.1 and the survey presented by Post et al.[PVH*03]. Since most methods fail in certain situations and can produce false positives, results usually have to be confirmed by manual inspection.

Manual Verification Stream surfaces can be helpful in this situation: by choosing a starting curve around a candidate vortex core, the resulting stream surface can then be displayed using an s-colormap that allows visual confirmation of swirling behavior. Figures 3.15(b) and 3.14 provides examples.

Obtaining the starting curve is a simple task. A point x_V on the presumed vortex core line is specified together with a radius estimate r_V. Then, surface computation is initiated from a circle whose center and radius are given by x_V and r_V respectively and which is contained in a plane perpendicular to the local flow velocity $v(x_V)$ (see Figure 3.15(a)).

The radius estimate r_V can be user-specified or automatically selected by matching a physical model of the circumferential velocity against the local flow field. We have obtained good results by using the *Rankine vortex* definition (cf. [Lug96] and Figure 3.13). We have found that a simple algorithm that determines the best match by comparing the model to successively increasing radii is sufficient to obtain a good estimate r_V (see [GTSS04]).

Automatic Verification Of special interest in this context is a geometric approach by Jiang et al.[JMT02]. Their method automates the verification process by seeding streamlines close to the vortex core line and observing their rotation with respect to the core axis. As a streamline progresses, its position is projected onto a plane perpendicular to the core line, and the number of rotations around the intersection of plane and core line is evaluated. If, for several streamlines, the number of rotations exceeds a given threshold, the vortex is assumed verified. Using stream surfaces, we can achieve a similar approach by replacing a small number of streamlines by an arbitrary number of s-lines on the surface. As an added benefit over Jiang's scheme, we are able to harness the adaptive resolution that generates an optimal surface description. Thus, stream surfaces can be used to automatically verify swirling motion patterns.

Vortex Core Extraction Vortex core lines are often extracted by a local matching of flow pattern against a preconception of swirling motion. For example, the in engineering applications very popular method of Sujudi and Haimes[SH95] performs such a pattern matching per cell of the computational mesh of a dataset. The result is a list of disconnected line segments, one per matching cell, that are indicative of short pieces of vortex core line segments contained in these cells. While Roth and Peikert's reformulation of this method in terms of the *parallel operator*[PR00] (see also Section 1.2.1) allows for the generation of longer, connected line segments, the resulting curves are not smooth. Furthermore, relying on second order derivatives, the results are often noisy and of bad visual quality (compare Figure 3.16). Furthermore, as mentioned above, false positives are not infrequent.

We therefore propose to use stream surfaces to generate a smooth and continuous representation of the vortex core line in the following fashion. As in the previous paragraphs, a surface is seeded on a circle around the core axis. Then, a sequence of N time lines on this surface is computed and discretized according to a pre-specified resolution. For the $i-th$ time line, we obtain a sequence of M points

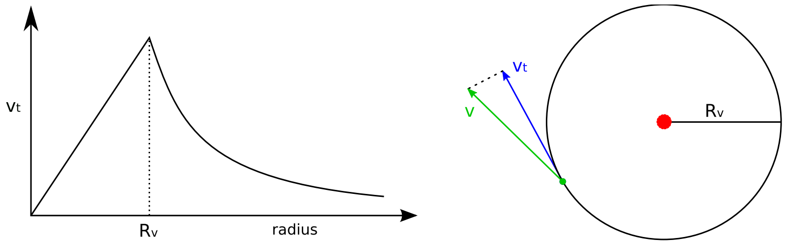


Figure 3.13: Circumferential velocity v_t in Rankine vortex model.

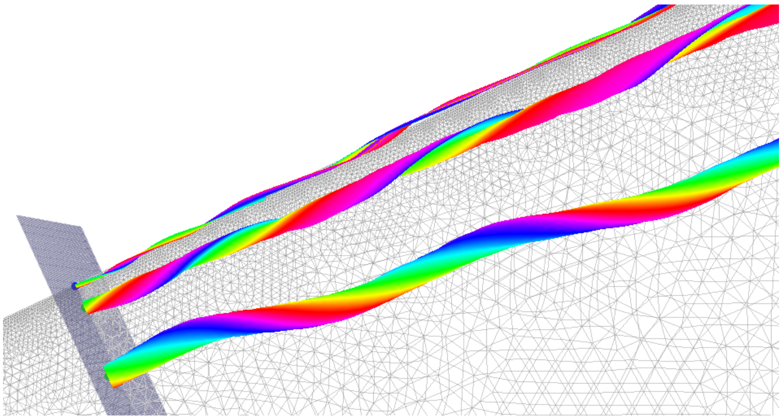


Figure 3.14: Visual conformation of swirling motion through using appropriately colored stream surfaces. Surface points along individual streamlines (lines of constant s) are assigned the same color. The surfaces are started on circles contained in the indicated plane.

$$T_j^i := \mathbf{S}(s_j, t_i), \quad \text{with } i = 0, \ldots, N-1 \text{ and } j = 0, \ldots, M-1. \tag{3.8}$$

We then approximate the center of rotation of the stream surface as the *center of gravity* of the polygon spanned by connecting T_j^i and T_{j+1}^i, where $j+1$ is understood modulo M to close the polygon. The center of gravity is then defined as

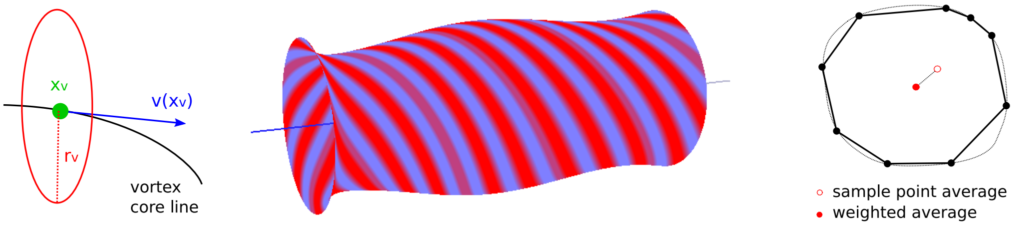
$$c^i := \frac{1}{2ML} \sum_{j=0}^{M-1} T_j^i \cdot (||T_j^i - T_{j-1}^i|| + ||T_{j+1}^i - T_j^i||), \tag{3.9}$$

where

$$L = \sum_{j=0}^{M-1} ||T_{j+1}^i - T_j^i||.$$

In other words, every T_j^i is weighted by the length of the two polygon edges connecting to it. These weights account for a possibly uneven sampling of time lines. In comparison to the unweighed center of gravity, c^i is a good approximation of the center of gravity of the time line itself (cf. Figure 3.15(c) for an illustration). Now, the polyline defined by the sequence of the c^i is taken as a new approximation to the vortex core line, which we call *gravity line*.

Given a good starting curve for the stream surface and sufficiently high N and M, gravity line vortex cores can significantly improve on the results of other schemes. In Figure 3.16, we demonstrate improved vortex core lines extracted by our scheme in contrast to the results of the Sujudi-Haimes method (which is described in Section 1.2.1) on a typical dataset. Remark that while the starting circle could be chosen manually, this would require several passes of trial and error



(a) Seeding curve (b) Resulting surface (c) Center of gravity

Figure 3.15: Vortex core verification and vortex core line extraction using stream surfaces.

if the location of a vortex core line is not exactly known. In this sense, gravity line vortex core extraction is to be understood as a post-processing tool that improves on the results of other vortex core extraction algorithms.

3.5.3 Miscellaneous Applications

Stream surfaces have also been employed as key visualization elements by others. For example, they form the basis for the *Saddle Connectors* approach of Theisel et al. [TWHS03]. They essentially make use of stream surfaces to compute intersections between stable and unstable manifolds of interconnected saddle points to determine the unique streamline that connects them. Thereby, the achieve a simplified topological description of the corresponding manifolds and connections.

Most recently, Tricoche et al. [TMJ06] have made use of our stream surface algorithm to visualize bioelectric fields in a torso around the heart region. The algorithm is applied directly to the electric field, and surfaces are seeded on iso-

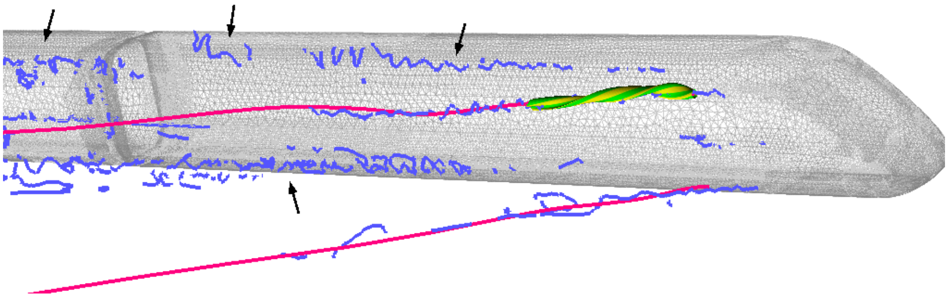


Figure 3.16: Stream surface based vortex core line extraction on the ICE train: Sujudi-Haimes output with false positives indicated by arrows (blue), vortex core lines computed as gravity lines (magenta), visual verification of the upper vortex using a wrapping stream surface, with color mapping to show the rotation around the vortex axis (yellow/green).

contours of the electric flux. The resulting surfaces visualize the geometry of the current induced by the dipole equivalent cardiac source and illustrate interconnections that exist between different connections of the heart surface. They also apply similar techniques to study the electrical fields in the brain.

3.6 Discussion

In this chapter, we have provided an overview of stream surface based visualization and presented our contribution in the form of a novel advancing front scheme that delivers significantly better results than its predecessors. It enables the reliable and fast computation of high-quality approximation to complex surfaces in large datasets. Furthermore, we have demonstrated the use of stream surfaces as powerful and versatile basic elements that advanced visualization techniques may be built upon and have pointed out several such examples. An application of our method to the visualization of application datasets has proven the illustrative abilities of stream surfaces. Especially the visualization of vortex breakdown with this technique has yielded heretofore unknown insight into the geometric structure of the breakdown bubble in the datasets under consideration.

There are several promising directions for future work. First and foremost, a generalization to path surfaces, i.e. integral surfaces in time-dependent vector fields, would allow further insight into these types of flows. However, there is a plethora of technical difficulties that preclude straightforward application of our algorithm in such settings. For example, the current front advancement scheme does not guarantee temporal locality of individual front segments. Hence, the entire dataset must be kept available to the algorithm. This poses a major obstacle for very large, time-dependent datasets. A second problem worth looking at is the complex algorithmic structure of the presented scheme. Since it is based on heuristic criteria, a rigorous mathematical analysis that would yield e.g. convergence proofs is not possible. One aspect of this problem is the simultaneous treatment of both correct mathematical approximation and generation of high-quality graphical output. A separation of these two processes seems promising with respect to an overall conceptual simplification. This could also lead to much improved triangulations that take into account a variety of global surface properties that are not accessible with the current implementation.

Chapter 4

Topology-Based Methods for Three-Dimensional Datasets

4.1 Motivation

The topological analysis of a vector field enables qualitative understanding of the dynamical system generated by the field in terms of the phase portrait. However, a graphical representation of the topological graph is necessary to convey the structural connections. While such a representation is straightforward for a two-dimensional domain of definition, with its topological graph consisting of points and lines, higher dimensions pose serious difficulties. There, the limit sets that constitute the topological graph may be of dimension greater than one. In three dimensions, for example, there are separation surfaces and closed invariant tori. Even in simple cases, a straightforward depiction of the topological graph is difficult as these surfaces, as opposed to curves, occlude each other.

Furthermore, a straightforward topological analysis of numerical three-dimensional vector field datasets has proven elusive. Critical points and closed orbits are extracted with relative ease [WS02], but closed invariant surfaces are very hard or impossible to compute. For vector fields describing technical flows, certain subsets of the boundary can appear as one-dimensional limit sets of the vector field. This phenomenon is know as separation or attachment, and a topological graph must be considered incomplete if it does not take these into account. For the typical case of an object embedded in a surrounding flow, it is not uncommon that the topological graph consists exclusively of these separation and attachment manifolds on the object boundary and their corresponding basin surfaces, and no critical points are present in the flow vector field itself. While the detection and extraction of such one-dimensional boundary limit sets has been treated successfully[KHL99, SJH05, TGS05], the corresponding basin surfaces are not reliably computable owing to technical difficulties. Moreover, they also contribute to the occlusion problem mentioned above.

In this chapter, we will take a different approach and generalize the topology tracking scheme of Tricoche et al.[TWSH02] to arbitrary dimensions, which we will then use this scheme as a basis to develop a novel method for the topological visualization of three-dimensional flow structures based on moving section planes. We obtain a powerful scheme that is ideally combined with volumetric visualization techniques to provide in-context visualization of both flow fields and related quantities. Furthermore, we apply the tracking directly to the analysis of the evolution of vortex breakdown bubbles.

We will next briefly consider previous work on topology-based visualization of three-dimensional flow fields and then present the core algorithm that forms the basis for the visualization methods presented in this chapter.

4.2 Previous Work

While the two-dimensional case has been treated extensively (cf. [LHZP05] for an overview and [Tri02] for a comprehensive introduction) and has proven a valuable tool in the analysis of two-dimensional flows, three-dimensional vector field topology has not been able to match this success so far, mostly due to the problems detailed above.

Among the first to study three-dimensional flow topology were Helman and Hesselink[HH91]. They proposed methods for detecting and classifying first order critical points based on the eigensystem of the Jacobian matrix. Simultaneously, Globus et al.[GLL91] presented a software system for visualizing the topological skeleton of three-dimensional vector fields. Their work was followed up by Nielson and Jung[NJ99] who introduced specialized tools to this purpose with increased efficiency and accuracy for the special case of tetrahedral meshes.

Theisel et al. attacked the problem of mutually occluding separation surfaces through the use of saddle connectors[TWHS03]. Their idea is based on saddle connections in the form of a single streamline that corresponds to the intersection of the stable and unstable manifolds of two saddles. They omit the corresponding manifolds from the topological skeleton and depict only the connecting streamline, in essence providing a further level of topological abstraction. They made use of stream surfaces to compute these manifolds explicitly and performed the intersection based on stream surface geometry and subsequently generalized this method to include boundary information as well [WTHS04].

4.3 Topology Tracking

As discussed in Section 2.7, dynamical systems that depend on a parameter are well studied. Concerning numerical realization, however, the theory has little to offer. In the following, we will present an algorithm that allows an analysis of the structural changes on a special subclass of vector fields, namely piecewise linear vector fields that are defined on simplicial meshes of arbitrary dimension. While this restriction seems enormous at first glance, application datasets resulting from numerical simulations commonly have this specific form, or are easily converted to it (cf e.g. [SBM*05, Wie03]). To be precise in meaning, we will require that the mesh is a simplicial complex.

4.3.1 Linear Interpolation

Definition 4.1. *A* simplicial complex $\mathcal{K}$ *is a set of simplices that satisfies the following conditions.*

1. *Any face of a simplex of* $\mathcal{K}$ *is also in* $\mathcal{K}$*.*
2. *The intersection of any two simplices* $\sigma_1, \sigma_2 \in \mathcal{K}$ *is a face of both* σ_1 *and* σ_2*.*

Furthermore, a simplicial n-complex $\mathcal{K}$ *is a simplicial complex in which the largest dimension of any simplex in* $\mathcal{K}$ *is* n*.*

For instance, a simplicial 2-complex must contain at least one triangle, and must not contain any tetrahedra or higher-dimensional simplices. We exploit this rigorous definition to guarantee that a piecewise interpolant on the mesh is well-defined and continuous. We will introduce its construction next. In the following, let $\mathcal{K}$ a n-simplicial complex embedded in $I\!R^n$, $n > 1$.

Let $S \in \mathcal{K}$ an n-simplex and let $p_i, i = 0, \dots, n$ its vertices, i.e. its 0-subsimplices. Then, the barycentric coordinates of a point $x \in I\!R^n$ with respect to the p_i are the coordinates

$$(\beta_0(x) + \cdots + \beta_n(x))x = \beta_0(x)p_0 + \cdots + \beta_n(x)p_n.$$

They are linear in x and obey the properties $\beta_i(p_j) = \delta_{ij}$ and $\beta_0 + \cdots + \beta_n = 1$. Barycentric coordinates are the natural coordinates relative to a simplex. When the β_i are not negative, x is contained in S. Next, let $v_i \in I\!R^n, i = 0, \dots, n$ vectors associated with the p_i. Then we define the linear vector interpolant over S using

$$v(x) := \beta_0(x)v_n + \cdots + \beta_n(x)v_n \qquad \text{for } x \in S.$$

By taking into account the union of all simplices of $\mathcal{K}$, we obtain a piecewise linear interpolant v on K. Moreover, v is continuous, since two adjacent simplices share the same vertices on their common face and whose weights are the only non-zero weights. However, v is not differentiable across faces and vertices. In each simplex, the Jacobian Dv is constant.

The linear vector field induced by Dv on any simplex S contains an isolated critical point if and only if $\det Dv \neq 0$. In this case, its location can be expressed in barycentric coordinates as a solution to the linear system

$$\begin{pmatrix} v_1 - v_0 \\ v_2 - v_0 \\ \vdots \\ v_n - v_0 \end{pmatrix} \begin{pmatrix} \beta_1(0) \\ \beta_2(0) \\ \vdots \\ \beta_n(0) \end{pmatrix} = v_0.$$

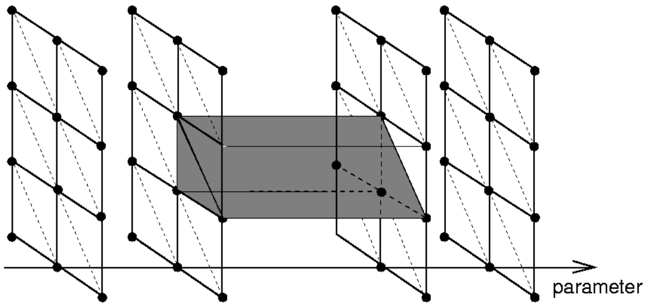


Figure 4.1: Product of two-dimensional complex and parameter space yields prismatic cells.

β_0 is not explicitly given by the system, but it is implicitly given determined by $\sum_{i=0}^{n} \beta_i(x) = 1$. If any of the $\beta_i(0)$ is negative, the critical point is located outside the simplex and is ignored. Remark that there can be at most one critical point per simplex, and its Poincaré index is either +1 or −1 (cf. Section 2.6).

4.3.2 Critical Point Paths

In the same sense that we applied linear interpolation to the representation of a vector field v, we will also use it to model its parameter-dependent cousin v_s. We hence assume the above setting with the modification that every vertex p_i of the simplicial complex $\mathcal{K}$ is associated to a sequence of vector values $v_i^j, j = 0, \ldots, N$ for an integer $N \geq 1$. In other words, the parametric vector field is given as a finite number of slices $v^j(x), j = 0, \ldots, N$ along the one-dimensional parameter axis. The corresponding ascending sequence of parameters is called s^j. This allows us to define a linear interpolant between two slices in the following way:

$$v_s(x) := \frac{1 - s^j}{s^{j+1} - s^j} v_s^j(x) + \frac{s}{s^{j+1} - s^j} v_s^{j+1}(x) \qquad \text{if } s \in [s^j, s^{j+1}]$$

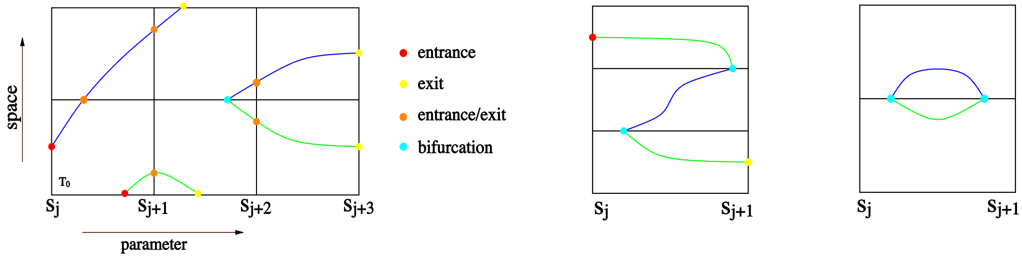
This construction corresponds to a tensor product $\mathcal{K} \times [s^{\min}, s^{\max}]$ that is composed of generalized *prisms* $S \times [s^{\min}, s^{\max}]$ where $S \in \mathcal{K}$ and $s^{\min} < s^{\max} \in \mathbb{R}$ (see Figure 4.1).

If $v_s(x)$ is found to contain a critical point for $s^j \leq s \leq s^{j+1}$ and x, its path under a change of s can be explicitly computed. Let S the containing simplex and v_i^j its associated sequence of vector values as above. Furthermore, let β_i its barycentric coordinates with respect to S. Then we define

$$w_i(s) := \frac{1 - s^j}{s^{j+1} - s^j} v_i^{j+1} - \frac{s}{s^{j+1} - s^j} v_i^j$$

and thus

$$v_s(x) = w_0 + \beta_1(x)(w_1 - w_0) + \cdots + \beta_n(x)(w_n - w_0) \qquad \text{for } x \in S.$$



(a) entry/exit/bifurcation events (b) reconstructible case (c) undetected case

Figure 4.2: Critical point paths in adjacent tetrahedra and entry, exit and bifurcation events.

A straightforward application of Cramer's rule and expansion of the determinants yields

$$\beta_1(s) = \frac{\det(-w_0(s),\ w_2(s) - w_0(s),\ \ldots,\ w_n(s) - w_0(s))}{\det(w_1(s) - w_0(s),\ w_2(s) - w_0(s),\ \ldots,\ w_n(s) - w_0(s))} =: \frac{b_1(s)}{q(s)}$$

with two polynomials $p_1(s)$ and $q(s)$ of degree n. A similar construction for $i = 2, \ldots, n$ provides us with the coordinates $\beta_i(s)$ of the critical point at parameter s. Brief computation reveals that the b_i and q are polynomials of degree n. By construction, $q(s) \neq 0$ if v_s has only isolated critical points. Naturally, if $\beta_i(\hat{s}) < 0$ for some $\hat{s} \in [s^j, s^{j+1}]$, the critical point has moved outside the simplex, in which case the local perspective offered by S is no longer valid. Therefore, the smallest root of $b_i, i = 0, \ldots, n$ determines the intersection of the critical point path with the cell faces. To determine if the path enters or leaves the cell, we evaluate the derivative of the corresponding coordinate with respect to s

$$\beta_i'(\hat{s}) = \left(\frac{b_i}{q}\right)'(\hat{s}) = \frac{b_i'(\hat{s})}{q(\hat{s})} \qquad \text{since } b_i(\hat{s}) = 0.$$

If $\beta'_(\hat{s}) > 0$, we speak of an *entry event*, otherwise of an *exit event*. The case $\beta'_(\hat{s}) = 0$ indicates that the critical point path touches a cell face and remains inside or outside, respectively. Furthermore, we do not distinguish between spatial and parameter domains and consider also call intersections of the path with the prism faces $s = s^j$ and $s = s^{j+1}$ entry and exit events (see also Figure 4.2).

Local bifurcations are in this setting limited to two types only. Of interest here is the qualitative behavior of a critical point during its motion through the interior of a prism. As a matter of fact, any bifurcation that would involve simultaneously two or more critical points present in the cell, either before or after the bifurcation point, is impossible because of the linear nature of the corresponding interpolant. Practically, such bifurcations can only be encountered on the common boundary of two neighboring simplices, as detailed in the following. The kind of bifurcations we are interested in here correspond to the transition of a critical point from one type to another. The spontaneous disappearance (or creation) of a critical point inside a cell is also impossible, since one would move from a situation where the local index is ±1 (a singularity is present) to a situation where this index equals 0 (no isolated critical point in the cell), which would locally break the consistency monitored by the invariant Poincaré index. Possible bifurcation involve the simultaneous zero-crossing of the real part of two eigenvalues. In two-dimensions, this type of bifurcation is called *Hopf bifurcation*, and only the transition from a sink to a source is possible except in very degenerate cases that involve a change of the global index (see also the discussion in [Tri02], p. 129). Over $I\!R^3$, there are more possible combinations, for example a transition from a saddle to a source (cf. the classification on pp. 26). Consequently, we only have to detect such bifurcations in between entry and exit events by a comparison of the type as determined from Dv on both entry and exit. If no change is detected, we assume that the critical point type remains constant along its local path segment. Otherwise, the roots of q can be used to determine the exact location of the bifurcation.

A local bifurcation that involves more than one critical point can occur on the simplex boundaries. On a face, it must involve exactly two critical points. Again, obeying the Poincaré index conservation limits the possible types to creation and annihilation of critical point pairs with opposite indices. Such bifurcations are detected if $\det Dv$ vanishes on an entry or exit event. Figure 4.2 visualizes these situations. There is one corner case, however (cf. Figure 4.2(c)). To learn this type of bifurcations, we must know about one of the involved critical points first. From this, we can reconstruct the complete picture. However, we typically perform detection of critical points on parameter slices. Therefore, if a critical point pair is created and annihilated without passing over a parameter slice s^j, it is missed by our approach. Typically, this is not a problem, since these structural changes are completely local and short-lived.

Bifurcations involving more critical points can theoretically occur on simplex edges or even vertices. Here the number of critical points is limited by the number of simplices adjacent to the edge or vertex. However, these cases are very unlikely, and especially so under numerical treatment. Therefore, we opt to ignore them in good conscience.

4.3.3 Algorithm

Having gathered the facts above, we are now in a position to state our tracking algorithm. As input we assume the simplicial complex $\mathcal{K}$, the sequence of discrete vector fields v_i^j, and the set of all critical points in all parameter slices

$$Z^j := \left\{ S \in \mathcal{K} : S \text{ contains a critical point of } v_s^j(z) \right\}, \qquad j = 0, \ldots, N$$

The algorithm then constructs the *structural graph*, i.e. the graph that consists of all entry/exit/bifurcation events connected by path segments.

Algorithm 4.2. *Path tracking for a single critical point*

Assume that a critical point z *is located in* $S \in \mathcal{K}$ *at parameter* $s^j < s < s^{j+1}$*. Then, compute the path in positive parameter direction:*

1. *Compute* b_i *and* q *for* S *and determine entrances and exits by computing all roots of* $b_i, i = 0, \ldots, n$*.*
2. *If there is no exit later than* s*,* z *exits* S *at* s^{j+1}*; the path is complete.*
3. *If there are exits in* S*, then* z *leaves* S *at the exit with the smallest parameter* s' *greater than* s*. If* Dv *on the path at* s' *is singular, there is bifurcation and the path completes on the face on which the exit lies.*
4. *Determine the adjacent simplex* S' *corresponding to the exit face.*
5. z *is now in* S'*. Assign* $S \leftarrow S'$ *and restart at 1.*

Following a path in negative parameter direction is achieved in a completely analogous manner. Both directions are completely equivalent. We use Algorithm 4.2 as a building block for computing the paths of all singularities present in two consecutive parameter slices s^j and s^{j+1}.

Algorithm 4.3. *Structural graph construction between* s^j *and* s^{j+1}

1. *Let* $B = \emptyset$ *be the set of bifurcations encountered in between* s^j *and* s^{j+1} *and let* $Z_0 := Z^j$ *and* $Z_1 := Z^{j+1}$*.*
2. *for every* $S \in Z_0$*: follow the path of* $z \in S$ *in positive direction*
   - (a) *if it ends in* S' *at* s^{j+1}*, eliminate* S' *from* Z_1*.*
   - (b) *if it ends at a bifurcation, add it to* B*.*
3. *for every* $S \in Z_1$ *(singularities not reached by paths from* s^j*): follow the path of* z *backward in time*
   - (a) *it must end at a bifurcation; add it to* B
4. *for all bifurcations in* B*: check if* B *has two paths connecting to it; if it does not, there must be an unknown critical point involved (Figure 4.2(b)). Follow its path forward or backward in time depending on whether the bifurcation is a creation or annihilation event.*
   - (a) *the path must end at a bifurcation; add it to* B*; goto 4.*

The algorithm essentially avoids multiple tracing of the same path by making use of the equivalence between forward and backward tracing (i.e. if a path extends from $t = 0$ to $t = 1$, we only need to trace it forward). The extra effort in step 4 is required because non-intuitive situations can occur (see Figure 4.2(b)). The end result is structural graph that completely describes the continuous structural variation of the vector field between the two parameter slices. Application to several successive slices is straightforward as it only involves connecting the paths from different slices according to which critical point they start/end at. We next present some examples of successful application of our algorithm in several situations.

4.3.4 Practical Concerns

Numerical datasets are often subject to noise, especially if the computations involve some kind of differentiation. It is common practice to apply smoothing operators to datasets in order to undo some of the damage done by previous computations. Numerical noise usually reflects in short-lived pairs of artificial singularities that exist in isolation and are not part of the dataset's structural evolution over time (Figure 4.2(c)). It can also occur that a path is "interrupted" by a pair of artificial bifurcations that enclose a path segment of very short duration (Figure 4.2(b)).

What seems a drawback at first can be turned into an advantage: instead of applying extensive smoothing the dataset we filter the resulting set of singularity paths by removing short paths that last less than e.g. one parameter interval. Fine-grained filtering can be applied on the structural graph directly and can be implemented in an efficient way by first removing edges that represent paths corresponding to short intervals and successively removing all isolated vertices. In our experiments, we found this method to be very effective in treating noisy data sets. It turns out that conventional smoothing does not significantly reduce the number of artificial critical points. It however affects the structure of the dataset in such a way that the structural evolution is obscured or changed. This is especially true for minimum/maximum tracking described in the next section.

4.4 Tracking Extremal Values of a Scalar Field

By applying the above approach to gradient fields of scalar quantities, we are able to track the evolution of minima, maxima and saddle points along the parameter axis, since these scalar critical points correspond to critical points in the gradient field.

Moreover, if one is only interested in, for example, minima, the resulting structural graph can be filtered to only include paths of type attracting node. Remark that spiraling behavior is not exhibited by gradient fields. We have applied this methodology in Section 4.6.

Aside from straightforward tracking of critical point paths and computing the structural graph of a vector valued dataset, our algorithm permits application in other contexts as well. We will next describe a visualization technique that is built on the tracking algorithm.

4.5 Moving Section Planes

In engineering practice, it is quite common to avoid occlusion problems in the visualization of three-dimensional datasets by employing planar sections. Typically, color mapping is used to depict scalar quantities of interest on the section plane. The central idea of the vector field visualization method introduced in this chapter is to extend these basic and widely used planar sections as tool for exploring flow volumes of stationary flows. The planes smoothly travel in a continuous way along curves that can be either obtained automatically by standard feature extraction schemes or directly provided by the user to explore a particular region. Building upon our tracking algorithm, we accurately track the vector field topology observed on the section planes. This allows detection and visualization of essential properties of a three-dimensional vector field in terms of section plane representation.

4.5.1 Plane Trajectory

More precisely, we start with a smooth curve $C(s)$, parameterized on $[0,1]$ through the domain of definition of a vector field v. Furthermore, let $N(s)$ a curve of unit vectors on the same interval. Then, for every s, there is a corresponding plane P_s defined by the point $C(s)$ and the normal $N(s)$. On this plane, define the restricted and projected vector field

$$v_s(x) := v(x) - < N(s), v(x) > \frac{N(s)}{||N(s)||} \qquad \text{for } x \in P_s \tag{4.1}$$

The plane is topologically equivalent to $I\!R^2$, therefore, the v_s can be treated as a planar vector field and its topological graph is extracted in a straightforward fashion. Furthermore, v_s is a parameter-dependent vector field, hence its structural changes can be tracked as the plane travels with increasing s (see Section 4.3).

For typical exploration tasks, the normals are chosen as the normalized tangents to $C(s)$, i.e.

$$N(s) = \frac{\dot{C}(s)}{||\dot{C}(s)||}.$$

However, there are specific situations that suggest a different choice of $N(s)$, for example if the section plane rotates on an axis. We will discuss several choices in Section 4.5.4.

4.5.2 Planar Resampling and Projection

Let $\Omega \subset I\!R^2$ a region and $v_s : \Omega \to I\!R^2$ a parametric vector field.

To apply the algorithms from Section 4.3, we need to construct a simplicial complex and a discrete representation of v^s thereon. We proceed with the construction as follows:

First, let

$$x^{i,k} = (\Delta x_0, \Delta x_1)^T, \quad i, k \in I\!N.$$

with $\Delta x_0, \Delta x_1 > 0$. Hence, the $x_{i,k}$ describe a uniform grid in $I\!R^2$. Then, there are vectors $e_s^0, e_s^1 \in I\!R^3$ and orthogonal to $N(s)$ such that the P^s is given in parametric form by

$$P_s(x) = C(s) + x_0 e_s^0 + x_1 e_s^1 \qquad x \in I\!R^2.$$

Since Ω is bounded, there are only finitely many points $x^{i,k}$ for which the corresponding sample point $P_s(x^{i,k}) \in \Omega$. We use the following simple triangulation scheme to generate a simplicial complex $\mathcal{K}_s$ from this point set.

Let $x^{i,k}$, $x^{i+1,k}$, $x^{i,k+1}$, $x^{i+1,k+1}$ four logically adjacent sample points. Then:

- If all four are inside Ω under P_s, then construct two triangles using these four sample points and add them to $\mathcal{K}_s$.
- If one of the four points is outside Ω under P_s, construct a triangle using the three other sample points and add it to $\mathcal{K}_s$.
- If less than three points are inside the domain, do not construct a triangle.

Then, for all vertices of $\mathcal{K}^s$, the corresponding associated vector field values are determined in accordance with Eq. 4.1.

Using this construction, we have constructed a simplicial complex $\mathcal{K}_s$. Hence, given a sequence

$$0 = s^0 < s^1 < \cdots < s^n = 1$$

and associated complexes $\mathcal{K}^j$ and vector values $v_j^{i,k}$, application of Algorithms 4.2 and 4.3 is almost straightforward. The only possible issue encountered is that these algorithms assume that the simplicial complex remains unchanged in between s^j and s^{j+1}. This problem, however, is easily solved by considering only the intersection $\mathcal{K}^j \cap \mathcal{K}^{j+1}$.

4.5.3 Section Plane Tracking

The moving section planes in combination with the tracking algorithm provide a vector field visualization primitive. Given a flow domain of interest, we can prescribe a trajectory, let the section plane travel along it, and track the resulting topological changes.

From a theoretical point of view, restriction to planar sections is not an exact tool in the sense that the topology of the projected vector field on the plane

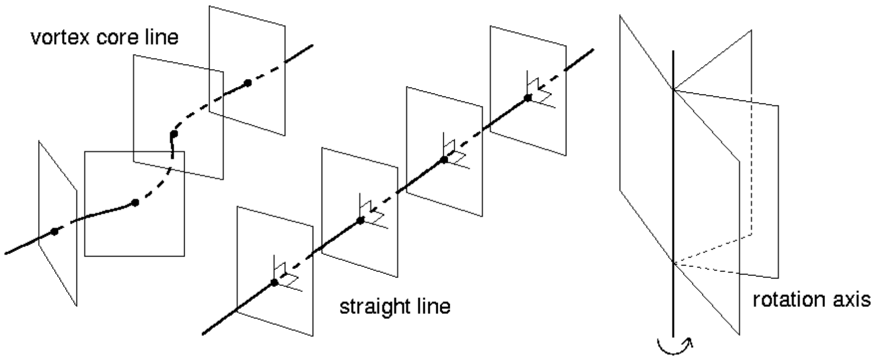


Figure 4.3: Different types of moving section plane trajectories

cannot take into account the behavior of trajectories that move away from the plane. While Poincaré's return map (which might be understood as an example of analysis on planar sections, cf. [GH83, Poi]) is a very successful tool for analysis of closed invariant manifolds, it is built on the fact that *trajectories return to it* and that by observing successive returns, one can learn about the trajectory behavior in a larger part of space. No such property exists in the case of planar sections as we defined them here.

However, the fact that hyperbolic structures dominate the behavior of a dynamical system gives rise to the following argument: Hyperbolicity (Definition 2.21) implies that the local behavior of trajectories can be decomposed into stable and unstable parts, both orthogonal to the flow direction. Therefore, if the planar section is roughly perpendicular to the flow, all information about the hyperbolic nature of a trajectory intersecting it is preserved in the projected vector field. While this argument cannot be put in exact mathematical terms, we demonstrate in Section 4.6 that it gives rise to a visualization method that is both powerful and intuitive, given some *a priori* information about the dataset that allows a selection of planes parallel to structures of interest.

4.5.4 Typical Trajectories

During our experiments (described below), there are several types of trajectories that follow from specific aspects of a dataset under consideration (cf. Figure 4.3).

Vortex Core Lines A natural idea is to select a vortex core line to serve as trajectory. These feature lines are the center of the swirling flow and are therefore natural candidates to capture the local symmetry of the flow. Various schemes are available that permit their automatic extraction. Essentially, if the section plane is parallel to the plane of rotation, the component of the velocity field along the vortex core line is discarded.

Straight Line A second type of trajectory is a user-selected straight line. This approach has two major applications. First, if the focus is on large-scale vortical structures, the mean flow direction in the corresponding region can be selected along with a convenient start position. We use this technique for visualizing the primary vortex of the Delta Wing dataset, see Section 4.6.2. The second application arises when one is interested in a single vortex but automatic vortex core extraction failed. In this case a straight line can be selected that approximates the vortex core line.

Recirculation Bubble Axis A last type of trajectory is directly fitted to the visualization of vortex breakdown. Because this phenomenon implies a dramatic change in the vortex structure, schemes for vortex core line fail or at best provide scattered segments. Fortunately, recirculation bubbles, though asymmetric in general, usually exhibit a central axis that constitutes their overall orientation. Therefore we explore such regions by rotating the cutting plane around this axis. More specifically, the outer boundary of a recirculation bubble is closely related to the existence of two stagnation points located at both ends. To visualize the flow structures enclosed in the bubble we select the segment connecting both stagnation points which naturally yields the required axis for our rotating cutting plane (see Section 4.6.2).

4.5.5 Plane Orientation

The orientation of the plane P_s, along with its trajectory through the volume, is a critical aspect and must therefore be chosen carefully. Depending on the method selected to traverse the volume, the orientation of the cutting plane along the path is either fully determined or must be chosen according to the local flow structure. The first situation occurs both during investigation of the recirculation bubble of vortex breakdown or when following a straight line to capture large-scale features. In the latter case the trajectory provides the plane normal. The second situation corresponds to the tracking of a single vortex, either along a pre-computed vortex core line or a straight line approximating its overall orientation. In that case neither the curve tangent vector nor the local flow direction can be considered as normal. When dealing with a vortex core line the inaccuracy in the extraction method results in an approximated position of the actual vortex core which can have a negative impact on the resulting normal value. The same holds true when approximating the curved, possibly complex path of a vortex by a straight line segment, as described previously. For these reasons we devise a way to compute a suitable normal at each point along the discrete path. Practically, the quality of a normal is evaluated with respect to the amount v_n of (normalized) flow crossing the corresponding plane:

$$v_n = \int_{P'} \frac{< v, n >}{||v|||}, \tag{4.2}$$

where P' is a small region around the considered point on the plane and n is the plane normal. To maximize this quantity we adopt the following iterative scheme: n is initialized as the velocity vector at the considered point along the line. Next we sample the vector field at a few locations evenly distributed around this point on the initial plane. We compute the mean vector of the normalized sample values and the corresponding value of v_n. The mean vector then replaces the current normal in the next iteration. The iteration proceeds until no significant improvement of v_n can be achieved.

4.5.6 In-Context Visualization using Volume Rendering

The sparse visual representation provided by the moving section plane topology is ideally suited to a combination with other, dense representations. Such dense visualizations include volume rendering of scalar fields. In the following, we examine a combination of both methods.

The motivation for using volume rendering within topological analysis analysis is twofold. On one hand, it provides a volumetric context to the topological structures extracted by the method described previously. This gives additional cues to understand the three-dimensional structure of the vector field being visualized. On the other hand the framework of multi-dimensional transfer functions is extremely powerful in allowing for the simultaneous and coherent representation of complementary derived quantities. We will demonstrate the usefulness of such an approach in Section 4.6 and consider specific aspects of volume rendering of flow-related quantities next.

Flow-Derived Quantities

The use of multi-dimensional transfer functions for volume rendering, combined with an interactive selection technique, was pioneered by Kniss et al.[KKH01, KKH02]. Multi-dimensional transfer functions are especially effective in visualization of complex flow structures because of the large number of flow-related variables traditionally used to characterize and quantify local properties of the flow vector field. As observed in previous work, having more than two domain variables in the transfer function greatly complicates the user interface, so we have restricted ourselves to two-dimensional transfer functions here. Thus, the exploratory visualization process involves finding a pair of variables that proves most effective in allowing important features to be displayed, and experimenting with transfer functions to highlight different structures and regions of interest within the flow domain.

Non-trivial flow features do not always have simple and universally accepted definitions in terms of flow-related variables. We point the reader to Section 1.1.5 for a discussion and commonly used variables. Thus, finding a transfer function which appropriately highlights a region of interest in the flow feature can be a fairly non-intuitive task. Interactive editing and selection of transfer functions has proven invaluably helpful in our experiments.

For the examples given below, we have used the *Simian* volume renderer[Kni] that features interactive user-guided transfer function editing through the use of *dual-domain interaction*[KKH01].

4.6 Visualization of Vortices and Vortex Breakdown

In the following, we present a number of visualization experiments based on the tracking scheme given above. The corresponding datasets are described in Section 1.4.

In general, the evolution of vortex breakdown bubbles can be linked to the appearance of *stagnation points* in the flow velocity field (cf. Section 1.1.5 and [SVL01]). These critical points in the instantaneous flow field are both spiral saddles. The upstream saddle is repelling, forcing the flow around the bubble, while the downstream saddle is attracting, inducing recirculation in between the two critical points (see also Figure 3.7(b) on p. 56). Treating time as a parameter enables a direct application of Algorithm 4.3 to the problem of tracking the location of these saddle points. We have examined two datasets that exhibit vortex breakdown with this method.

4.6.1 Rotating Lid Cylinder

Due to the high viscosity of the fluid and the high degree of symmetry the velocity field is of very good numerical quality. Furthermore, the flow comes close to being a standard model of vortex breakdown. To adequately visualize the structural graph, which in itself describes the four-dimensional evolution of critical points, we have reduced the problem to two dimensions by exploiting the intrinsic symmetry of the data, as the critical points are known to stay very close to the central axis of the cylinder.

The tracking results are of almost analytical quality (see Figure 4.4). The structural graph contains no noise at all, and filtering is not necessary. The breakdown bubbles come into existence in a saddle-saddle creation bifurcation, and vanish in a saddle-saddle annihilation. Figure 4.4 (green curve) shows these results. Primary and secondary breakdown each create a pair of stagnation points. Around timestep 1888, the two phenomena join, only to re-split at timestep 2458 and successively decay.

Besides the direct evolution of the bubble, we examined a number of related flow quantities that might indicate the cause of the breakdown. Several theories about vortex breakdown hint at a correlation with critical points in the acceleration field (cf. [Rüt05]). To this purpose, we have also computed the structural

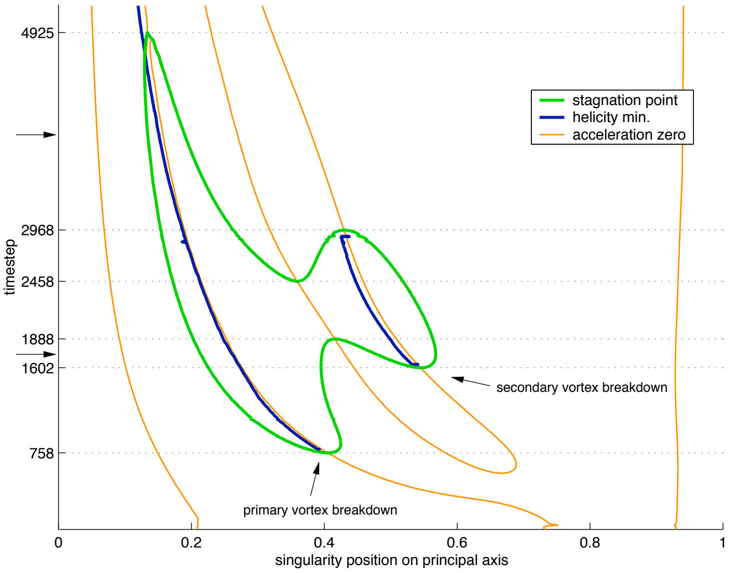


Figure 4.4: Structural graph of the rotating lid cylinder dataset. The green paths represent the stagnation points in the velocity field.

graph for the instantaneous acceleration field, which is defined as

$$a(x,t) = Dv(x) \cdot v(x).$$

Furthermore, we have applied extremal value tracking (Section 4.4) to a number of scalar quantities that are derived from the flow field. An interesting correlation appears between the creation bifurcation and a minimum of signed helicity. The latter is a quantity that encodes the scalar product of rotation plane normal and velocity. It is given by

$$\nu := \frac{< \nabla \times v, v >}{||\nabla \times v||||v||}.$$

As both acceleration and helicity are computed as derivatives of the original flow, some filtering was necessary to remove artifacts. Interestingly, there is a strong correlation, as the path of acceleration zeros (orange in Figure 4.4 lead right up to the velocity field bifurcations. Correspondingly, helicity minima appear at the same time as the bifurcation. Although the dataset had been under close scrutiny before, these interrelations of velocity, acceleration and helicity had gone unnoticed before. Our results were confirmed in [Rüt05].

We further note that the structural graph, if a dimensionality reduction is possible, provides an excellent overview of the structural evolution in a dataset. It is hence simple to quickly select timesteps of interest for further visualization. An example is provided in Figure 4.5, where we have selected two timesteps and visualized the breakdown structure using separation stream surfaces integrated

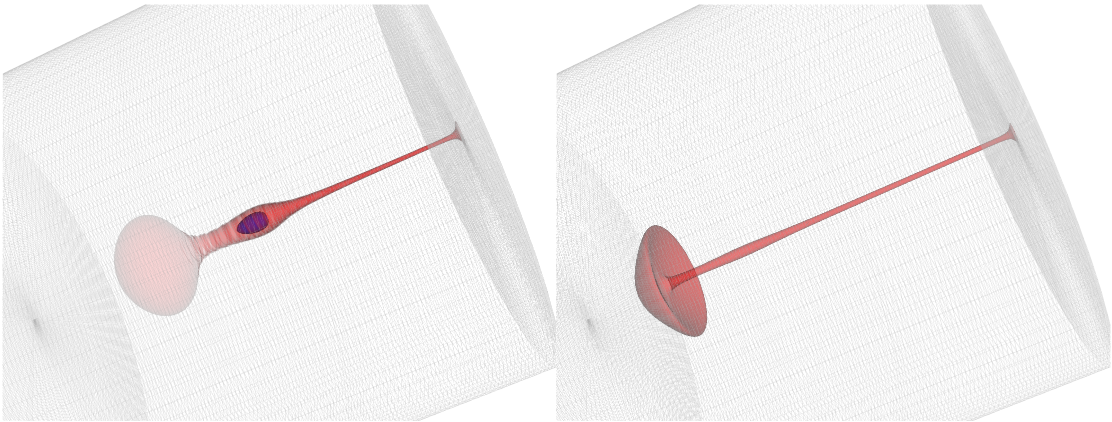


Figure 4.5: Two timesteps in the Rotating Lid Cylinder: timestep 1700 (left) and timestep 4400 (right).
.

from the unstable eigenplane of the repelling saddles: timestep 1700 shows both breakdown bubbles, while timestep 4400 shows the typical inverted mushroom shape of the decaying primary breakdown bubble.

In order to apply the moving section plane scheme to a visualization of the breakdown bubble structure, we let the plane rotate around the central axis of the cylinder, which coincides with the core line of the (single) central vortex. The resulting structures are depicted in Figure 4.6. The highly symmetric configuration of the breakdown bubble is visible, as is the nested secondary bubble. The recirculation centers appear as closed paths of critical points intersects the planes orthogonally.

4.6.2 Delta Wing dataset

For this dataset, we used the moving section plane method with the intention to study the vortical system above the wing. Figure 4.7(a) provides an overview of the flow structures above the wing that results from a plane traveling along the wing symmetry axis. The primary vortices are featured prominently, and the vortex axis results from the tracking of the corresponding singularities. Using the cutting plane orientation scheme described in Section 4.5.5 with the vortex core as section plane path, both secondary and tertiary vortices are visible (Figures 4.7(b)). Moreover, the planar sections reveals the nature of interaction between the three vortices (Figure 4.7(c)). They include the separation between the primary and secondary vortices and wing edge separation, i.e. the flow sheet that emanates from the wing edge and separates the flow above the wing from the surrounding flow. Both appear as a separatrix in the plane. These interactions are known in theory, but it is not possible to extract them using straightforward

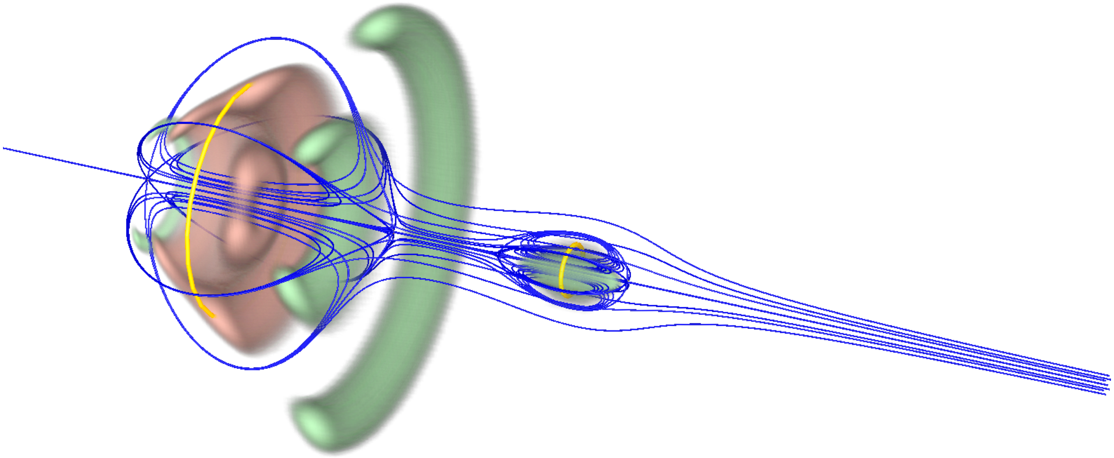


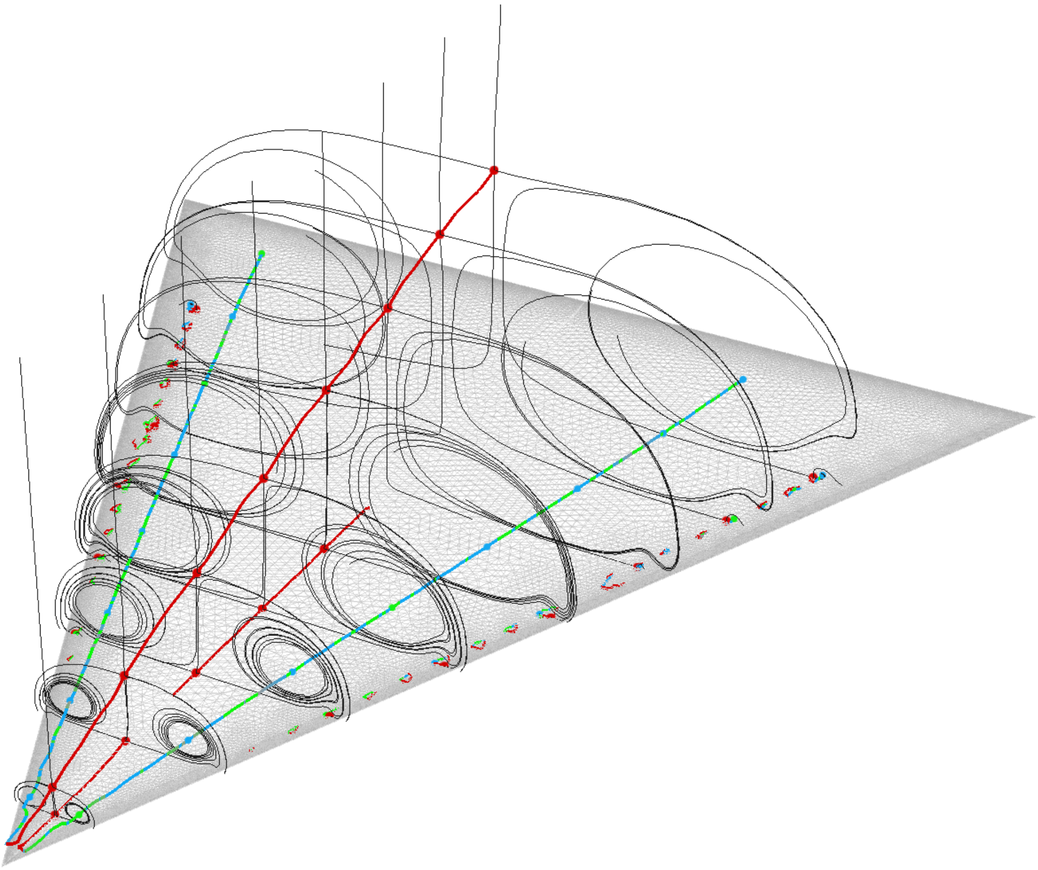
Figure 4.6: Structure of the vortex breakdown bubbles in the Rotating Lid Cylinder. Moving section plane topology is illustrated with stagnation points (red), critical point paths (yellow), and separatrices (blue). Volume rendering illustrates additional aspects and provides context to the topological visualization.

topological analysis. Hence, the section planes are an ideal tool for this type of situation and require little a priori knowledge about the dataset.

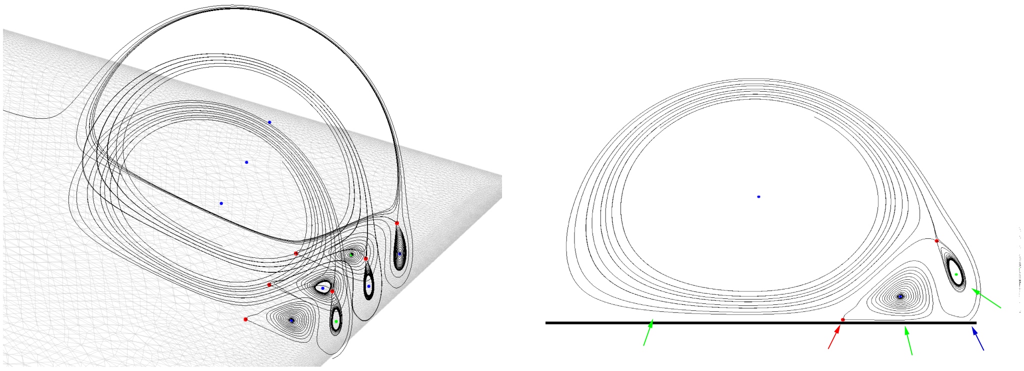
The structure of the breakdown bubbles above the wing is visualized by a moving section plane rotating on an axis coinciding with the overall vortex core line direction (see Figure 4.8). The right side breakdown is quite asymmetric in comparison to the very regular case in the rotating lid dataset, and the recirculation core (yellow) is strongly deformed. The red region corresponds to a volume rendering of regions of flow of negative axial direction, further enhancing the recirculation visualization. On the left side, the same visualization yields three different recirculation regions. However, volume rendering and topological extraction are not coincident. This is a strong hint at a very transient flow in this region that is only inadequately described by the stationary vector field we consider.

Furthermore, we have applied the critical point tracking to the three-dimensional evolution of the stagnation points on both sides of the wing. Again, by observing that the stagnation points move along the core lines of the primary vortices, we were able to reduce depiction of the structural graph to two dimensions. For this dataset, the results were more strongly afflicted by numerical noise and yielded a great number of paths in the structural graph. However, applying the graph filtering described in Section 4.3.4 allowed to simply and effectively discard numerical artifacts.

The structural graph of the right vortex breakdown region (cf. Figure 4.9(a)) clearly shows the evolution of the stagnation points as they move along the vortex core axis above the wing. Again, acceleration zeros and a helicity minimum seem to play a role in formation of breakdown, although the correlation is not as



(a) overview



(b) automatic orientation (c) vortex interaction

Figure 4.7: Moving section planes visualize topological structures above the wing.

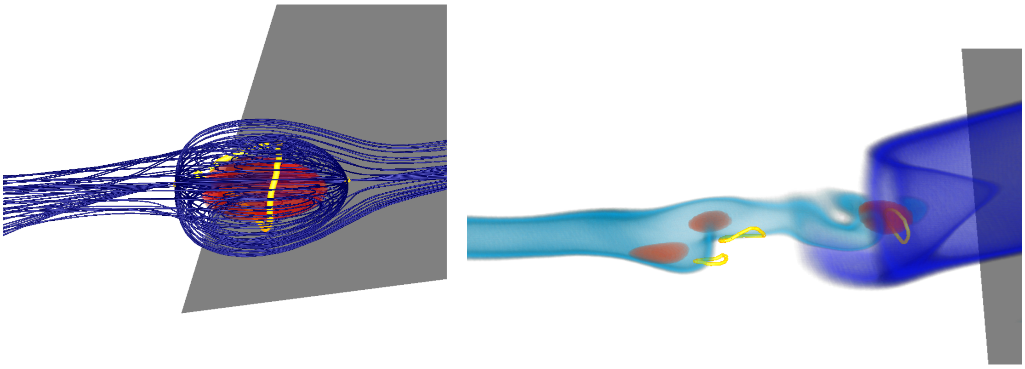

(a) right breakdown bubble (b) left irregular breakdown

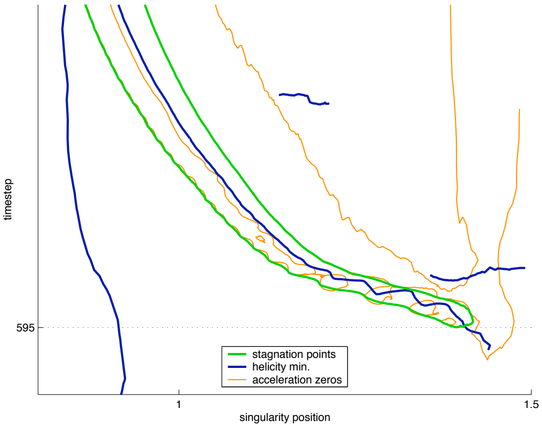
Figure 4.8: Volume rendering and in-context visualization of vortex breakdown above the delta wing.

obvious as in the rotating lid dataset. This is, in part, to be blamed upon the lack in resolution and the resulting numerical instability of differentiation. The left region (Figure 4.9(b)) is even more complex, and it is clearly visible how the stagnation points begin to oscillate and disappear around timestep 730, to be followed by what appears to be a sequence of short-lasting vortex breakdowns in different places. In this case, the structure graph helps in grouping the velocity field singularities that would otherwise just be isolated singularities in the field without any context. Using the structural graph as a guide to find interesting timesteps in the evolution, we produced Figure 3.8 (p. 57) from timestep 760 where the graph indicates the presence of two successive breakdown bubbles. Although this image conveys the basic structure of the breakdown bubbles, for an accurate interpretation the structural graph is needed.

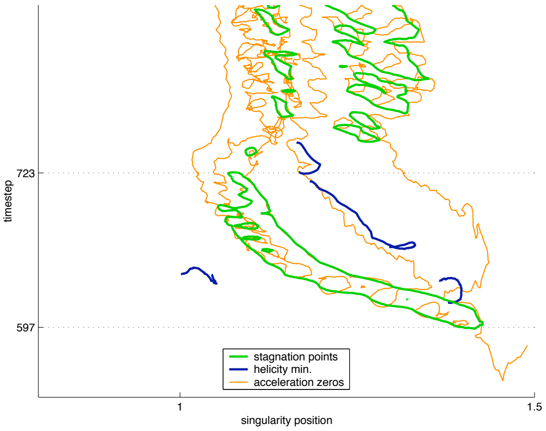
Figure 4.9(c) gives a direct comparison between the evolution of stagnation points on the left and right sides and the corresponding flow structures (displayed by stream surfaces). While the behavior is almost similar in the beginning, the left side quickly deteriorates and becomes strongly oscillatory. Here, the structural graph can provide for a direct qualitative comparison that is very hard to achieve by other means such streamlines or stream surfaces (cf. Figure 4.10).

4.7 Discussion

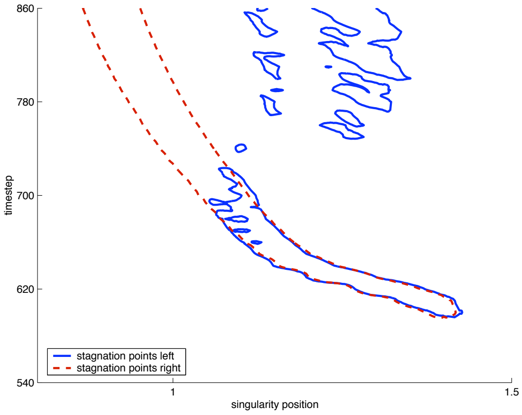
In the present chapter, we have provided a novel means of topological analysis of three-dimensional datasets by means of moving section planes [TGK*04]. Assuming reasonable a priori knowledge of the dataset under consideration, the section planes provide a valuable tool to examine and visualize flow structures in a topological context. Furthermore, we have developed a tracking algorithm for critical points in three-dimensional time-dependent flow fields[GTS04] and have pointed



(a) right region



(b) left region



(c) direct comparison

Figure 4.9: Structural graphs of delta wing vortex breakdowns.

(a) right breakdown (b) left breakdown

Figure 4.10: Stream surfaces depicting vortex breakdown do not provide a detailed insight into the breakdown structure.

out how it can be used in the visualization of time-dependent phenomena. Both visualization methods are based on a single underlying algorithm that is concerned with tracking of critical points in an n-dimensional setting.

We have successfully applied both methodologies to the analysis and visualization of vortex breakdown and obtained novel insight into the phenomenon, namely the appearance of helicity minima at the start of the breakdown process. Furthermore, we have provided a means of depicting the breakdown evolution in am abstract and schematic way that allows for a qualitative and quantitative comparison of such evolutions and the efficient topology-based investigation of very large transient datasets.

There is ample opportunity for further work: As the control of vortex breakdown is the ultimate goal, we feel that the method we have devised should provide an objective way to compare breakdown evolutions for example with respect to modifications of the wing design. Therefore, we would like to apply our method in such a context. Secondly, aside from tracking only the critical points, the question arises whether it is feasible to track their stable and unstable manifolds as well. While this has already been achieved by Tricoche [TWSH02] in the two-dimensional case, it seems quite difficult in higher dimensions.

Chapter 5

Lagrangian Techniques

The notion of *Lagrangian Coherent Structures* (LCS) and its quantitative assessment using the so-called *Finite-Time Lyapunov Exponent* (FTLE) provide a promising alternative that combines a well articulated theoretical basis and a physical intuition. Specifically, according to the underlying formalism, coherency in both steady and transient flows can be characterized in terms of repelling and attracting manifolds. Yet, despite the versatility of this approach and its ability to re-conciliate in a consistent formalism topological concepts and features of interest, its practical application is fundamentally hampered by the prohibitive computational cost associated with the necessary advection of a dense set of particles across a spatiotemporal flow domain.

In this chapter, we will propose an incremental, data-driven refinement algorithm that permits a reduction of this computational cost by significantly constricting the number of particle trajectories required to perform visualization and analysis based on FTLE and LCS. We exploit the coherence of particle paths to generate smooth approximations of the so-called flow map, from which the FTLE is computed. The presented approach enables high-resolution analysis of complex three-dimensional flows and facilitates insightful visualization and accurate assessment of coherence.

Furthermore, we show that it is often not necessary to perform full three-dimensional analysis. In analogy to the moving section plane concept of Section 4.5.2, given some a-priori knowledge about the flow field, it is often sufficient and in some cases even beneficial to consider FTLE on planar subsections, further reducing computational effort.

In the following, we will first introduce the concepts of FTLE and LCS and review previous work. We then proceed to present our incremental approximation scheme and visualization techniques based upon it and describe visualization experiments in both two and three dimensions. Finally, we conclude this chapter with a performance analysis of our algorithm.

5.1 The Finite Time Lyapunov Exponent

We will first present a formal introduction to the topics of FTLE and LCS. A derivation of FTLE is given in the context of general non-autonomous dynamical systems. We therefore recall the concepts introduced in Chapter 3. Detailed references on the topic are given in Section 5.3.

To find separatrices in time-dependent dynamical systems, one might take an approach similar to the autonomous case and consider critical points of the instantaneous vector field and try to grow these by computing trajectories corresponding to the stable and unstable manifolds. However, it can be demonstrated that separatrices in time-dependent flows are usually not connected to instantaneous critical points (although they might be located nearby). In general, instantaneous trajectories may diverge quickly from the actual (time-dependent) trajectories. Hence, instead of trying to directly compute separatrices, one tries to obtain them indirectly by considering the behavior of trajectories near such structures. This approach avoids an explicit a priori determination of limit sets, which is a formidable undertaking.

Considering a generic hyperbolic point and its associated stable and unstable manifolds, two trajectories that are initially located on either side of the unstable manifold will diverge from each other eventually in forward time. The same is true for the unstable manifold and backward time. In essence, this qualitative difference is the rationale behind the name "separatrix". It seems obvious that by measuring the divergence of trajectories, the existence of separatrix-like structures can be inferred. In more detail, forward and backward *stretching* can be measured. The Finite Time Lyapunov Exponent is the tool of choice to this purpose.

Let ϕ a differentiable dynamical system defined on a domain $I \times \Omega$ with $\Omega \subset I\!R^n$ an open region and $I \subset I\!R$ an open interval. In the context of FTLE, the map ϕ has historically been termed *flow map*, and we will adhere to this convention here. Recalling Definition 2.4, for an arbitrary point $(t_0, x_0) \in I \times \Omega$, the flow map

$$(t_0, x_0) \longmapsto x = \phi_t(t_0, x_0)$$

gives the point x on the trajectory through (t_0, x_0) after some time interval t. Note that ϕ has continuous dependence on initial conditions (Theorem 2.10). Therefore, an arbitrary point x_1 close to x_0 at t_0 will have a similar trajectory, at least locally in time. As t grows, the distance between $\phi(t_0, x_0)$ and $\phi(t_0, x_1)$ may change, and one would like to observe this distance. Thus, more formally, we consider a small perturbation δx_0 to x_0 and find for its evolution

$$\delta x(t) = \phi_t(t_0, x_0 + \delta x_0) - \phi(t_0, x_0) = D_x\phi_t(t0, x_0)\delta x_0 + O(||\delta x_0||^2)$$

by a Taylor series argument. Here, $D_x\phi_t$ denotes the Jacobian of ϕ with respect to x. Since $||\delta x_0||$ is small, we can neglect the $O(||\delta x_0||)$ term. The magnitude of the perturbation after time t is then given by

$$||\delta x(t)|| \;=\; \sqrt{(D_x\phi_t(t_0, x_0)\delta x_0)^2} \;=\; \sqrt{< \delta x_0, D_x\phi_t(t_0, x_0)\ ,\ D_x\phi_t(t_0, x_0)\delta x_0 >}.$$

Here, $D_x\phi_t(t_0, x_0)^*$ denotes the adjoint of $D_x\phi_t(t_0, x_0)$. For simplicity, we will denote

$$\Delta_t(t_0, x_0) := \left(D_x\phi_t(t_0, x_0)\right)^* D_x\phi_t(t_0, x_0)$$

Now, letting δx_0 vary over all directions, and applying the spectral matrix norm $|| \cdot ||_2$, one obtains

$$\max_{||\delta x_0||=\varepsilon} ||\delta x(t)|| \;=\; ||\Delta_t(t_0, x_0)||_2 \;=\; \sqrt{\lambda_{\max}(\Delta_t(t_0, x_0))}$$

for some $\varepsilon > 0$ small. Here, $\lambda_{\max}$ denotes the largest eigenvalue. Finally, we define

Definition 5.1. *The scalar function* $\sigma^t : I \times \Omega \to I\!\!R$ *given by*

$$\sigma_t(t_0, x_0) \;=\; \frac{1}{|t|} \ln \sqrt{\lambda_{\max}(\Delta_t(t_0, x_0))}. \qquad (5.1)$$

is called Finite Time Lyapunov Exponent

A comparison to Definition 2.52 confirms that this is indeed an appropriate name and that Eq. 5.1 is a finite-time analogue of the Lyapunov exponent. For the study of dynamical systems that are bounded in time, the latter is impractical due to its reliance on $t \to \pm\infty$ asymptotic analysis. However, the essential idea behind both concepts is identical.

Note that in Eq. 5.1 we used $|t|$ instead of just t for the normalization of σ. This accommodates trajectories that are both forward and backward in time.

5.2 Lagrangian Coherent Structures

We formally define Lagrangian Coherent Structures as ridges in the FTLE field. The ridges (or the ridge set) of a smooth function of several variables is a set of curves or surfaces whose points are, loosely speaking, local maxima in at least one dimension. Let f a function on $I\!\!R^n$. Clearly, a point x_0 is a *local maximum* of f if there is a neighborhood of x_0 such that $f(x_0) > f(x)$ for all x in this neighborhood. Relaxing this condition such that $f(x_0) > f(x)$ holds only for a $(n-1)$-dimensional neighborhood implies that there is one degree of freedom for points that violate the condition. We call this one-dimensional locus a *ridge line.* Note that an analogous construction is possible for local minima and *valley lines.*

More formally (following [EGMP94]), let $U \subset I\!\!R^n$ an open set, and $f : U \to I\!\!R^n$ smooth. Let $x_0 \in U$ and $\nabla_{x_0} f$ the gradient, and let $H_{x_0} f$ the $n \times n$ Hessian matrix of f at x_0. Denote by $\lambda_1, \ldots, \lambda_n$ the ordered eigenvalues of $H_{x_0} f$ with corresponding eigenvectors $e_1, \ldots, e_n$.

Definition 5.2. *The point x_0 is a point on a k-dimensional ridge of f, $k < n$, if the following conditions hold:*

1. $\lambda_{n-k} < 0$
2. $< \nabla_{x_0} f, e_i >= 0 \qquad$ *for* $i = 1, 2, \ldots, n-k.$

Next, we give a formal definition of LCS.

Definition 5.3. *For each t, a* Lagrangian Coherent Structure *(LCS) is a ridge of the scalar field σ^t of Definition 5.1.*

Since we assumed ϕ_t smooth, σ_t is smooth, and therefore LCS are smooth curves. However, this assumption is unrealistic for flow applications. [SLM05] have considered more general assumptions and shown that the concept can still be applied.

5.3 Related Work

An extraction of flow-relevant structures in terms of an instantaneous separation rate dates back to Okubo[Oku70] and Weiss[Wei91]. The use of FTLE as a means to characterize coherent Lagrangian structures in transient flows was introduced by Haller (who calls it Discrete Lyapunov exponent or DLE) in his seminal paper[Hal01b] in 2001. He presented FTLE as a geometric approach, contrasting it with an analytic criterion that he proposed simultaneously. Both approaches aim at characterizing coherent structures in terms of preservation of certain stability types of the velocity gradient along the path of a particle. This work followed previous papers by the same author, investigating similar criteria derived from the eigenvectors of the Jacobian of the flow velocity along pathlines to determine the location of LCS in the two-dimensional setting[Hal00, HY00].

This initial research generated a significant interest in FTLE and its applications to the structural analysis of transient flows in the fluid dynamics community, both from a theoretical and from a practical viewpoint. Haller provided a study of the robustness of the coherent structures characterized by FTLE[Hal02] and showed that they are well applicable even under approximation errors in the velocity field. In the same paper, he suggests to identify stable and unstable manifolds with ridge lines of the FTLE field. Shadden et al. provided a more formal discussion of the theory of FTLE and LCS[SLM05] in 2D. One major contribution of their work was to offer an estimate of the flow across the ridge lines of FTLE and to show that it is small and typically negligible. An extension of FTLE to arbitrary dimensions was discussed in[LSM06]. These tools have been applied to the study of turbulent flows[Hal01a, GRH06, MHP*06] and used in the analysis of vortex ring flows[SDM06]. However, the visualizations presented in these papers were chosen on a case-by-case basis, and no systematic investigation of different visualization types was considered.

5.4 Numerical Realization

In practice, a straightforward approach to computing σ^t for a given time-dependent vector field is to sample the flow map ϕ_t on a regular grid, corresponding to the domain of interest, and approximate $D_x\phi_t$ numerically, e.g. by finite differences. The flow map ϕ_t itself is computed by numerical integration in the original vector field (see Section 2.10). Although this is a conceptually simple procedure, the computational effort in terms of number of pathlines (i.e. trajectories in the time dependent system) makes this almost infeasible even for low sampling resolutions. A typical example with good resolution will require millions of flow map evaluations. In the following, we will present an algorithm that adaptively approximates the flow map for a given domain with a reduced number of actual pathline integrations, permitting a significant reduction in computation time with respect to the straightforward approach. Using certain convergence properties of subdivision schemes, we have formulated an algorithm for the incremental refinement of flow map approximations that estimates the local approximation quality and refines the approximation only when required. Before we approach the case of flow maps in two or three dimensions, we will first introduce the basic principle for the simpler functional case in one dimension.

5.4.1 Incremental Approximation of Maps

Assuming a smooth function $f : [0,1] \to I\!R$ on the unit interval, and a discretization of $[0,1]$ with $2^l + 1$ points

$$x_i^l := \frac{i}{h}, \qquad h = \frac{1}{2^l} \qquad\qquad i = 0, \dots, 2^l$$

for $l > 0$, the corresponding discrete samples of the function values at the sample points are

$$f_i^l := f(x_i^l), \qquad i = 0, \dots, 2^l.$$

The piecewise linear interpolant Lf^l of the f_i^l is given by

$$Lf^l(x) = \frac{x_{i+1}^l - x}{x_{i+1}^l - x_i^l} f_i^l + \frac{x - x_i^l}{x_{i+1}^l - x_i^l} f_{i+1}^l \qquad \text{for } x \in [x_i^l, x_{i+1}^l].$$

It is readily shown that Lf^l converges point-wise to the function as l increases. More specifically,

$$\lim_{h \to 0} ||f - Lf^l||_\infty \ \le \ \max_{[0,1]}(f'')h^2$$

holds. Therefore, the local convergence speed as $l \to \infty$ allows an estimate of the local variation of f.

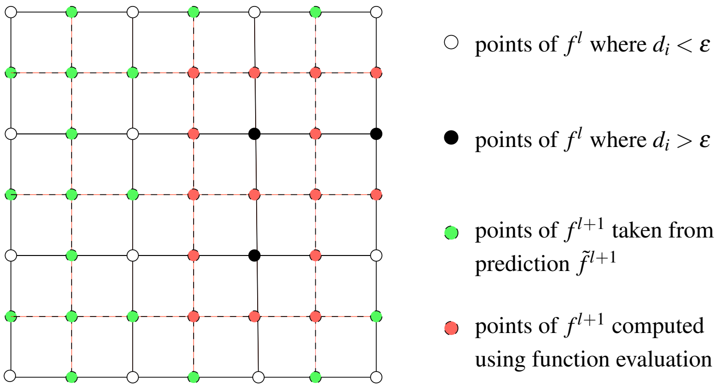

○ points of f^l where $d_i < \varepsilon$
● points of f^l where $d_i > \varepsilon$
🟩 points of f^{l+1} taken from prediction $\tilde{f}^{l+1}$
🟥 points of f^{l+1} computed using function evaluation

Figure 5.1: Incremental refinement form f^l to f^{l+1}.

We next define the subdivision operator S corresponding to linear midpoint subdivision. Its application to f^l yields a new sequence Sf^1 with $2^{l+1}+1$ points, i.e.

$$\begin{aligned}(Sf^l)_{2i} &:= f_i^l\\ (Sf^l)_{2i+1} &:= \frac{1}{2}\left(f_i^l + f_{i-1}^l\right)\end{aligned}$$

for $i = 0 \ldots 2^l$.

Assuming a discretization f^{l-1} of f exists on level $l-1$, we can construct a *prediction* $\tilde{f}^l$ of the discretization f^l on level l by simply letting

$$\tilde{f}_i^l = (Sf^{l-1})_i.$$

Comparing $\tilde{f}_i^l$ and f_i^l allows us to estimate the local convergence rate and hence the quality of the approximation. We define

$$d_i := ||\tilde{f}_i^l - f_i^l||$$

and construct a new approximation on level $l+1$ using

$$f_{2i}^{l+1} := f_i^l$$

$$f_{2i+1}^{l+1} := \begin{cases}\tilde{f}_{2i+1}^{l+1} & \text{if } \max\{d_i, d_{i-1}\} < \varepsilon\\ f(\dfrac{2i+1}{2^l}) & \text{otherwise}\end{cases}$$

In other words, if the local convergence rate is acceptable, we use the predicted value, or replace it with the actual function value otherwise. If ε is chosen small

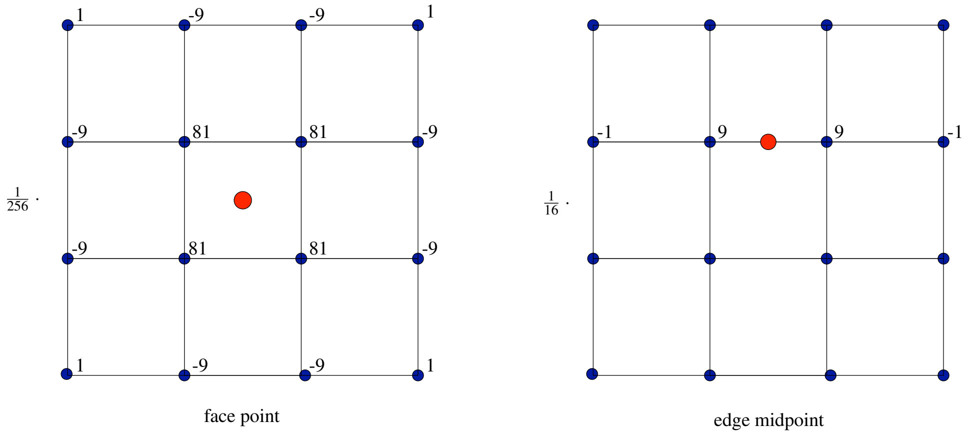


Figure 5.2: Subdivision stencils for the tensor-product four-point scheme in 2D. Left: stencil for face points. Right: edge point stencil.

enough and the second derivative of f is bounded almost everywhere, we construct approximation sequences that in the limit converge to the original function in a point-wise sense. For practical purposes, one is usually satisfied with a maximum discretization level $l_{\max}$.

Using linear subdivision in the above derivation is the simplest possible choice. Unfortunately, the resulting curve is not smooth in the sense that it has piecewise constant derivatives. To obtain approximations of a higher degree of differentiability, the linear subdivision can be replaced by any other type of interpolating subdivision scheme. We have selected the four-point subdivision scheme (cf. [PBP02]) that produces C^1-interpolants in the limit and reproduces cubic polynomials exactly. The corresponding subdivision operator S is given by

$$\begin{aligned}(Sf^l)_{2i} &:= f_i \\ (Sf^l)_{2i+1} &:= \frac{1}{16}\left(-1f^l_{i-1} + 9f^l_i + 9f^l_{i+1} - 1f^l_{i+2}\right).\end{aligned}$$

for $i = 0 \ldots 2^l$. For the boundary points, indices are clamped to 0 and 2^l.

For the functional case, four-point scheme subdivision is equivalent to computing the uniform Catmull-Rom spline interpolating f^l and evaluating it at the interval midpoints to generate the odd points in $\tilde{f}^{l+1}$. Although both descriptions are straightforward in the one-dimensional setting, we feel that extension to higher dimensions using tensor products is much simpler to describe in terms of subdivision. Moreover, the four-point scheme is simply a special case of Kobbelt's $2k$-schemes[Kob94] that produce C^{k-1}-interpolants. It is therefore quite simple to adapt our approach to produce higher degrees of smoothness. Care must be taken, however, in choosing an appropriate k, since higher-order schemes in general tend to overshoot. This is not a conceptual problem for the approximation algorithm, since these overshoots will be corrected during the computation. However, more function evaluations are required.

Here, the primary application of the presented algorithm is approximation of flow maps, which are then derived to find FTLE fields. Therefore, we have chosen the four-point scheme as a good compromise between required number of evaluations and smoothness mandated by the actual FTLE computation.

5.4.2 Efficient FTLE Computation

It is quite straightforward to generalize the above derivation to arbitrary dimensions and vector-valued flow maps. To illustrate the adaptive computation, we will describe the refinement algorithm for the two-dimensional case. An identical construction applies to the three-dimensional case as well.

Typically, the domain of interest Ω is not the unit square but rather an arbitrary rectangle positioned somewhere in the flow domain. To make the following description simpler, we introduce the parameterization $P : [0,1]^2 \to \Omega$ from the unit square to the rectangle and approximate the composition $\phi_{\mathbf{t}}(t_0, P(\cdot))$. This has no consequences for the algorithm, but must be kept in mind for later derivative computation are required. On the same note, the discretization need not have the same size on both axes of the unit square (this makes sense in the case of a strongly elongated rectangle), but for the sake of simplicity we will treat both directions uniformly.

Algorithm 5.4. *Incremental Approximation*

1. *Choose $\varepsilon > 0$, $l > 0$ (typically $l = 4$) and compute the initial flow map approximations f^{l-1} and f^l on the grid with $(2^l + 1)^2$ points and spacing $h = \frac{1}{2^l}$.*

   (Note: f^{l-1} is a subset of f^l and therefore entails no additional computations).

2. *Compute the distance sequence $d_{i,j}$ by point-wise comparison of $f^l_{i,j}$ and $f^{l-1}_{i,j}$*

3. *Subdivide the approximation f^l to obtain the prediction $\tilde{f}^{l+1}$ on level $l + 1$ (Fig. 5.2).*

4. *Set $f^{l+1}_{i,j} = \tilde{f}^{l+1}_{i,j}$ if $d_{i',j'} < \varepsilon$ for all grid neighbors that were present on level l;*
   else compute $f^{l+1}_{i,j} = \mathbf{x}(t, t_0, P(\frac{i}{2^l}, \frac{j}{2^l}))$. (cf. Fig. 5.1)

5. *If the maximum level is not reached, continue at 2.*

The end result of this algorithm is a dense approximation of ϕ_t that can then be used for visualization and analysis purposes. Figure 5.3 illustrates the adaptivity of our algorithm by showing the distribution of flow map evaluations for a specific

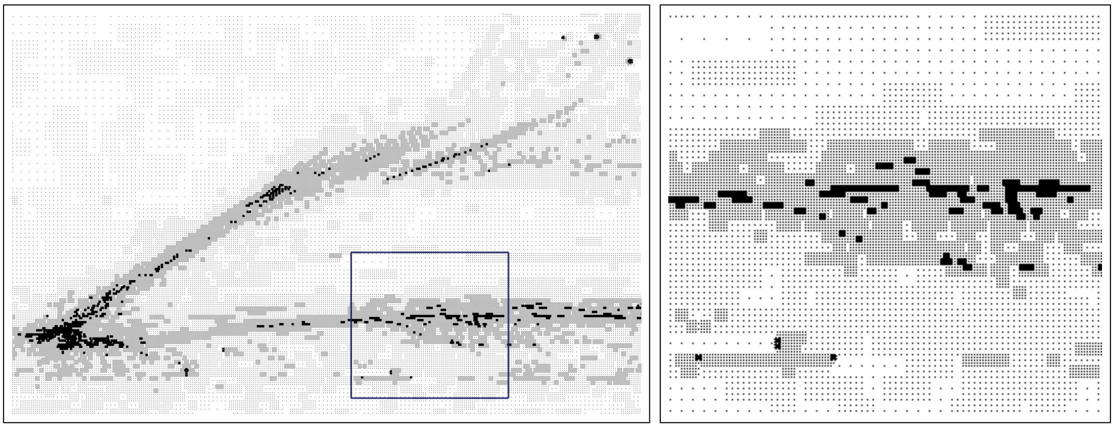


Figure 5.3: Visualization of the incremental refinement algorithm on the plane from Fig. 5.6. Black points indicate evaluations of the flow map ϕ_t. The right image shows a close-up of the framed region.

example. The ε parameter indirectly determines the accuracy of refinement. It directly refers to the distance between predicted and actual flow map result and thus depends on the scale of the dataset under consideration and the contained structures therein and must be chosen adequately. While this might seem problematic at first glance, our experiments have shown that this is not an issue in practice.

Implementation of the incremental refinement scheme is quite straightforward. We employ the Dopri5 scheme detailed in Section 2.10 to compute trajectories of ϕ_t.

5.4.3 Ridge Extraction

Numerically, ridge or valley extraction is a difficult task. Both are determined using the eigenstructure of the local Hessian matrix. This matrix of second order derivatives is the key obstacle in reliable ridges extraction because of the numerical noise introduced in the differentiation process.

Nevertheless, several algorithms exist to extract ridges explicitly from a given discrete scalar field. They exploit a reformulation of the ridge criterion (Definition 5.2) in the following form: On a d-dimensional ridge in $\mathbb{R}^n$, the eigenvectors e_i of the Hessian matrix corresponding to the $(n-d)$ smallest eigenvalues λ_i, $i = d+1, \ldots, n$, are perpendicular to the ridge. Furthermore, it is required that the derivative along these directions vanishes, i.e.

$$< \nabla_x, e_j >= 0. \tag{5.2}$$

Secondly, the ridge is a local maximum along these directions, hence

$$\lambda_i < 0 \tag{5.3}$$

is necessary. Valley lines are defined equivalently using the negative scalar field. We refer the reader to [EGMP94] for further details.

One-dimensional ridges in a two-dimensional domain are thus brought in a form that allows an application of the *Marching Cubes* algorithm[LC87] in the case of rectangular computational grids. For two-dimensional ridges in a three-dimensional domain, the lack of orientation of the corresponding eigenvectors prohibits this approach. However, the *Marching Ridges* approach of Furst et al.[FP98] and the crease extraction scheme by Kindlmann et al.[KTW06] are inspired by Marching Cubes and provide different ways to work around this limitation. Sadlo and Peikert[SP07] have applied these algorithms and demonstrated successful visualization of FTLE fields using ridge surfaces.

In our experience, however, depiction of LCS directly as ridge surfaces often fails as a visualization tools for complex three-dimensional datasets. Simultaneous representation of multiple surfaces often suffers from occlusion problems. A similar problem occurs in topological visualization of three-dimensional vector fields (see also Section 4.1). Furthermore, in the datasets we examined, a reliable automatic extraction of ridges was not feasible due to significant noise (cf. Figure 5.7). After all, the ridge criterion relies on a second-order derivative of σ, which is in itself a derivative of ϕ_t. Hence, these third-order derivatives introduce an instability into the ridge surface extraction that does not pay off in terms of effectiveness of visualization, and we have not applied it here. Remark however that we do use ridge line extraction on planar sections, as explained below.

5.5 Visualization of Coherent Flow Structures

The algorithm described above bridges the gap between the theoretical concept of FTLE and an efficient visualization of the resulting structures.

Because of its objective and fully automatic nature, the data-driven FTLE computation can be carried out in an offline manner. This yields a high-resolution, two- or three-dimensional time-dependent scalar field that quantifies the structural coherence of the flow. We now describe how the resulting information can be leveraged from a visualization standpoint to produce images that effectively support the insightful exploration of complex flow structures in practically relevant datasets. We organize our presentation with respect to the dimension of the considered FTLE field, which allows us to underscore the specific features associated with each setting.

For each of the following visualizations, we choose a fixed $t > 0$ and define forward and backward FTLE values

$$\sigma^+ := \sigma_t \qquad \text{and} \qquad \sigma^- := \sigma_{-t}$$

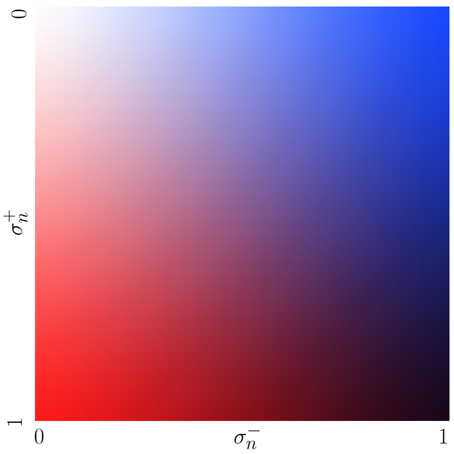


Figure 5.4: Transfer function for the $\sigma_n^+ \times \sigma_n^-$ space.

Furthermore, we compute the extremal FTLE values $\sigma^{\min} := \min\{\sigma^+, \sigma^-\}$ and $\sigma^{\max} := \max\{\sigma^+, \sigma^-\}$ over the domain of interest and introduce the normalization

$$\sigma_n^+ := \frac{\sigma^+ - \sigma^{\min}}{\sigma^{\max} - \sigma^{\min}} \qquad \text{and} \qquad \sigma_n^- := \frac{\sigma^- - \sigma^{\min}}{\sigma^{\max} - \sigma^{\min}}.$$

that maps the range of the $\sigma^\pm$ to the interval $[0, 1]$.

5.5.1 Three-dimensional FTLE

FTLE volume data constitute a direct and conceptually straightforward means to characterize the structural contents of a selected subregion of the flow domain. In essence, two complementary approaches can be used to visualize this information. The first one corresponds to displaying the whole three-dimensional FTLE data in a way that naturally underscores the fuzzy nature of individual coherent structures. This type of representation is powerful since it offers an overview of the flow that emphasizes its most prominent features. As such, it also provides a context for more targeted visualizations. In contrast, the second approach derives surfaces from the FTLE volume to create a skeleton-type representation that reduces the visual complexity of the visualization and explicitly computes a structural segmentation of the domain.

To transform the FTLE volume data into a volume rendering visualization we map σ_n^+ and σ_n^- to the axes of a two-dimensional transfer function domain (see Figure 5.4). As demonstrated in Section 4.5.6, this approach permits to identify interesting correlations between the different dimensions of the transfer function space and therefore allows for a very selective definition of features of interest. We use this idea to capture the shape of significant flow patterns and isolate them for display.

Note that we do not make use of an explicit depiction of LCS by means of ridge surfaces, for the reasons explained in Section 5.4.3. Rather, LCS are given implicitly through our choice of transfer function.

To generate the images in the following, we have used the *Simian* volume renderer[Kni].

5.5.2 FTLE on Planar Sections

An FTLE computation on a planar section provides a means to characterize the structural coherence of trajectories that intersect the plane. Compared to the three-dimensional case it also dramatically reduces the number of particles that must be advected and thus the overall complexity of the analysis. While in its original form, the definition of the FTLE is based on the 3×3 Jacobian $D_x\phi_t$ of the flow map, a planar sampling can only provide two spatial derivatives, resulting in a 3×2 Jacobian matrix. However, the matrix

$$\Delta_t = (D_x\phi_t)^* \, D_x\phi_t$$

is still a 3×3-matrix. Hence, the definition of the spectral norm $|| \cdot ||_2$ carries across to these non-square matrices, such that the FTLE derivation can be applied to planar samplings of the flow map as well. Although there is a conceptual difference in the sense that planar FTLE captures less information than fully three-dimensional FTLE computation, our experiments indicate that for many cases, these differences are insignificant (see also Section 5.6 and Fig. 5.5). Furthermore, observe that the orientation and the exact position of the plane origin do not directly impact the ability of the planar FTLE approach to capture coherency properties, as long as it is chosen in such a way as to intersect the full extent of the considered region of interest. This basic property is of great significance in practice since it guarantees the robustness of the corresponding analysis under small but arbitrary changes of the plane orientation and or location. In turn, this implies that a small number of planes can be selected a priori (e.g. by exploiting the overall symmetry of a considered dataset) to obtain the signature of all significant flow patterns.

Additionally, the analysis applied to the resulting scalar image offers an effective basis to perform the complex, time-consuming, and typically error-prone task of seeding integral curves and surfaces to probe a three- or four-dimensional flow domain. In the course of our experimentation we have investigated several techniques using this basic idea that we describe next.

Ridge lines The definition introduced for the 3D case applies similarly in the 2D case to characterize ridge lines. The numerical challenges associated with their extraction are somewhat mitigated by the lower dimension even if it remains difficult. In a recent paper, Mathur et al.[MHP*06] proposed an iterative scheme combining an integration along the gradient and a stop criterion based on Equation 5.2. In

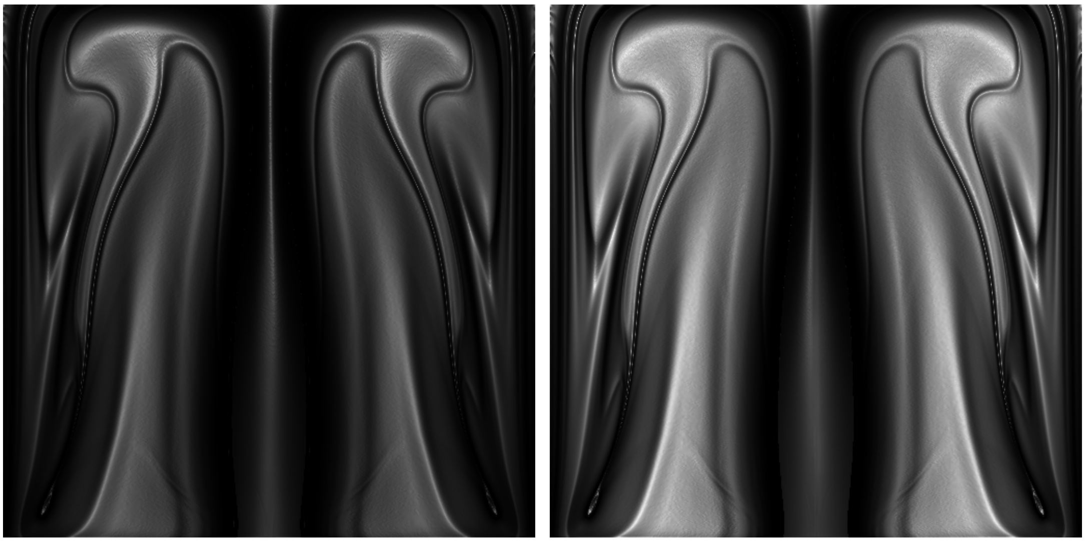


Figure 5.5: Comparison of planar FTLE vs. a section of three-dimensional FTLE in the Cylinder dataset (dark areas correspond to high values of σ_n^+). The chosen section is perpendicular to the lid and contains the cylinder axis.

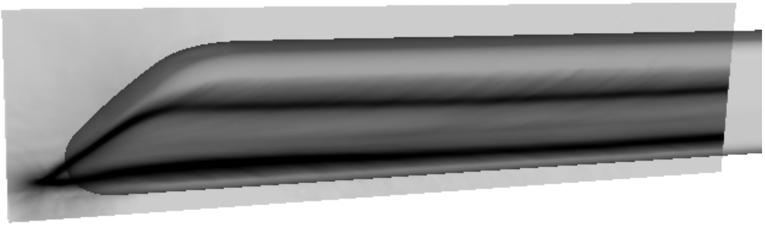
contrast the solution that we adopted in our work uses smooth and analytic reconstruction kernels for the derivatives of the FTLE field similar to [KTW06] and applies filtering criteria based on feature length and ridge strength (as measured by μ_{min}) to discard weak features and false positives. The resulting lines can then be used to seed a dense set of neighboring particles or a stream surface.

Stochastic particle seeding FTLE can be interpreted as a probability density function (PDF) for particle seed distribution. The fuzziness of the resulting representation matches naturally the fuzzy nature of flow coherence and the uncertainty involved in their computational characterization. The resulting pathlines correspond to trajectories with strong separation behavior. Technically, the seed points are selected as follows. σ^+ is sampled on a computational grid (uniform sampling in our case) with points $x_i, i = 0, \ldots, N$. We then compute a discrete cumulative distribution functions (CDF), i.e. a sequence $(p_j), j = 0, \ldots, N+1$ with

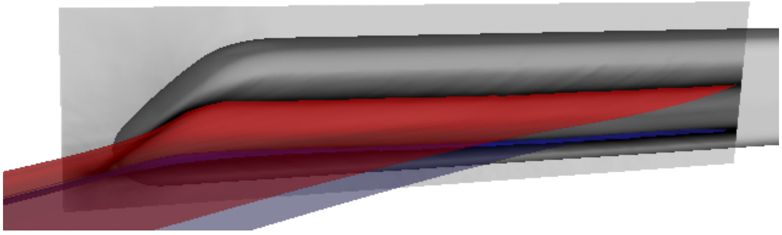
$$p_0 := 0, \qquad p_j = \frac{\sum_{i=0}^{j-1} \sigma^+(x_i)}{\sum_{i=0}^{N} \sigma^+(x_i)}$$

Clearly, the p_i induce a partition of $[0,1]$ into N intervals. To seed a trajectory, we generate a uniformly distributed sample $p \in [0,1]$ and determine j such that $p_j < p \leq p_{j+1}$. Then, the trajectory is seeded at x_j. The same construction also applies to σ^-. Figure 5.12 provides an example.

Image-based user interface Planar FTLE images provide the user with a look-up map over which interesting regions that may be difficult or impossible to



(a) Separation strength (σ^+).



(b) Stream surfaces integrated from ridge lines of σ^+.

Figure 5.6: ICE train: Left side separation structures are extracted using a cutting-plane FTLE analysis with a plane parallel and close to the side of the train. Extraction of forward FTLE ridges delivers starting curves for stream surface integration.

extract automatically can be manually and selectively identified by simple brushing to provide a PDF similar to the one mentioned previously but this time geared towards the specific focus of the analysis. This permits to reduce the visual complexity of the final image, to emphasize most prominent aspects in the data, and it provides an intuitive interface to do so.

5.6 Visualization Results

In the following, we present some results from our experiments with the visualization techniques introduced above. The datasets under consideration are detailed in Section 1.4.

5.6.1 Choice of Parameters

Until now, we have not discussed a specific choice of trajectory length, i.e. we have not selected $t > 0$ for σ^+ and σ^-. In general, an optimal selection of t is the subject of an ongoing debate. Choosing t small with respect to the characteristic time (cf. 1.1.1) emphasizes structures that are coherent on such short time scales. However, these structures might not be meaningful for the overall structure of the

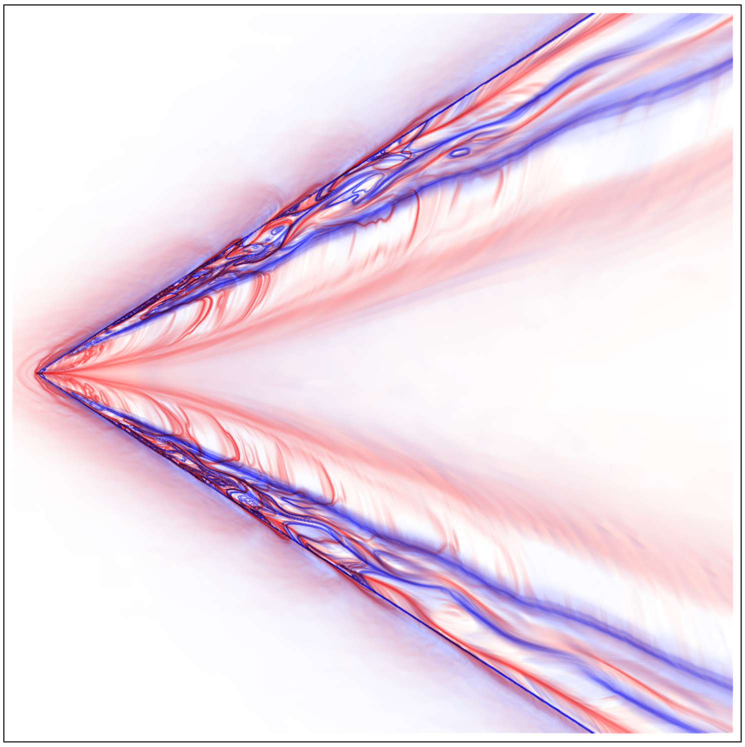


Figure 5.7: Close-up of the delta wing tip: planar FTLE reveals highly non-trivial separation (red) and attachment (blue) structures and demonstrates the complexity of flow structures in modern CFD datasets.

flow and disappear if t is increased. Conversely, increasing t beyond the characteristic time usually does not change the apparent structures much. Therefore, setting t greater than the characteristic time seems a good choice. The resulting coherent structures for different choices of t are demonstrated for a simple example (von Kármán vortex street) in Figure 5.8.

Through the refinement parameter $\varepsilon > 0$ required as input to Algorithm 5.4 essentially controls the tolerated curvature of the approximand. Therefore, it is independent of specific datasets. In the following, we have chosen $\varepsilon = 10^{-3}$.

Rotating Lid Cylinder The cylinder dataset is an ideal benchmark dataset. Due to its high degree of symmetry with respect to the central axis, we have used it to study the differences between planar FTLE and a slice of the three-dimensional FTLE field (cf. Section 5.5.2). Fig. 5.5 shows a direct comparison. While the three-dimensional FTLE slice provides more contrast, the observed structures are qualitatively identical.

ICE train Fig. 5.9 shows a volume rendering of $\sigma^+ \times \sigma^-$ in a box around the nose of the front wagon of a high-speed train. The images permit a good overview

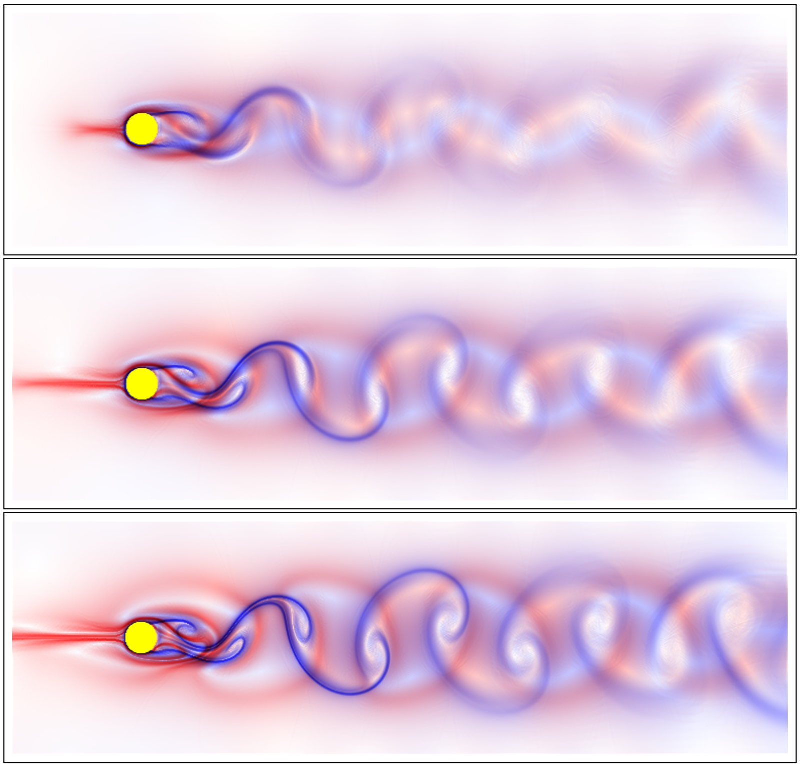


Figure 5.8: Effect of variation of integration length on the resulting structures. Colormap of $\sigma^+ \times \sigma^-$. Integration times are 0.25 **(top)**, 0.5 **(middle)**, 0.75 **(bottom)**.

of the prevalent separation and attachment structures close to the surface. On the right side, vortex shedding at the top edge of the wagon can be inferred. On the left side, the separation structures show that a significant volume of air streams above the top of the wagon and contributes to the vortex shedding on the right side. To obtain a more detailed depiction of these separation lines, we have computed a high-resolution planar FTLE section along and close to the left side of the train (Fig. 5.6(a)). From this section, we extracted ridge lines of forward FTLE and used them as starting curves for stream surfaces, see Fig. 5.6(b). Using this approach, it was possible to quickly achieve a detailed picture of these separation structures.

The stream surfaces were computed using the algorithm given in Section 3.4.

Delta wing From a visualization point of view, the main challenge in this dataset is the complexity of the flow above the wing with its nesting and interacting coherent structures. Fig. 5.10 provides and overview. The outermost layers occlude most of the inner structures, therefore, auxiliary techniques such as clipping planes of volume cropping have to be applied to obtain a good depiction

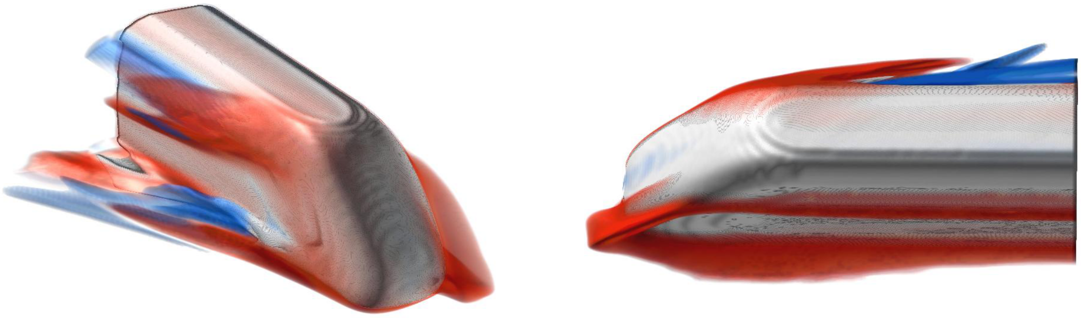


Figure 5.9: Volume rendering of $\sigma_n^+ \times \sigma_n^-$ around the nose of an ICE train reveals separation and attachment structures.

of the flow. Overall, however, the insight provided by 3D FTLE visualizations is insufficient. To reduce visual complexity, we have again applied the planar FTLE technique. Fig. 5.12 shows very short streamlines stochastically seeded in regions of high forward and backward planar FTLE directly above the wing, giving a very good illustration of alternating surface separation and attachment lines. The separation surfaces emanating from these lines are further examined using planar FTLE on an plane perpendicular to the wing axis (cf. Fig. 5.13). The resulting 2D image (Fig. 5.14 shows the impressive complexity of the flow above the wing. Several vortices are clearly visible, and the separation surface emanating from the wing edge exhibits a rolling-up type structure. To generate three-dimensional visualizations from this image, we have employed used-driven fuzzy pathline seeding (see painted areas in Fig. 5.14 and corresponding streamlines in Fig. 5.13). This allows a further study of the observed structures. For example, we observed a point of very high FTLE in the center of the right primary vortex (yellow dot). The corresponding streamlines show the typical vortex breakdown bubble strength and explain this point as the strong separation generated by the upstream swirl saddle that usually accompanies the breakdown bubble.

5.7 Performance Analysis and Discussion

To study the performance of our adaptive approximation, we have examined a number of test cases (cf. Table 5.1) using a subset of the datasets described in Section 1.4. Overall, the incremental refinement approach allows a reduction of computational effort by a factor of 5 to 10. In our experiments, we have found that in the initial exploration of a dataset, the region of interest is chose such that it contains a large volume w.r.t. to typical structure size. This is mostly because the location of these structures is not a-prior known. Hence, this is the case that profits most from the incremental approximation. As the region of interest is refined to the scale of the actual structures, the benefits are reduces. We therefore believe that our algorithm is ideally suited to the exploration of datasets, where

Dataset	**time-dep.**	**Resolution**	#PL	rel. l_2-error	rel. l_∞-error
delta wing	no	1024×1153	12%	$5.32 \cdot 10^{-5}$	$7.38 \cdot 10^{-3}$
delta wing slice	no	2048×1024	7%	$6.64 \cdot 10^{-5}$	$9.18 \cdot 10^{-3}$
Cylinder	*yes*	2048^2	9%	$6.79 \cdot 10^{-8}$	$9.12 \cdot 10^{-6}$
Cylinder	*yes*	128^3	14%	$9.43 \cdot 10^{-7}$	$1.33 \cdot 10^{-1}$
High-speed train	*no*	$289^2 \times 65$	5%	$3.81 \cdot 10^{-4}$	$3.37 \cdot 10^{-1}$
delta wing box	*no*	$257 \times 321 \times 65$	25%	$9.59 \cdot 10^{-6}$	$2.6 \cdot 10^{-3}$

Table 5.1: Incremental approximation test results.

it can provide significantly reduced computation times.

Furthermore, our numerical experiments show that as the refinement parameter ε decreases, the relative errors with respect to the non-adaptive computation tend towards zero. Therefore, we are confident that our ideas are applicable to dataset of even higher complexity than the test cases presented above.

5.8 Discussion

In this chapter, we have presented an incremental algorithm for the computation of flow map approximations and FTLE fields that allows for a significant reduction of computational effort, allowing for for high-resolution visualization and analysis of FTLE and the corresponding Lagrangian Coherent Structures. In all our experiments, our algorithm behaved stable and delivered good performance improvements. Moreover, we have demonstrated the power of FTLE-based visualization methods on several examples from relevant application areas. Furthermore, we have shown that even for 3D flows, 2D FTLE analysis and derived methods such as particle seeding yield insightful results and can further reduces computation times and facilitate user-guided exploration of datasets.

There are many avenues for future work:

- Here, we have only considered application of the incremental refinement algorithm in two- and three-dimensional settings. However, it should be possible to compute time-varying three-dimensional FTLE fields in a four-dimensional setting using this approach.
- Computation of the flow map is in essence a highly parallel task, since the pathline computations are mutually independent. It is therefore possible to exploit parallel machine architectures to further accelerate these computations. We have proposed a hardware-accelerated implementation on the graphical processing unit (GPU) in [GLT*07] that allows a significant performance increase
- While we have provided some effective visualization types, we would like to devise coherent structure visualization that effectively combine particle,

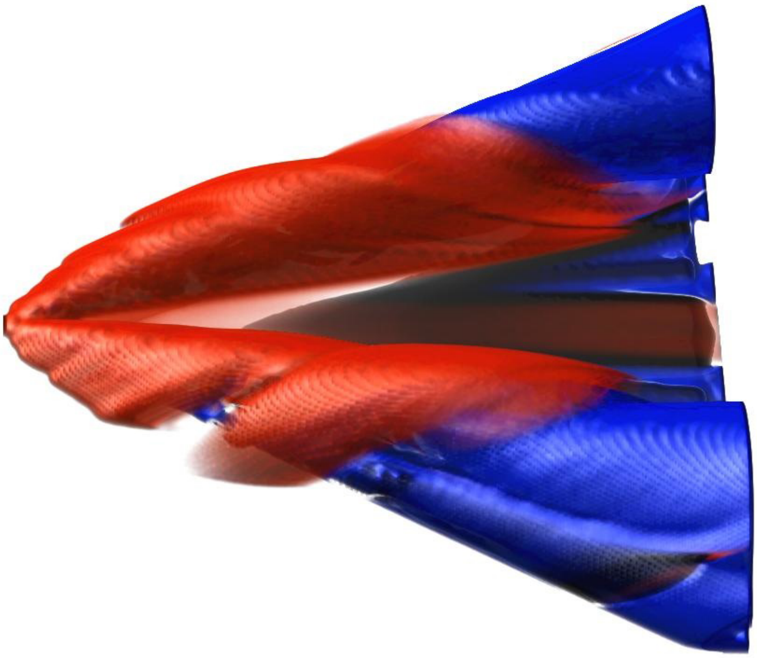


Figure 5.10: Volume Rendering of $\sigma_n^+ \times \sigma_n^-$ (TDELTA). Wing edge separation and the primary attachment layer. Occlusion is problematic and has to be resolved through the use of cropping or clipping (Figure 5.11).

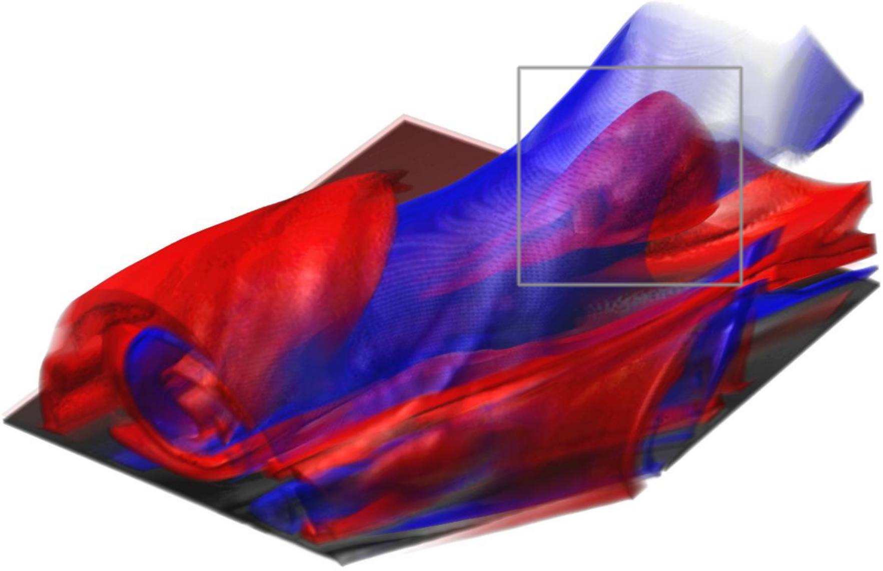


Figure 5.11: Crop along the middle third of the left wing edge (cf. Figure 5.10). The interplay of separation and attachment structures is visible on the front face. The grey box highlights the separation structure that characterizes a vortex breakdown bubble.

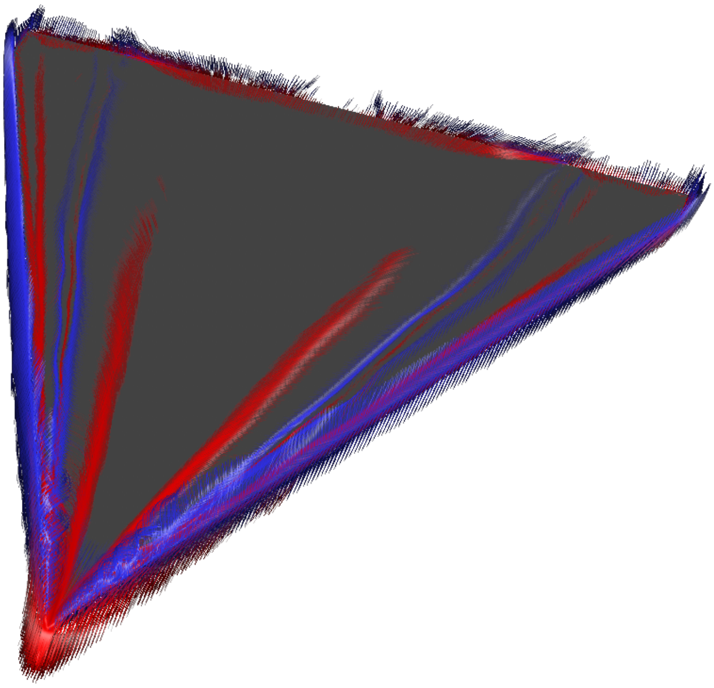


Figure 5.12: Separation and attachment structures on the wing surface. Pathlines were seeded according to PDFs of σ^+ (red) and σ^- (blue).

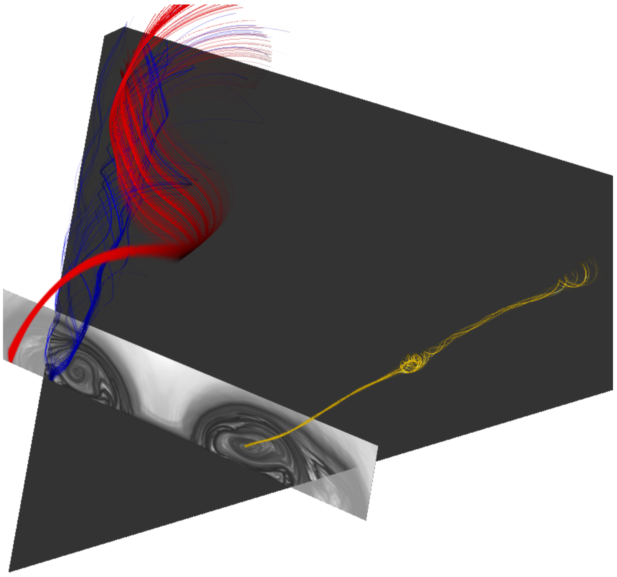


Figure 5.13: Visualization of primary (red) and secondary (blue) separation structures. Pathlines seeded according to PDF in Figure 5.14.

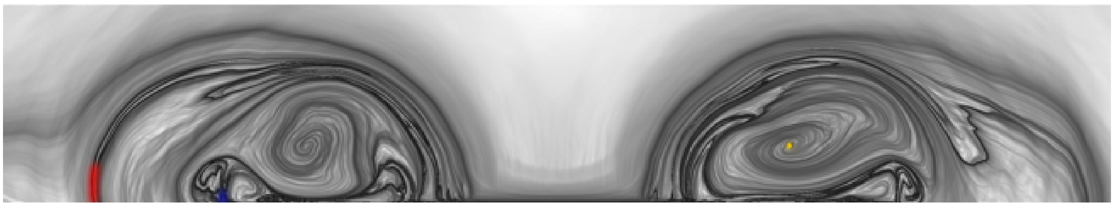


Figure 5.14: Planar FTLE visualization on a section plane perpendicular to the main flow direction. Darker regions correspond to regions of high σ^+. Colored regions indicate user-guided PDFs that were used to seed trajectories in Figure 5.13.

line, surface, and volume representations, possibly in combination with illustrative rendering methods, to convey the progressive transitions between neighboring or intertwined flow structures. On the same note, we would like to examine new user interfaces that facilitate working with FTLE-based and derived visualization types.

- Recently, a number of flow-derived quantities closely related to FTLE, such as e.g. the Finite Separation Lyapunov Exponent[KL02, JL02] have appeared in the literature. In similarity to FTLE, their computation relies on the flow map. Therefore, a straightforward application of our algorithm to the computation of these quantities seems possible and should be investigated.

Appendix A

Pseudocode

In this appendix, we will discuss implementation details for some of the algorithms we employ and present pseudocode fragments for specific parts of these algorithms. While we will stay close to C++ from both a syntactic and semantic point of view, we have opted to deviate from certain aspects of the language definition where necessary to keep the presentation simple. In this case, source comments indicate the intended semantics.

C++ Standard Template Library (STL) datatypes are used throughout the code. The well-defined `list<>` and `vector<>` templates and their corresponding `iterator` types free us from the burden to manually specify such data structures and operations thereon. For a general reference on both C++ in general and STL datatype operations, we recommend the excellent book by Stroustrup [Str00].

A.1 The Dopri5 Integration Scheme

In this section, we present pseudocode of our implementation of the Dopri5 intergration scheme that is described in Section 2.10.

The implementation focusses on the dense output capability of Dopri5 and the structure `dopri5_step` encapsulates one successful integration step and the capability to interpolate over it. The structure `dopri5` encapsulates integrator state and several control variables, such as the prescribed tolerances in terms of `reltol` and `abstol`. These values are set by the user before or during the integration. Additionally, it provides the method `step()` to advance the integration. The latter returns a structure `dopri5_step` that contains all information about the last successful step taken, or an error code in case of failure. In case the user does not specify an initial step size (`h 0̄.0`), the method `h_init()` performs an educated guess by evaluating the vector field in the vicinity of the starting point. To allow treatment of systems of ordinary differential equations of arbitrary dimension, we presume the existence of a type `value_type` that encapsulates vector operations.

```
// numerical precision
const double epsilon = std::numeric_limits<double>::epsilon();

// coefficients for the stages k1 ... k7
const double
  a21 = 0.2,
  a31 = 3.0/40.0,       a32 = 9.0/40.0,
  a41 = 44.0/45.0,      a42 = -56.0/15.0,      a43 = 32.0/9.0,
  a51 = 19372.0/6561.0, a52 = -25360.0/2187.0, a53 = 64448.0/6561.0,
  a54 = -212.0/729.0,
  a61 = 9017.0/3168.0,  a62 = -355.0/33.0,     a63 = 46732.0/5247.0,
  a64 = 49.0/176.0,     a65 = -5103.0/18656.0,
  a71 = 35.0/384.0,                            a73 = 500.0/1113.0,
  a74 = 125.0/192.0,    a75=-2187.0/6784.0,    a76 = 11.0/84.0;

// 4th order approximation coeff.
const double
  d1 = -12715105075.0/11282082432.0,
  d3 = 87487479700.0/32700410799.0,
  d4 = -10690763975.0/1880347072.0,
  d5 = 701980252875.0/199316789632.0,
  d6 = -1453857185.0/822651844.0,
  d7 = 69997945.0/29380423.0;

// error (5th order / 4th order ) approximation coeff.
const double
  e1 = 71.0/57600.0,      e3 = -71.0/16695.0, e4 = 71.0/1920.0,
  e5 = -17253.0/339200.0, e6 = 22.0/525.0,    e7 = -1.0/40.0;


// ----------------------------------------

struct dopri5_step
{
  value_type p[5];     // coefficients of interpolation polynomial
  double     t0, t1;   // step begin and end time

  // interpolate solution over step in [t0, t1]
  value_type y( double t )
  {
    double a = (t - t0) / (t1 - t0);
    double b = 1.0 - a;

    // Horner-type evaluation of the polynomial
    return p[0]+a*(p[1]+b*(p[2]+a*(p[3]+b*p[4])));
  }
  // interpolate derivative over step in [t0, t1]
  value_type dy( double t )
  {
    double a = (t - t0) / (t1 - t0);
    double b = 1.0 - a;

    value_type pc = p[3]+b*p[4];

    // Horner−type evaluation of the polynomial
    return (p[1]+b*(p[2]+a*pc)-
            a*(p[2]+a*pc-b*(p[3]+(b-a)*p[4])))/(t1-t0);
    }
};

// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

struct dopri
{
  enum result
  {
      OK,
      T_MAX_REACHED,
      STEPSIZE_UNDERFLOW,
  };

  double reltol;     // relative tolerance to obey when adapting h
  double abstol;     // absolute tolerance to obey when adapting h

  double h;          // last step size
  double h_max;      // maximal step size (user)
  int    neval;      // number of function evaluations so far


  double     t;      // current integration time
  double     t_max;  // maximum integration time
  value_type y;      // current integration position
  value_type k1;     // k1 is stored between steps to ensure
                     // first−same−as−last (FSAL) principle

  double facold;     // stepsize control stabilization


public:

  // constructor with default values
```

```
dopri5() : reltol( 1e-7 ), abstol( 1e-7 ),
           h( 0.0 ), h_max( 0.0 ),
           t( 0.0 ), t_max( 0.0 ),
           facold( 1e-4 ), neval( 0 )
{
}

// heuristic to determine initial step size
double h_init( function rhs, double h_max )
{
  double direction = (tmax-t >= 0) ? 1 : -1;
  double h;

  h_max = fabs(h_max);

  value_type sk = abstol + reltol * abs( y );

  double dnf = norm_square( k1 / sk );
  double dny = norm_square( y / sk );

  if( (dnf <= 1e-10) || (dny <= 1e-10) )
    h = 1.0e-6;
  else
    h = sqrt( dny/dnf ) * 0.01;

  h = min( h, h_max );

  // perform an explicit Euler step (predictor)
  value_type k3 = y + h * direction * k1;
  value_type k2 = rhs( t + direction * h, k3 );
  n_eval++;

  // estimate the second derivative of the solution
  sk  = abstol + reltol * abs( y );
  double der2 = norm( (k2-k1)/sk ) / h;

  // step size is computed such that
  // h**(1/5) * max( norm(k1), norm(der2) ) = 0.01
  double der12 = max( fabs(der2), sqrt(dnf) );

  double h1;

  if( der12 <= 1.0e-15 )
    h1 = std::max( 1.0e-6, h*1.0e-3 );
  else
    h1 = pow( 0.01/der12, 0.2 );
  h = std::min( 100.0*h, h1 );
  h = std::min( h, h_max );

  return direction * h;
}
```

The function `step()` takes a single integration step of size `minh`. h contains the next selected step size. The right hand side of the ordinary differential equation is evaluated through the function object `rhs` that can be called in the form `rhs( double t, value_type x )` and returns the corresponding vector. Note that the integrator can be used for both forward and backward integration by specifying a t_max that is less than t. `step()` returns `OK` if a successful step was possible. The return values `TMAX_REACHED` and `STEPSIZE_UNDERFLOW` indicate the obvious conditions.

```
result step( function rhs, dopri5_step& step )
{
  return (b >= 0.0) ? fabs(a) : -fabs(a);

  const double direction = (t_max - t >= 0) ? 1 : -1;

  value_type k2, k3, k4, k5, k6, k7, y_new;
  bool reject = false;

  // compute maximum stepsize
  double local_h_max = h_max;

  if( local_h_max == 0.0 )
    local_h_max = t_max - t;

  // if this is the first step, compute k1 (FSAL)
  // also needed for h_init()
  if( n_steps == 0 )
  {
    k1 = rhs( t, y );
    n_eval++;
  }

  // determine stepsize if none was given (h == 0.0)
  if( h == 0.0 )
    h = h_init( rhs, local_h_max );
  else
    h = direction * h;

  // integration step loop
// this loop is left if a step is accepted
while( true )
{
  bool last = false;

  // check for stepsize underflow
  if( 0.1*abs(h) <= abs(t)*epsilon )
    return STEPSIZE_UNDERFLOW;

  // ensure that integration does not proceed beyond t_max
  if( (t + 1.01*h - t_max) * direction > 0.0 )
  {
    last = true;
    h = t_max - t;
  }

  // perform Runge–Kutta stages
  y_new = y + h*a21*k1;
  k2 = rhs( t+c2*h, y_new );

  y_new = y + h * ( a31*k1 + a32*k2 );
  k3 = rhs( t+c3*h, y_new );

  y_new = y + h * ( a41*k1 + a42*k2 + a43*k3 );
  k4 = rhs( t+c4*h, y_new );

  y_new = y + h * ( a51*k1 + a52*k2 + a53*k3 + a54*k4 );
  k5 = rhs( t+c5*h, y_new );

  y_new = y + h * ( a61*k1 + a62*k2 + a63*k3 + a64*k4 + a65*k5 );
  k6 = rhs( t+h, y_new );

  y_new = y + h * (a71*k1 + a73*k3 + a74*k4 + a75*k5 + a76*k6 );
  k7 = rhs( t+h, y_new );

  double err = 0.0, h_new, fac11;

  // estimate error
  value_type ee = h * ( e1*k1 + e3*k3 + e4*k4 +
                        e5*k5 + e6*k6 + e7*k7 );

  value_type sk = abstol + reltol * max( abs(y), abs(y_new) );
  double err = norm_square( ee / sk ) / value_type::size();
```

```
      if( err <= 1.0 )
      {
        // accept step and return

        // set values dopri_step structure
        // interpolation polynomial, time interval etc.
        step.p[0] = y;
        step.p[1] = y_new - y;
        step.p[2] = h*k1 - step_.p[1];
        step.p[3] = -h * k7 + step_.p[1] - step_.p[2];
        step.p[4] = h * ( d1*k1 + d3*k3 + d4*k4 + d5*k5 + d6*k6 + d7*k7 );
        step.t0 = t;
        step.t1 = t+h;

        // compute next (potential) stepsize
        double emax = std::max( err, 1.0e-4 );
        double fac = pow( err, 0.17 ) / pow( emax, 0.04 );

        if( fac > 5 )
          fac = 5;
        else if( fac < 0.1 )
          fac = 0.1;

        h /= fac;

        // update integrator state
        k1 = k7;
        t  = t+h;

        return last ? T_MAX_REACHED : OK;
      }
      else
      {
        // step rejected, must retry with smaller h
        double mult = pow( err, 0.17 ) / 0.9;

        if( mult > 5 )
          mult = 5;

        h /= mult;
      }
    }
  }
};
```

A.2 Improved Stream Surface Algorithm

This section presents pseudo-code for the improved stream surface algorithm from Section 3.4. It incorporates basic data structures and an algorithm skeleton. In the given form, the code has been structured for clarity of presentation.

A typical implementation would incorporate a mesh data structure for stream surface representation and manipulate it during the integration of the surface. To keep the code simple, we have omitted this part of the implementation. Furthermore, we have skipped on several heuristics that improve performance but make the code much harder to understand.

The primary data structure is based on the types `node` and `front`. Following the terminology introduced in Section 3.1, a `front` is a list of streamlines that traverse the stream surface as it is integrated. Each streamline is encapsulated in a `node` structure to simplify streamline handling and to manage information about front triangulation. We will not discuss details on streamline integration here, but refer the reader to Section A.1.

```
struct node
{
  bool open_left;    // true if node is not triangulated to the left
  bool open_right;   // true if node is not triangulated to the right
  bool finished;     // true if streamline could not be continued
            // (set by step())
  bool remove;     // true if node should be tried to remove
            // the next time it is touched

  point current; // current sample point on streamline
  point last;    // previous sample point on streamline

  // construct a new node
  // – initializes a streamline at p
  // – finished, open_left, open_right, remove are set to false
  // – current and last are set to p
  node( point p );

  // determine the next sample point on the streamline
  // – set finished=true if the streamline cannot be continued
  // – set last=current, and current is assigned the new sample
  step();
};
```

The flags `open_left` and `open_right` keep track of whether the current sample point has been used in the front triangulation. By definition, boundary nodes are assumed to be triangulated to the corresponding side. For example, `open_left` is always false for the leftmost node.

The method `step()` is used to advance a streamline and determine the next sample point, which is then assigned to `current`. As the algorithm frequently need the previous sample point, it is kept in `last`. A failure of streamline integration is indicated by `finished`. The extra flag `remove` is needed to indicate such `nodes` that are eliminated by the merging. The `node` cannot be deleted right away since they are needed to finish the merging triangulation as soon as their neighbor nodes are in the correct state.

```
typedef list<node> front;
typedef list<front> frontlist;

frontlist fronts;

const double angle_max, angle_min, dist_max, dist_min;
```

Multiple separate fronts may exist as a result of splitting, and they are stored in `fronts`. The variables `angle_`{`min|max`} and `d_`{`min|max`} are assumed to be set according to the thresholds $\alpha_{\min}$, $\alpha_{\max}$ and $d_{\min}$, $d_{\max}$ as defined in Section 3.4.3, and that `fronts` is populated with the initial front as a discretization of the starting curve. Note that it is usually best if this discretization does not violate the front resolution criteria (cf. Sec. 3.4.3).

Nodes and fronts are pointed to by `front::iterator` and `frontlist::iterator`, respectively. We next introduce a short procedure that splits a ribbon (indicated by two adjacent node iterators) by creating a new node in the middle of the joining segment. This node is then inserted into the front.

```
void split_ribbon( frontlist::iterator fi,
                   front::iterator ni0,
                   front::iterator ni1 )
{
  point p_mid = (ni0->last + ni1->last)/2;

  node n( p_mid );

  // insert new node before ni1
  fi->insert( n, ni1 );
}
```

The overall algorithm is iterative in nature: in a loop, the current node in the current front is examined, and based on its state, several possible actions are undertaken to advance the front. This main loop is encapsulated in the following procedure:

```
void advance_surface()
{
  frontlist::iterator fi = fronts.begin();

  front::iterator ni_left, ni, ni_right;

  ni = fi->begin();

  // main loop
  while( true )
  {
    // going beyond the end of the list of nodes
    // restart with first node
    if( ni == fi->end() )
      ni = fi->begin();

    // set left and right node iterators
    ni_left = ni_right = ni;

    if( ni_left != fi->begin() )
      --ni_left;

    if( ++ni_right = fi->end() )
      ni_right = ni;
```

The iterator `fi` points to the current front (starting with the first front), while `ni` points to the current node of `fi` (starting with the leftmost node). To simplify repeated access to neighbor nodes, two iterators `ni_left` and `ni_right` are maintained in each loop iteration. To simplify the handling of boundary nodes, we set the corresponding neighbor iterator to `ni` if no neighbor node exists. At the start of each iteration, node and front mangement is performed:

```
    // if there is only one node left in the current front,
    // both node and front must be erased
    if( ni_left == ni_right )
    {
      // delete the last node
      fi->erase( ni );

      // delete the front
      fronts.erase( fi );

      // start with the next front in the next iteration
      if( fronts.empty() )
        break;
      else
    {
      fi = fronts.begin();
      ni = fi->begin();
    }

    // restart loop
    continue;
  }
```

If only a single node remains, the current front is finished and is deleted from `fronts`.

At a finished node, the front must be split. This is simply achieved by appending a new front to `fronts` and splicing all right-hand neighbors of the current front to the new front. Since there are no triangulation requirements for this case, the node can be erased from the front right away.

```
  // if *ni is finished, the front must be split
  if( ni->finished )
  {
    // is ni at the beginning or end of the front?
    // if yes, do not split, just erase
    if( ni != ni_right && ni != ni_left )
    {
      ni_right->open_left = false;

      frontlist::iterator fi_new =
        fronts.insert( front(), fronts.end() );

      fi_new->splice( fi_new->end(), *fi, ni_right, fi->end() );
    }

    fi->erase( ni );
    ni_left->open_right = false;

    // restart loop at the beginning of the current front
    ni = fi->begin();
    continue;
  }
```

If a node marked with `removable` is encountered, a merging triangulation must be generated. This is only possible if both left and right neighbors can be triangulated to the current node. If this condition is not met, we restart the main iteration at the neighbor that cannot be triangulated.

```
// if *ni is marked as removable, the triangulation to left
// and right nodes must have to be finished before removal
if ( ni->remove )
{
  // can both sides be triangulated at once?
  if( ni_left->open_right && ni_right->open_left )
  {
    // can triangulate to both sides at once
    emit_triangle( ni_left->last,
             ni_left->current, ni->current );
    emit_triangle( ni_left->last,
             ni->current, ni_right->last );
    emit_triangle( ni_right->last,
             ni->current, ni_right->current );

    ni_left->open_right = false;
    ni_right->open_left = false;

    // erase *ni and restart loop at next node
    ni = fi->erase( ni );
    continue;
  }
  else if( !ni_left->open_right )
  {
    // cannot triangulate the left node,
    ni = ni_left;
    continue;
  }
  else // ni_right->open_left == false
  {
    // cannot triangulate the right node
    ni = ni_right;
    continue;
  }
}
```

To keep the algorithm simple and avoid unnecessary duplication of symmetric cases, triangulation is only constructed between the current node and its left neighbor. If this condition is not met, iteration is restarted at the left or right neighbors based on their open flags.

```
// is triangulation to the left possible?
if( ni->open_left && !ni_left->open_right && ni_left!=ni) )
{
  // no, continue left until a node can be advanced further
  ni = ni_left;
```

```
    continue;
  }

  // is triangulation to the right possible?
  if( !ni->open_left && ni->open_right )
  {
    // no, continue right until a node can be advanced further
    ni = ni_right;
    continue;
  }
```

A current node that is triangulated to both sides implies that two neighboring ribbons are abreast, and their leading edges form a front segment. In this situation, refinement criteria that depend on front segment angle must be checked. If neither refinement or coarsening are indicated, a step is taken for the current node, it is marked open to both sides, and the outer iteration is restarted. Otherwise, if the curvature criterion holds, both ribbons are split. In the third case, the combined front length falls below $d_{\min}$, indicating that the ribbons must be merged.

```
  // is the current node completely triangulated?
  if( !ni->open_left && !ni->open_right )
  {
    // for non−boundary nodes that are not to be removed,
    // we check refinement criteria
    if( ni != ni_left && ni != ni_right &&
      !ni_left->remove && !ni_right->remove )
    {
      point L = ni_left->open_right ? ni_left->last :
                        ni_left->current;
      point R = ni_right->open_left ? ni_right->last :
                        ni_right->current;
      point M = ni->current;

      double a = angle( L, M, R );
      double d = distance( L, M ) + distance( M, R );

      // curvature angle exceeded?
      if( a < angle_max )
      {
        // yes, must split both front segments
        split_front( fi, ni_left, ni );
        split_front( fi, ni, ni_right );

        continue;
      }
      // convergence detected?
      if( d < dist_min && a > angle_min )
      {
        // yes, mark node for removal in the next pass
        ni->remove = true;
        continue;
      }
    }

    // no refining/coarsening triggered
    // take an integration step
    ni->step();

    // mark node open on both sides (after step), but boundary
    // nodes are always closed on the corresponding sides
    ni->open_left = (ni != ni_left);
    ni->open_right = (ni != ni_right);

    // restart loop
    continue;
  }

  // now, can triangulate if ni is not on the left boundary
  // continue with ni_right in that case
  if( ni_left == ni )
  {
    ni = ni_right;
    continue;
  }
```

At this point in the iteration the current node is found ready for triangulation since all other cases have been previously excluded. The actual triangulation is performed on the ribbon spanned by `ni_left` and `ni`. According to Hultquist's greedy minimal tiling (cf. Section 3.3.1) the shorter of the two diagonals is chosen as the new leading edge, a triangle is generated, and the open flags are marked accordingly. Before the ribbon is triangulated, the width of the ribbon is checked to trigger a ribbon split in the case of flow divergence.

```
  // all quadrilateral points of the ribbon
  // ni_left -> ni are available
  point L0 = ni_left->last;
  point L1 = ni_left->current;

  point R0 = ni_right->last;
  point R1 = ni_right->current;
```

```
      // is front distance split criterion fulfilled?
      if( distance( L1, R1 ) > dist_max )
      {
        // yes, split front between L0 and L1
        split_front( fi, ni );
        continue;
      }

      // triangulate between ni and ni_left
      double len0 = distance( L0, R1 );
      double len1 = distance( L1, R0 );

      if( len0 < len1 )
      {
        // left diagonal
        emit_triangle( L0, R1, R0 );
        ni->open_left = false;
      }
      else
      {
        // right diagonal
        emit_triangle( L1, R0, L0 );
        ni_left->open_right = false;
      }

      // iteration restarts
    }
    // end of main loop
  }
```

The main loop is left and the function terminated after the last front has been deleted.

Bibliography

[And73] Andronov A. A.: *Qualitative Theory of Second-Order Dynamic Systems*. John Wiley & Sons, 1973.

[CH97] Cai W., Heng P.-A.: Principal stream surfaces. In *VIS '97: Proceedings of the 8th conference on Visualization '97* (Los Alamitos, CA, USA, 1997), IEEE Computer Society Press, pp. 75–ff.

[CK90] Cash J. R., Karp A. H.: A variable order runge-kutta method for initial value problems with rapidly varying right-hand sides. *ACM Trans. Math. Softw. 16* (1990), 201–222.

[CL93] Cabral B., Leedom L.: Imaging vector fields using line integral convolution. *Computer Graphics (SIGGRAPH '93 Proceedings) 27*, 4 (1993), 263–272.

[CM04] Chorin A. J., Marsden J. E.: *A Mathematical Introduction to Fluid Mechanics*. Texts in Applied Mathematics. Springer, Berlin, 2004.

[Dal83a] Dallmann U.: Topological structures of three-dimensional flow separation. *AIAA Paper 83-1735* (1983).

[Dal83b] Dallmann U.: *Topological Structures of Three-Dimensional Flow Separations*. Tech. Rep. 221-82 A 07, Deutsche Forschungs- und Versuchsanstalt fuer Luft- und Raumfahrt, 1983.

[EGMP94] Eberly D., Gardner R., Morse B., Pizer S.: Ridges for image analysis. *Journal of Mathematical Imaging and Vision 4* (1994), 355–371.

[Esc84] Escudier M. P.: Observations of the flow produced in a cylindrical container by a rotating endwall. *Experiments in Fluids 2* (1984), 189 – 196.

[FP98] Furst J., Pizer S. M.: Marching optimal-parameter ridges: An algorithm to extract shape loci in 3d images. In *MICCAI '98: Proceedings of the First International Conference on Medical Image*

Computing and Computer-Assisted Intervention (London, UK, 1998), Springer-Verlag, pp. 780–787.

[GGTH07] GARTH C., GERHARDT F., TRICOCHE X., HAGEN H.: Efficient computation and visualization of coherent structures in fluid flow applications. *Proceedings of IEEE Visualization 2007 (to appear)* (2007).

[GH83] GUCKENHEIMER J., HOLMES P.: *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields.* Springer-Verlag, 1983.

[GLL91] GLOBUS A., LEVIT C., LASINSKI T.: A tool for visualizing the topology if three-dimensional vector fields. In *IEEE Visualization Proceedings* (October 1991), pp. 33 – 40.

[GLT*06] GARTH C., LARAMEE R., TRICOCHE X., SCHNEIDER J., HAGEN H.: Extraction and visualization of swirl and tumble motion from engine simulation data. In *Proceedings of The Topology-Based Methods in Visualization Workshop* (2006).

[GLT*07] GARTH C., LI G. S., TRICOCHE X., HANSEN C. D., HAGEN H.: Visualization of coherent structures in transient 2d flows. In *Topology-Based Methods in Visualization 2007, to appear* (2007), Mathematics + Visualization, Springer.

[GRH06] GREEN M., ROWLEY C., HALLER G.: Detection of lagrangian coherent structures in 3d turbulence. *J. Fluid Mech. to appear* (2006).

[GSBB87] GLADWELL I., SHAMPINE L. F., BACA L. S., BRANKIN R. W.: Practical aspects of interpolation in runge-kutta codes. *SIAM J. Sci. Stat. Comput. 8*, 3 (1987), 322–341.

[GTS04] GARTH C., TRICOCHE X., SCHEUERMANN G.: Tracking of vector field singularities in unstructured 3D time-dependent data sets. In *Proceedings of IEEE Visualization 2004* (2004), pp. 329–226.

[GTSS04] GARTH C., TRICOCHE X., SALZBRUNN T., SCHEUERMANN G.: Surface techniques for vortex visualization. In *Proceedings Eurographics - IEEE TCVG Symposium on Visualization* (May 2004).

[Hal00] HALLER G.: Finding finite-time invariant manifolds in two-dimensional velocity fields. *Chaos 10*, 1 (2000), 99–108.

[Hal01a] HALLER G.: Distinguished material surfaces and coherent structures in three-dimensional flows. *Physica D 149* (2001), 248–277.

[Hal01b] HALLER G.: Lagrangian structures and the rate of strain in a partition of two-dimensional turbulence. *Physics of Fluids 13*, 11 (2001).

[Hal02] Haller G.: Lagrangian coherent structures from approximate velocity data. *Physics of Fluids 14*, 6 (june 2002), 1851–1861.

[Hal05] Haller G.: An objective definition of a vortex. *Journal of Fluid Mechanics 525* (2005), 1–26.

[HH91] Helman J. L., Hesselink L.: Visualizing vector field topology in fluid flows. *IEEE Computer Graphics and Applications 11*, 3 (May 1991), 36–46.

[Hig91] Higham D. J.: Highly continuous runge-kutta interpolants. *ACM Trans. Math. Softw. 17*, 3 (1991), 368–386.

[HNW93] Hairer E., Nørsett S. P., Wanner G.: *Solving Ordinary Differential Equations I, second edition*, vol. 8 of *Springer Series in Comput. Mathematics*. Springer-Verlag, 1993.

[HS74] Hirsch M. W., Smale S.: *Differential Equations, Dynamical Systems and Linear Algebra*. Academic Press, New York, 1974.

[HS84] Hestenes D., Sobzyk G.: *Clifford Algebra to Geometric Calculus*. D. Reidel Publishing Company, 1984.

[Hul92] Hultquist J. P. M.: Constructing stream surfaces in steady 3d vector fields. In *Proceedings of IEEE Visualization 1992* (Boston, MA, 1992), Kaufman A. E., Nielson G. M., (Eds.), pp. 171 – 178.

[HY00] Haller G., Yuan G.: Lagrangian coherent structures and mixing in two-dimensional turbulence. *Physica D 147* (2000), 352–370.

[JH95] Jeong J., Hussain F.: On the Identification of a Vortex. *Journal of Fluid Mechanics 285* (1995), 69 – 94.

[JL02] Joseph B., Legras B.: Relation between kinematic boundaries, stirring, and barriers for the antarctic polar vortex. *J. Atmos. Sci. 59* (2002), 1198 – 1212.

[JMT02] Jiang M., Machiraju R., Thompson D.: Geometric verification of swirling features in flow fields. In *IEEE Visualization Proceedings* (2002), pp. 307 – 314.

[Ken98] Kenwright D. N.: Automatic detection of open and closed separation and attachment lines. In *IEEE Visualization Proceedings* (Los Alamitos, CA, 1998), Press I. C. S., (Ed.), pp. 151–158.

[KHL99] Kenwright D., Henze C., Levit C.: Feature extraction of separation and attachment lines. *IEEE Transactions on Visualization and Computer Graphics 5*, 2 (1999), 135–144.

[KKH01] Kniss J., Kindlmann G., Hansen C.: Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings IEEE Visualization 2001* (October 2001), pp. 255–262.

[KKH02] Kniss J., Kindlmann G., Hansen C.: Multi-dimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics 8*, 3 (July-September 2002), 270–285.

[KL02] Koh T.-Y., Legras B.: Hyperbolic lines and the stratospheric polar vortex. *Chaos 12* (2002), 382 – 394.

[Kni] Kniss J. M.: Simian – volume rendering.

[Kob94] Kobbelt L.: *Iterative Erzeugung glatter Interpolanten.* PhD thesis, Universität Karlsruhe, 1994.

[KTW06] Kindlmann G., Tricoche X., Westin C.-F.: Anisotropy creases delineate white matter structure in diffusion tensor mri. In *Proceedings of Medical Imaging Computing and Computer-Assisted Intervention, MICCAI '06* (2006).

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

[LGD*05] Laramee R., Garth C., Doleisch H., Schneider J., Hauser H., Hagen H.: Visual analysis and exploration of fluid flow in a cooling jacket. In *Proc. IEEE Visualization '05 Conference* (2005), pp. 623–630.

[LHZP05] Laramee R. S., Hauser H., Zhao L., Post F. H.: Topology-based flow visualization, the state of the art. In *Topology-Based Methods in Visualization 2005* (2005), Hauser H., Hagen H., Theisel H., (Eds.), Springer.

[LMGP97] Löffelmann H., Mroz L., Gröller E., Purgathofer W.: Stream arrows: enhancing the use of stream surfaces for the visualization of dynamical systems. *The Visual Computer 13*, 8 (1997), 359 – 369.

[LSM06] Lekien F., Shadden S., Marsden J.: Lagrangian coherent structures in n-dimensional systems. *Physica D submitted* (2006).

[LST03] Langbein M., Scheuermann G., Tricoche X.: An efficient point location method for visualization in large unstructured grids. In *Proceedings of Vision, Modeling, Visualization* (2003).

[Lug79] Lugt H. J.: *The Dilemma of Defining a Vortex.* Springer, 1979, pp. 309–321.

[Lug96] Lugt H. J.: *Introduction to Vortex Theory.* Vortex Flow Press, Inc., 1996.

[MDSB02] Meyer M., Desbrun M., Schröder P., Barr A.: Discrete differentialgeometry operators for triangulated 2-manifolds. In *VisMath* (2002).

[MHP*06] Mathur M., Haller G., Peacock T., Ruppert-Felsot J., Swinney H.: Uncovering the lagrangian skeleton of turbulence. *Phys. Rev. Lett. submitted* (2006).

[Nie04] Nielson G. M.: Dual marching cubes. In *VIS '04: Proceedings of the conference on Visualization '04* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 489–496.

[NJ99] Nielson G. M., Jung I.-H.: Tools for computing tangent curves for linearly varying vector fields over tetrahedral domains. *IEEE Transactions on Visualization and Computer Graphics 5*, 4 (1999), 360–372.

[OK97] Okada A., Kao D. L.: Enhanced line integral convolution with flow feature detection. In *Proceedings of IS&T/SPIE Electronic Imaging* (1997).

[Oku70] Okubo A.: Horizontal dispersion of floatable particles in the vicinity of velocity singularities such as convergences. *Deep-Sea Res. 17* (1970), 445 – 454.

[PBP02] Prautzsch H., Boehm W., Paluszny M.: *Bézier and B-Spline Techniques.* Springer Berlin, 2002.

[PD81] Prince P. J., Dormand J. R.: High order embedded runge-kutta formulae. *Journal of Computational and Applied Mathematics 7*, 1 (1981).

[Pei62] Peixoto M. M.: Structural stability on two-dimensional manifolds. *Topology I* (1962).

[PFTV92] Press W. H., Flannery B. P., Teukolsky S. A., Vetterling W. T.: *Numerical Recipes: The Art of Scientific Computing*, 2nd ed. Cambridge University Press, Cambridge (UK) and New York, 1992.

[Poi] POINCARÉ H.: Sur les courbes définies par une équation différentielle. *J. Math. 1:167–244, 1875. J. Math. 2:151–217, 1876. J. Math. 7:375–422, 1881. J. Math. 8:251–296, 1882.*

[Pop03] POPINET S.: Gerris: a tree-based adaptive solver for the incompressible euler equations in complex geometries. *J. Comput. Phys. 190*, 2 (2003), 572–600.

[PR00] PEIKERT R., ROTH M.: The Parallel Vectors operator - a vector field visualization primitive. In *IEEE Visualization Proceedings '00* (2000), pp. 263 – 270.

[PVH*03] POST F., VROLIJK B., HAUSER H., LARAMEE R., DOLEISCH H.: The state of the art in flow visualization: Feature extraction and tracking. *Computer Graphics Forum 22*, 4 (2003), 775–792.

[QV97] QUARTERONI A., VALLI A.: *Numerical Approximation of Partial Differential Equations*. Springer Series in Computational Mathematics. Springer, 1997.

[RP98] ROTH M., PEIKERT R.: A higher-order method for finding vortex core lines. In *IEEE Visualization Proceedings '98* (1998), pp. 143 – 150.

[Rüt05] RÜTTEN M.: *Topologische Untersuchung des Wirbelplatzens zur Identifikation von Wirbelplatzparametern*. PhD thesis, TU Clausthal-Zellerfeld, 2005.

[SBH*01] SCHEUERMANN G., BOBACH T., HAGEN H., MAHROUS K., HAHMAN N., JOY K.: A tetrahedra-based stream surface algorithm. In *IEEE Visualization Proceedings* (2001).

[SBM*05] SCHROEDER W. J., BERTEL F., MALATERRE M., THOMPSON D., PEBAY P. P., O'BARA R., TENDULKAR S.: Framework for visualizing higher-order basis functions. In *Proceedings of IEEE Visualization 2005* (Los Alamitos, CA, USA, 2005), IEEE Computer Society, pp. 43–50.

[Sch99] SCHEUERMANN G.: *Topological Vector Field Visualization with Clifford Algebra*. PhD thesis, Universität Kaiserslautern, 1999.

[SDM06] SHADDEN S., DABIRI J., MARSDEN J.: Lagrangian analysis of fluid transport in empirical vortex ring flows. *Physics of Fluids 18* (2006), 047105.

[SH95] SUJUDI D., HAIMES R.: *Identification of Swirling Flow in 3D Vector Fields*. Tech. Rep. AIAA Paper 95–1715, American Institute of Aeronautics and Astronautics, 1995.

[Sha85] SHAMPINE L. F.: Interpolation for runge-kutta methods. *SIAM J. Numer. Anal. 5* (1985).

[SHJK98] SCHEUERMANN G., HAMANN B., JOY K. I., KOLLMANN W.: Visualizing local topology. *Journal of Electronic Imaging 9*, 4 (1998).

[SJH05] SURANA A., JACOBS G., HALLER G.: Extraction of separation and reattachment surfaces from 3d steady shear flows. *submitted to AIAA Journal, available at http://web.mit.edu/ghaller* (2005).

[SLM05] SHADDEN S., LEKIEN F., MARSDEN J.: Definition and properties of lagrangian coherent structures from finit-time lyapunov exponents in two-dimensional aperiodic flows. *Physica D 212* (2005), 271–304.

[Sma67] SMALE S.: Differentiable dynamical systems. *Bull. Amer. Math. Soc. 73* (1967), 747–817.

[SP07] SADLO F., PEIKERT R.: Visualizing lagrangian coherent structures: A comparison to vector field topology. In *Topology-Based Methods in Visualization 2007, to appear* (2007).

[Sta98] STALLING D.: *Fast Texture-Based Algorithms for Vector Field Visualization.* PhD thesis, Freue Universität Berlin, 1998.

[Str00] STROUSTRUP B.: *The C++ Programming Language, Special Edition.* Addison-Wesley, Boston, 2000.

[SVL01] SOTIROPOULOS F., VENTIKOS Y., LACKEY T. C.: Chaotic advection in three-dimensional stationary vortex-breakdown bubbles: Šil'nikov's chaos and the devil's staircase. *Journal of Fluid Mechanics 444* (Oct. 2001), 257–297.

[SWH05] SAHNER J., WEINKAUF T., HEGE H.-C.: Galilean invariant extraction and iconic representation of vortex core lines. In *Proc. Eurographics / IEEE VGTC Symposium on Visualization (EuroVis '05)* (Leeds, UK, June 2005), K. Brodlie D. Duke K. J., (Ed.), pp. 151–160.

[TGB*04] TRICOCHE X., GARTH C., BOBACH T., SCHEUERMANN G., RUETTEN M.: Accurate and efficient visualization of flow structures in a delta wing simulation. In *Proceedings of 34th AIAA Fluid Dynamics Conference and Exhibit, AIAA Paper 2004-2153* (2004).

[TGK*04] TRICOCHE X., GARTH C., KINDLMANN G., DEINES E., SCHEUERMANN G., RÜTTEN M., HANSEN C.: Visualization of intricate flow structures for vortex breakdown analysis. In *Proceeding of IEEE Visualization '04 Conference* (2004), pp. 187–194.

[TGar] TRICOCHE X., GARTH C.: Topological methods for visualizing vortical flows. In *Mathematical Foundations of Visualization, Computer Graphics, and Massive Data Exploration*. 2006 (to appear).

[TGS05] TRICOCHE X., GARTH C., SCHEUERMANN G.: Fast and robust extraction of separation line features. In *Scientific Visualization: The Visual Extraction of Knowledge from Data* (2005), Mathematics + Visualization, Springer, pp. 245–263.

[TMJ06] TRICOCHE X., MACLEOD R. S., JOHNSON C. R.: Visual analysis of bioelectric fields. In *Proceedings of the 2006 International Workshop on Visualization in Medicine and Life Sciences* (2006).

[Tri02] TRICOCHE X.: *Vector And Tensor Field Topology Simplification, Tracking, and Visualization.* PhD thesis, Universität Kaiserslautern, 2002.

[TWHS03] THEISEL H., WEINKAUF T., HEGE H.-C., SEIDEL H.-P.: Saddle connectors - an approach to visualizing the topological skeleton of complex 3d vector fields. In *IEEE Visualization '03* (2003).

[TWSH02] TRICOCHE X., WISCHGOLL T., SCHEUERMANN G., HAGEN H.: Topology tracking for the visualization of time-dependent two-dimensional flows. *Computers & Graphics 26*, 2 (2002), 249 – 257.

[vW92] VAN WIJK J. J.: Rendering surface particles. In *IEEE Visualization Proceedings* (1992), pp. 54 – 61.

[vW93] VAN WIJK J.: Implicit stream surfaces. In *Proceedings of IEEE Visualization '93 Conference* (1993), pp. 245–252.

[Wei91] WEISS J.: The dynamics of enstrophy transfer in two-dimensional hydrodynamics. *Physica D 48* (1991), 273 – 294.

[Wie03] WIEBEL A.: Tetrahedrization of irregular grids and 3d helmholtz-hodge decomposition of vector fields. Project Thesis, Dept. of Computer Science, University of Kaiserslautern, 2003.

[WS02] WISCHGOLL T., SCHEUERMANN G.: Locating closed streamlines in 3d vector fields. In *Joint Eurographics and IEEE TCVG Symposium on Data Visualization 2002* (2002), pp. 227 – 232.

[WTHS04] WEINKAUF T., THEISEL H., HEGE H.-C., SEIDEL H.-P.: Boundary switch connectors for topological visualization of complex 3d vector fields. In *VisSym 2004 : Joint Eurographics/IEEE Symposium on Visualization* (Konstanz, Germany, 2004), Deussen O., Hansen C., Keim D. A., Saupe D., (Eds.), Eurographics, pp. 183–192.

[ZDHD92] Zhang Z.-F., Ding T.-R., Huang W.-Z., Dong Z.-X.: *Qualitative Theory of Differential Equations*. Translation of Mathematical Monographs. American Mathematical Society, 1992.

Curriculum Vitae

Persönliche Daten

Name:	Garth
Vorname:	Christoph
Geburtsdatum:	9. Dezember 1977
Geburtsort:	Blieskastel
Nationalität:	Deutsch
Familienstand:	verheiratet, ein Kind

Bildungsweg

1984 - 1988	Grundschule Bruchmühlbach
1988 - 1997	Staatliches Gymnasium Landstuhl
1997 - 2002	Studium Diplom Mathematik mit Nebenfach Informatik an der Technischen Universität Kaiserlautern
seit Feb. 2003	wissenschaflicher Mitarbeiter im Fachbereich Informatik an der Technischen Universität Kaiserslautern

Abschlüsse

September 1997	Abitur am Staatlichen Gymnasium Landstuhl
Januar 2003	Diplom Mathematik mit Nebenfach Informatik Technische Universität Kaiserslautern