Interactive Visual Analysis of Vector Fields

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

Vorgelegt von

Sven Bachthaler

aus Kirchheim unter Teck

Hauptberichter:	Prof. Dr. D. Weiskopf
Mitberichter:	Prof. Dr. H. Theisel
Tag der mündlichen Prüfung:	05. 06. 2013

Visualisierungsinstitut der Universität Stuttgart

2013

Contents

List of Abbreviations

CFD	computational fluid dynamics
CPU	central processing unit
CUDA	compute unified device architecture
GLSL	OpenGL shading language
HLSL	high level shading language
fps	frames per second
FTLE	finite-time Lyapunov exponent
GPU	graphics processing unit
LCS	Lagrangian coherent structures
LIC	line integral convolution

Abstract

Visualization is a very active research area due to several reasons. For years, data sets have been getting larger and more complex, increasing the difficulty of handling this data. Furthermore, in technical application areas, visualization is an essential part of the engineering process. These developments drive the need for improvements of all aspects of scientific visualization, as well as the integration of information visualization techniques.

This thesis focuses on the development of visualization and analysis techniques for different types of vector fields—vector fields representing the flow of air or water, but also magnetic fields and vector fields derived computationally from scalar fields. The different techniques that were developed to handle such fields are organized in three parts: the first part presents methods that visualize vector fields in dense manner. The second part discusses methods that rely on topological approaches—the complexity of the visualization is reduced by concentrating on features of the data. In the third and final part, continuous scatterplots are introduced, which are designed to analyze correlations in multivariate data sets.

In the first part, the goal is to show as much information as possible and using every available pixel of the viewport to do so. However, one of the challenges of dense visualization methods is to maintain interactivity for high resolution visualizations. A cluster environment is used here to offer increased rendering performance and memory size for large and complex data sets. Additionally, an animation-based approach is presented that allows one to decouple the line-like patterns of LIC from the direction of animation. This decoupling is desirable since perception research suggests that LIC-based techniques combined with animation are non-optimal for local motion detection of the human visual system.

The second part focuses on topological methods to filter the data and hence, reduce the complexity of the resulting visualization. For time-dependent vector fields, Lagrangian coherent structures are used to visualize space-time manifolds that represent the topology of these fields. Furthermore, the dynamic of such fields is visualized directly on these space-time manifolds, allowing us to quantify the hyperbolicity close to the topological skeleton. In addition, another technique is presented in the second part that allows one to visualize the topology of magnetic fields based on dipoles. Here, traditional topological methods are non-optimal, hence, an alternative topology is developed that

visualizes the existence and magnitude of magnetic flux between dipoles.

In the final part, the mathematical basis and several computational approaches are presented to compute continuous scatterplots. These plots are designed to work with data sets defined on a continuous domain, which is typical for scientific visualization data. In contrast to traditional scatterplots, they visualize the density in the data domain, instead of merely plotting data attached at discrete sampling positions. The additional computational approaches are an improvement of the original approach in terms of flexibility—they allow a trade-off between output quality and rendering performance, as well as the use of generic interpolation methods.

Zusammenfassung

Es gibt mehrere Gründe dafür, warum die Visualisierung ein sehr aktives Forschungsgebiet ist. Seit Jahren werden Datensätze größer und komplexer, was die Handhabung dieser Daten immer schwieriger macht. Des Weiteren ist die Visualisierung in technischen Anwendungsgebieten ein essentieller Bestandteil des Entwicklungsprozesses. Diese Entwicklungen führen zu der Notwendigkeit, alle Aspekte der wissenschaftlichen Visualisierung zu verbessern sowie zusätzlich Techniken aus der Informationsvisualisierung einzubinden.

Diese Dissertation konzentriert sich auf die Entwicklung von Visualisierungs- und Analysetechniken für verschiedene Arten von Vektorfeldern—Vektorfelder, die Luft- oder Wasserströmung repräsentieren, aber auch magnetische Felder und Vektorfelder, die rechnerisch aus Skalarfeldern abgeleitet wurden. Die verschiedenen Techniken, die für diese Felder entwickelt wurden, sind in drei Teilen organisiert: Der erste Teil präsentiert Methoden, die Vektorfelder auf eine dichte Art visualisieren. Der zweite Teil diskutiert Methoden, die sich auf topologische Verfahren stützen—die Komplexität der Visualisierung wird reduziert, indem nur die wesentlichen Merkmale dargestellt werden. Im dritten und letzten Teil werden kontinuierliche Streudiagramme eingeführt, die entwickelt wurden, um Korrelationen in multivariaten Datensätzen zu analysieren.

Das Ziel im ersten Teil ist es, so viel Information wie möglich zu zeigen, und dabei jeden verfügbaren Pixel dafür zu nutzen. Eine der Herausforderungen solch einer dichten Visualisierung ist es jedoch, hohe Interaktivität für hochauflösende Visualisierungen zu gewährleisten. Hierfür wird eine Clusterumgebung verwendet, um höhere Renderinggeschwindigkeiten und größeren Speicherplatz für große und komplexe Datensätze zu erhalten. Zusätzlich wird eine auf Animation basierende Technik vorgestellt, die es erlaubt, die linienähnlichen Muster von LIC von der Animationsrichtung zu entkoppeln. Diese Entkopplung ist wünschenswert, da die Wahrnehmungsforschung zu dem Ergebnis kommt, dass auf LIC basierende Techniken kombiniert mit Animation nicht optimal für das menschliche Wahrnehmungssystem sind.

Der zweite Teil konzentriert sich auf topologische Methoden, um Daten zu filtern und damit die Komplexität der daraus resultierenden Visualisierung zu reduzieren. Für zeitabhängige Vektorfelder werden Lagrange-kohärente Strukturen verwendet, um Raum-Zeit-Mannigfaltigkeiten zu visualisieren, die die Topologie dieser Felder repräsentieren. Des Weiteren wird die Dynamik

dieser Felder direkt auf den Raum-Zeit-Mannigfaltigkeiten veranschaulicht, wodurch eine Quantisierung der Hyperbolizität in der Nähe des topologischen Skeletts möglich wird. Im zweiten Teil wird auch eine Technik vorgestellt, die es erlaubt, die Topologie von magnetischen Feldern zu visualisieren, die auf Dipolen basieren. Herkömmliche topologische Verfahren sind hier nicht optimal, daher wird eine alternative Topologie entwickelt, die die Existenz und Stärke des magnetischen Flusses zwischen Dipolen visualisiert.

Im letzten Teil werden sowohl die mathematische Basis als auch mehrere Berechnungsansätze für kontinuierliche Streudiagramme vorgestellt. Diese Diagramme wurden für Datensätze entwickelt, die einen kontinuierlichen Definitionsbereich haben, wie es für wissenschaftliche Visualisierungsdaten oftmals der Fall ist. Im Gegensatz zu herkömmlichen Streudiagrammen visualisieren diese die Dichte in der Datendomäne, anstatt lediglich die Daten einzuzeichnen, die an diskreten Abtastpunkten vorliegen. Die zusätzlich vorgestellten Berechnungsverfahren verbessern den ursprünglichen Ansatz in Hinblick auf Flexibilität—sie erlauben den Ausgleich zwischen Ausgabequalität und Renderinggeschwindigkeit sowie den Einsatz von generischen Interpolationsmethoden.

1 Introduction

Visualization is the science of creating visual representations from raw data with the intent of gaining insight into this data.

Using visualization to explain models or theories is not a recent invention—the basic idea is hundreds or even thousands of years old. Leonardo da Vinci was probably the first one to study turbulence in water. He conducted experiments approximately in 1510 A.C. and created hand-made drawings to visualize his results. However, at that time, visualization was limited by the ability and skill of human artists.

Significant changes came with the immense increase of computational power towards the end of the 20^{th} century. During that time, the research area of visualization experienced the most notable forward leap in its development so far. Leveraging the computer's abilities, visualization scientists were now able to create images or animations based on simulated or measured data. Furthermore, it was now possible to quickly change parameters of a visualization—unwanted data could be filtered, viewing angles could be adjusted, and important parts could be highlighted to increase the effectiveness of the resulting visualization. Being able to *interactively* manipulate the visualization process is an important feature that was not available prior to that time.

In recent years, visualization has became not only widely present in science, but ubiquitous in every day life as well. Probably the most prominent examples are weather forecasts that are commonly presented with visualizations of clouds or air pressure. In technical application areas, visualization is now an essential part of the engineering process. Inspired by real-world physical experiments, computational visualization is used to examine and refine the design of components or entire vehicles, with examples in both automotive and aerospatial industries.

According to Möller and Tory [TM04], visualization is a research area that, historically, was often divided into two major areas called *scientific visualization* and *information visualization*. There is no exact and completely accepted definition of how to separate these two areas entirely, since examples exist that belong to both or neither area. Usually, the differentiation is performed based on the application area, whether data is physically based or abstract, and whether

spatialization is inherently given or chosen. A typical example of scientific visualization is vector data of a wind tunnel (air flow), whereas financial data is often seen as an example of a data source for information visualization. Some techniques presented in this thesis are clearly falling into the scientific visualization category. However, especially the techniques presented in the third part cannot be easily categorized into the first or the second category.

In contrast to real-world physical experiments, visualization on a computer is not restricted by the laws of nature—therefore, creating a visualization technique can use physical experiments as an inspiration, but far more advanced techniques can be realized where real-world counterparts are non-existant. A well known visualization metaphor that is based on physical experiments is a *streak line*: these lines are inspired from patterns of smoke in a wind tunnel. On the other hand, applying more abstract techniques like, e.g., topological methods, is only feasible in a computer-based visualization system.

Advanced visualization techniques are often based on results from other research areas such as color theory or cognitive psychology which are employed by visualization scientists to find visual metaphors. With the use of these metaphors it is possible to explain numerically, linguistically, or logically complicated relations which would otherwise be hard or even impossible to understand.

Nowadays, visualization—and especially scientific visualization—is a very important and active research area due to several reasons. Data acquired by simulations are getting larger and more complex, which leads to the problem that trivial visualization methods are not effective, or their effect is even reversed: instead of helping the user, trivial methods stand in their own way, are overstressing the user, and possibly lead to misinterpretations. Therefore, a lot of effort is put into the development of effective visualization approaches to avoid such problems.

1.1 Research Questions

In 2004, Johnson [Joh04] published a report that identified the top problems and issues in visualization research. Some of the problems mentioned by Johnson are addressed in this thesis—namely the efficient utilization of novel hardware architectures, perceptual issues, multifield visualization, and the integration of both scientific and information visualization strategies. The following paragraphs motivate in more detail how these problems are related to this thesis.

Interactivity is one important aspect of an application for scientific visualizations. Without an interactive application, the exploration of data sets is very tedious or even impossible—however, users often do not know beforehand where to look for interesting spots in the data set. Hence, gaining insight and

analyzing the data is tightly coupled with the ability of an application to produce interactive results.

Depending on data set size or complexity, the creation of such an application can be a challenging task—the data set might be too large for the computer's memory or the computation of new images consumes too much time. Sometimes, this kind of problem vanishes over time because of Moore's law—memory size and computing power are increasing exponentially. However, if awaiting this development is not an option, or the underlying algorithms do not scale well, more elegant solutions are required. Firstly, improving the underlying theoretical background and hence algorithms that implement this theory, and secondly, employing specialized hardware to accelerate computations. For the latter, this includes the use of a single graphics processing unit (GPU) and also the use of a cluster system in which several individual computers are bundled to increase both computational performance and available memory.

Putting aside the problem of data set size and complexity, further questions remain. One typical application area for visualization is flow simulation—such data represents the direction and magnitude of flow at the corresponding location. When considering time-independent flow, represented by a vector field that does not change over time, animation might be a helpful method to visualize the flow. However, simple animation methods do not take into account what psychology research suggests with regard to human perception. The question is: how can we combine the non-disputed power of animation with the knowledge about human perception?

Furthermore, vector fields are not restricted to flow—a magnetic field is a vector field as well, however, there is no motion of any matter involved. In contrast to flow visualization where it is often favorable to create a dense visualization, magnetic fields are better visualized using more abstract methods since individual field lines are of less interest here. However, the obvious approach of using vector field topology to support the analysis of magnetic fields proves to be less successful than expected—more suitable topological constructs are necessary to effectively help the user to find interesting structures in a magnetic field.

Finally, vector fields are commonly only one part of simulation results. Using flow simulations as an example, quantities like pressure or temperature are often of interest as well. In addition, derived quantities like the gradient or the magnitude of vectors frequently contain valuable information as well—features in these data domains are often related to interesting spots in the data set. These additional data dimensions are traditionally analyzed using parallel coordinates, scatterplots matrices, or single scatterplots. However, both parallel coordinates and scatterplots have the intrinsic property of displaying only discrete data samples—this can be a major drawback in scientific visualiza-

tion where data samples are spatially related and interpolation or reconstruction methods are applicable to obtain a continuous representation of the data. Therefore, improving the scatterplot idea further to arrive at a tool that effectively visualizes data correlations for scientific data sets remains an interesting problem.

1.2 Interactive Visualization

Visualization is a process that is formally described as a concept of data flowing through a pipeline. This *visualization pipeline*, as introduced by Haber and McNabb [HM90], consists of the stages data aquisition, filtering, mapping, and rendering. The result of the rendering stage can comprise both images as well as image sequences—either video animations or in the form of an interactive application.

Once the visualization pipeline is established for a data set, it creates output images by applying the aforementioned steps to this data. However, it is almost never the case that parameters of this pipeline are configured in such a perfect way that every aspect of the input data is captured with the first result. If the input data is unfamiliar or newly acquired, the need for iteratively adapting the parameters of the visualization pipeline is even more evident. The concept of *exploration* is oftentimes the key to identify regions of interest and finally gain insight to a data set. For this reason, the process of visualization requires feedback from the user based on the current output by allowing manipulation of each stage in the pipeline.

In practice, a major requirement for the exploration of a data set is *interactivity*. A system is called interactive if it is able to provide feedback to user requests in a timely manner. There is no absolute definition of "timely" in the form of a fixed time span; it must merely be short enough to allow an explorative work flow with the system. Therefore, the term *interactive visualization* refers to systems with highly varying feedback times—ranging from several seconds to produce a single image up to systems with real-time behavior that render several frames per second (fps).

1.3 Vector Fields

In this thesis, the focus lies on *vector fields* as input data to the visualization pipeline. A field is a defined region in space that stores data at distinct locations within this region. However, for most scientific data sets, it is important to mention that the *domain* of the field is assumed to be continuous, i.e., data is available at every location within the domain, not only at discrete locations. Depending on the type of data that is stored at these locations, the field is called a scalar, vector, or tensor field.

Mathematically, a vector field is defined as a vector-valued function that assigns each point $\vec{x} = (x_1, x_2, ..., x_n)$ in the domain $D \subset \mathbb{R}^n$ a vector $\vec{v} = (v_1, v_2, ..., v_n)$:

$$\vec{v} : D \longrightarrow \mathbb{R}^n. \tag{1.1}$$

When visualizing vector data, it is important to consider the dimensionality of this data. Spatially, vector data is usually either 2D or 3D. Corresponding visualizations of such data are also called 2D or 3D, respectivly. However, there is the special case of visualizations that show 3D vectors on 2D curved surfaces ("hypersurfaces") embedded in 3D space. In other words, the originally three-dimensional vector field is intersected by a hypersurface, and tangential vectors are visualized on the hypersurface—therefore, the resulting visualization is called *2.5D*. Examples of 2.5D visualizations are presented in Chapters 2, 3, and 4. Such a tangential vector field $\vec{v}_T$ can be defined on a smooth and orientable manifold M (with or without boundary) as follows:

$$\vec{v}_T : M \longrightarrow TM \quad \text{with } \vec{v}_T(x) \in T_xM.$$

This vector field maps points $\vec{x} \in M$ to a vector in the corresponding tangent space at that point, T_xM. If the vector field is not tangential, it can be modified to a related tangential vector field that stays on the manifold M by removing the normal parts from the vectors.

A vector field can also be considered as a representation of the state of a dynamical system, described by differential equations. The evolution of points in this system is given by the solutions of

$$\vec{x}(t) = \vec{v}(\vec{x}(t), t). \tag{1.2}$$

Due to the similar nature of this model to fluid dynamics, vector fields are often referred to as *flow*. However, vector fields in fluid dynamics (in contrast to some vector fields in this thesis) have to fulfill several additional requirements, e.g., the Navier-Stokes equation, to represent physical flow.

The temporal dimension of flow data is also used to classify the corresponding vector fields as either *steady* if they are independent from time t, and *unsteady* if they are time-dependent, i.e., the flow data itself changes over time. Synonyms to these terms are *time-dependent*, or *-independent*, respectively, which are used throughout this thesis as well.

In scientific visualization, vector fields are usually represented as sampled data sets—vectors are stored at discrete locations only. As a result, data values between these locations are unknown. However, for most data sets, it is a valid assumption that the actual data is smooth, well-behaved, and the underlying domain is continuous—allowing to interpolate or reconstruct data values between sample locations. The result of an interpolation or reconstruction method is an analytic description based on the available discrete data samples.

A trivial interpolation approach is called *nearest neighbor*, where no additional data values are computed, but the (spatially) nearest data sample is identified. The result is a piecewise constant function, since the data value of the identified sample is returned as the interpolation result.

Nearest neighbor interpolation results are often not precise enough, therefore, *linear* interpolation is an alternative that linearly weighs the data values of neighboring samples, resulting in a piecewise linear function. *Barycentric* interpolation is a representative of linear interpolation methods and is used to compute data values inside triangles or tetrahedra.

For results of higher quality compared to linear interpolation, more advanced schemes exist. For example, higher-order polynomials are introduced to allow for curvature between neighboring data points, which is determined based on the information of surrounding data samples.

1.4 Vector Field Visualization

Visualizing vector fields has been a central topic in scientific visualization for years with a variety of application examples from engineering disciplines and sciences [Wei06]. The vast amount of work in this area can be categorized according to Laramee et al. [LHD+04] in four categories: direct, geometric, dense (texture-based), and feature-based flow visualization.

Direct vector field visualization tries to represent the data without translation to more abstract approaches. One example of this category is color coding of flow velocity, which works fairly well in the case of two-dimensional domains. Additional examples of this category are given in Section 1.4.1.

Although direct flow visualization delivers an overall picture of the flow, it is not well suited to communicate its long-term behavior. Geometric flow visualization tries to overcome this issue by integrating along the flow, using geometric objects for visualizing its properties. Examples of this category are stream lines, streak lines, and path lines, which are detailed in Section 1.4.2.

Both aforementioned categories more or less depend on the appropriate placement of seed points that are the basis for this kind of visualization. However, this is oftentimes a challenging task leading to the development of dense visualization techniques that completely avoid the seed point generation problem. A dense visualization commonly uses textures that are filtered with the local values of the flow to generate a visual pattern that indicates the direction of the vector field. Several sections in this thesis are based on dense flow visualization techniques, therefore, a detailed introduction to such methods is given in Section 1.4.3.

Feature-based flow visualization can be seen as the antipole of dense approaches. Instead of capitalizing on each pixel of the image and making use of

as much room as possible to display information, feature-based visualization extracts relevant information. The result is then based on the reduced data to avoid visual complexity and clutter while delivering an efficient flow visualization at the same time. Topology-based methods are an exemplary representative of this category, as they show only the skeleton of vector fields. The key ingredients of topological methods are *critical points* (i.e., points where the magnitude of the vectors is zero) and *separatrices*, i.e., lines that separate regions of different flow behavior. Further explanations of topology-based visualization are given in Section 1.5. In some cases, these topological constructs are related to physical properties. An example of such a physically meaningful structure is the *connectrix*, which is presented in Chapter 5.

1.4.1 Visualization with Glyphs

One of the oldest ways of visualizing a vector field is to draw geometric primitives, or *glyphs*, at selected sample positions in the vector field domain. These glyphs are designed to show direction, magnitude, or both of the vector field at the current location. The by far most common glyph is the arrow. Visualizations with arrow plots are used in this thesis with the first example shown in Figure 1.2(a). On the one hand, this technique is very easy to implement and requires practically no previous knowledge of the user, on the other hand, as already mentioned, its usefulness is limited to purveying only a rough idea of the underlying vector field. Post et al. [PPWS95] give an overview of visualizations based on glyphs.

Please be aware that scatterplots can be seen as a representative of the glyph-based visualization category as well. Being part of the family of data plots, scatterplots have been proven as successful and useful diagramming techniques in descriptive statistics and information visualization. They take discrete data points with two data dimensions as input, and produce a 2D plot of those data points by drawing respective dots on a diagram with two orthogonal axes repesenting the two data dimensions.

Although not being able to visualize the spatial properties of a vector field, 2D scatterplots are a useful tool for identifying correlations (or the lack of correlation) between data dimensions. These plots can be either 1D (commonly referred to as "histogram"), 2D or 3D with examples shown in Figure 1.1. Higher dimensional data can be visualized with a scatterplot matrix, i.e., several 2D scatterplots are aligned in a matrix so that their data dimensions can be varied to cover all possible combinations of data dimensions. Alternatively, correlations of several dimensions can also be identified using *parallel coordinates* [ID90].

For three data dimensions, three-dimensional scatterplots can be applied, however, they are often affected by problems involving occlusion, as well as visual perception problems. One way of overcoming some of these problems is

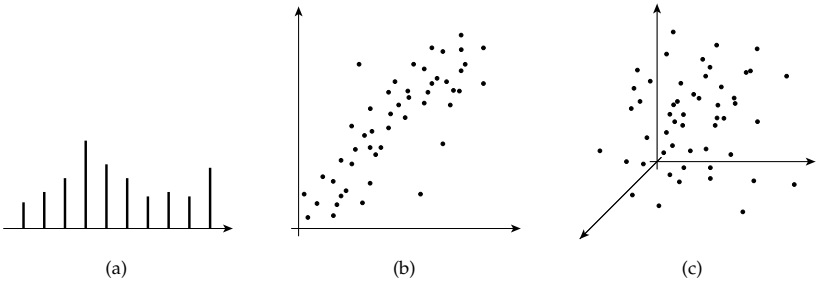


Figure 1.1: Example of data plots with different dimensionality: (a) 1D, (b) 2D, and (c) 3D.

shown by Sanftmann and Weiskopf with illuminated scatterplots [SW09] and with an interpolation and projection technique for 3D scatterplots that allows the user to exchange data dimensions smoothly [SW12]. Note that scatterplots showing only two data dimensions are not affected by such problems. In this case, discrete 2D data points are taken as input, and a 2D plot of those samples is produced by drawing respective glyphs (usually dots) on a diagram with two orthogonal axes representing the two data dimensions.

A novel visual representation of scatterplots is presented in the work of Chan et al. [CCM10] where local variation in data dimensions are highlighted. This results in so-called "flow-based scatterplots" that help the user to identify interesting changes of one variable with respect to the second mapped variable. Based on ideas of flow field analysis, operations are introduced to scatterplots that efficiently allow selecting points of interest.

The SimVis system [DGH03] presents an approach that shows how the combination of scatterplots and vector field visualization can be used effectively to analyze various vector fields. With this system, it is possible to find correlations between multiple data dimensions and with brushing and linking the spatial location of interesting features found in those data dimensions can be identified and further analyzed.

Traditional scatterplots are designed to work with discrete data only, however, scientific data sets are commonly interpreted as fields where interpolation and reconstruction methods can be applied, as described in Section 1.3. Chapter 6 shows how *continuous scatterplots* overcome this drawback and how they are applied to scientific data sets.

1.4.2 Visualization with Lines

Falling into the category of geometric flow visualization, so-called *characteristic lines* are able to convey the properties of a vector field that would otherwise be hard to depict with glyphs. Representatives of this category are *path*, *stream*, *streak*, and *time lines*. Please note that these lines with the exception of time lines are identical in steady vector fields, but are different in unsteady fields. The following paragraphs elaborate on the details of these field lines.

Equation 1.2, together with the condition $\vec{x}(t_0) = \vec{x}_0$, is the starting point for an initial value problem of an ordinary differential equation. Path lines are a solution to this problem, since they form an integral curve defined by

$$\vec{x}_{\text{path}}(t_1; \vec{x}_0, t_0) = \vec{x}_0 + \int_{t_0}^{t_1} \vec{v}(\vec{x}_{\text{path}}(t; \vec{x}_0, t_0), t)\mathrm{d}t.$$

These lines are called path lines, since such curves describe trajectories of massless particles which are released into a vector field. In a real-world experiment, path lines can be obtained by photographic long-time exposure of markers that are advected by flow.

In contrast to path lines, the concept of stream lines differs with respect to the time t. Here, a "snapshot" of a specific point in time t_i is considered, thus leading to a computation of the integral curve as follows:

$$\vec{x}_{\text{stream}}(t_1; \vec{x}_0, t_0) = \vec{x}_0 + \int_{t_0}^{t_1} \vec{v}(\vec{x}_{\text{stream}}(t; \vec{x}_0, t_0), t_i)\mathrm{d}t.$$

As opposed to path lines, stream lines are unique in the sense that they do not cross each other. This becomes apparent when considering that a particle cannot have two different velocities at the same point in a vector field.

The concept of streak lines is closely related to real-world physical experiments, where particles are continuously released at a fixed position $\vec{x}_0$ at times $t_i \in [t_{\text{min}}, t_1]$, e.g., dye released into water at a certain position, or a chimney releasing smoke into the air. A snapshot of these particles is then taken at time t_1, resulting in a pattern which is called a streak line:

$$\vec{x}_{\text{streak}}(t_i; \vec{x}_0, t_1) = \vec{x}_{\text{path}}(t_1; \vec{x}_0, t_i).$$

These streak lines were generalized by Wiebel et al. [WTS+07] in the sense that the source location $\vec{x}_0$ is not necessarily fixed anymore.

The fourth representative of characteristic lines are constructed by releasing a set of particles placed on a seed curve $\vec{x}_0(s)$ simultaneously at a distinct point in time t_0. Again, a snapshot is taken after some time at time t_1. The set of particles creates a curve which is displaced by the vector field over time and is called a time line. Time lines are given by

$$\vec{x}_{\text{time}}(s, t_1; \vec{x}_0, t_0) = \vec{x}_0(s) + \int_{t_0}^{t_1} \vec{v}(\vec{x}_{\text{time}}(s, t; \vec{x}_0, t_0), t)\mathrm{d}t.$$

Generalizing these characteristic lines to three dimensions results in surfaces, which are named according to their 2D counterpart.

1.4.3 Line Integral Convolution

For this thesis, especially dense, texture-based vector field visualization is of interest. Techniques of this sort use characteristic lines to create patterns that visualize the vector field. In order to avoid the problem of finding appropriate seed points for these characteristic lines, they cover the whole domain in a dense fashion.

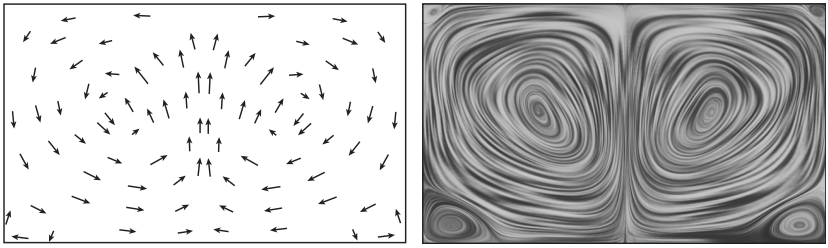
The most prominent representative of dense, texture-based techniques is line integral convolution (LIC), introduced by Cabral and Leedom [CL93]. Next to the technique called Spot Noise, published by van Wijk [vW91] two years earlier, LIC was one of the first methods that was able to visualize the vector field in a dense fashion and to accurately reflect vector fields with high local curvature. The basic idea of LIC is to visualize the vector field with long line patterns. The basic primitive on which this technique relies on is a noise texture. This texture spans the entire vector field domain and is convolved, or "smeared", along the path of stream lines. More precisely, the intensity I of a pixel at location $\vec{x}_0$ in the resulting image is computed according to

$$I(\vec{x}_0) = \int_{-L}^{L} k(t) N(\vec{\phi}_0(t)) \mathrm{d}t,$$

where $[-L, L]$ defines the support of the convolution kernel k, which depends on the parameter t denoting the time. Furthermore, the input noise texture is denoted as N and $\vec{\phi}_0(t)$ is the stream line that passes $\vec{x}_0$ at $t = 0$.

The convolution is commonly performed using a symmetric kernel, e.g., a box, tent, or preferrably a Gaussian kernel. An asymmetric kernel creates directional cues which are helpful if it is important to see the orientation of LIC lines. The result of this computation is a visual pattern with a high intensity correlation along stream lines, but only little correlation perpendicular to these lines. An example of a vector field visualization using LIC with a Gaussian convolution kernel is shown in Figure 1.2(b).

The original approach of LIC was extended into several directions. One major drawback of the original approach was the slow execution speed on hardware available at that time. Stalling and Hege [SH95] improved the original approach and presented Fast LIC, which was approximately one order of magnitude faster than the original. They achieve this mainly by reusing similar convolution integrals along a stream line. More precisely, their convolution method follows a different approach than the original method of Cabral and Leedom: instead of traveling along a stream line and gathering intensity values from the noise texture, Stalling and Hege employ a depositing scheme



(a) (b)

Figure 1.2: Example of the visualization of a Benard convection using (a) arrow glyphs and (b) line integral convolution.

that stores intensity values at pixel locations. For the convolution process, these values are collected during integration. Forsell [For94] extended the original LIC approach to work not only on uniform grids, but also on parametric surfaces like curvilinear grids. In addition, vector magnitude is visualized using an animation technique. Battke et al. [BSH97] presented a method that is based on Fast LIC and which is not restricted to curvilinear grids but works on arbitrary grids as well. However, the results presented in their paper [BSH97], are limited to relatively small grids composed of equilateral triangles. Weiskopf and Ertl [WE04] proposed an approach that overcomes this issue and works on arbitrary meshes as detailed in Section 1.4.5. Recently, Hlawatsch et al. [HSW11] published hierarchical line integral convolution, which allows one to compute long trajectories based on previously computed, shorter ones. Especially for higher-order integration in higher-order data, performance increases considerably.

Interrante and Grosch [IG97, IG98] extended Fast LIC to visualize 3D flow. To do this, they have to handle perceptual challenges like occlusion, depth perception, and visual clutter. Their approach, called Volume LIC, uses halos to improve depth perception. Occlusion and visual complexity is reduced by using a sparse noise texture. All previously mentioned approaches only operate on steady vector fields. Shen and Kao [SK97] overcome this limitation with Unsteady Flow LIC—an extension that introduces an adapted convolution method which works with path lines instead of stream lines to handle time-dependent vector fields.

1.4.4 Image-Based Flow Visualization Methods

In the dense, texture-based flow visualization category are further approaches, that, in contrast to LIC-based methods, rely on moving texels or

texture-mapped polygons.

Image Based Flow Visualization (IBFV) is a technique developed by van Wijk [vW02] which is able to visualize unsteady flow. The method is based on the advection and decay of textures. This approach is image-based as it warps white noise textures according to the direction of the flow. These textures are then blended with the current image in a frame-to-frame fashion. Instead of advecting each pixel of the texture, small quadrilaterals are deformed, leading to higher performance when compared to many other texture-based visualization methods.

A hardware-accelerated technique called Lagrangian-Eulerian Advection (LEA) was published by Jobard et al. [JEH01] which reaches interactive frame rates as well. This approach produces images with high spatio-temporal correlation, as particle paths are integrated in a Lagrangian step, while the color distribution is updated on the grid (i.e., the Eulerian step). This hybrid approach has the advantage that the dense aspect is guaranteed by the Eulerian step, which reduces the amount of particles that are necessary for the Lagrangian step to the amount of pixels in the final image. Similar to IBFV, subsequent time steps are blended to achieve spatial coherence.

Image-Space Advection (ISA) was presented by Laramee et al. [LJH03] and incorporates features both from LEA as well as from IBFV. This approach is able to produce animated textures on arbitrary 3D surfaces based on the IBFV approach, i.e., textures are advected and blended in image-space. ISA and IBFV are technically closely related—their similarities, and also their differences, are detailed in the work of Laramee et al. [LvWJH04].

An inherent problem of the restriction to image-space is that edges of the underlying geometry in object-space are ignored, which can lead to undesired visual continuity across silhouettes or edges. As an example, ISA uses edge detection mechanisms to stop the advection process if an edge is detected at the current processing location.

1.4.5 Hybrid Physical-/Device-Space LIC

Following the basic idea of LIC, the hybrid physical-/device-space approach presented by Weiskopf and Ertl [WE04] creates a vector field visualization on 2D surfaces that are embedded in 3D space. In this thesis, several chapters attend to the problem of visualizing vector fields on curved surfaces as well. Although alternatives to the hybrid approach exist and are shown in Section 1.4.4, this approach retains the benefits of image-space techniques and avoids inflow issues at silhoutte lines at the same time. Therefore, the hybrid approach of Weiskopf and Ertl was chosen as a base to build on, justifying an in-depth introduction to this technique in the following. Please note that parts of this section are adopted from our publication [BSWE06].

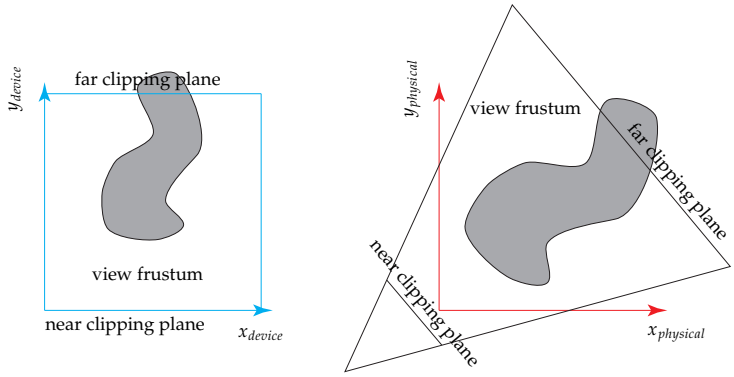


Figure 1.3: Coupled D-space (left) and P-space (right) representations of the same scene. Image adopted from [WE04].

The hybrid physical-/device-space approach adopts a Lagrangian view to particle tracing to compute LIC-like images. Particle paths are computed as introduced in Section 1.4.2. The positions $\vec{x} \in \mathbb{R}^3$ are restricted to locations on the surface embedded in $\mathbb{R}^3$. For a tangential vector field, the resulting curves stay on the surface.

So far, the vector and point quantities are given with respect to physical space (P-space). The basic idea of image-space methods is to perform the relevant computation in image-space. In fact, the image-space operations are performed in normalized device-space (D-space), which has the extent $[0, 1]^3$. D-space is ideal to compute LIC on a per-pixel basis with respect to the image plane, which results in a largely output-sensitive algorithm and a uniform density on the image plane. On the other hand, there are some aspects that are better represented in P-space, in particular, the 3D noise input for LIC in order to guarantee frame-to-frame coherence under camera motion. The advantages of P-space and D-space representations are combined by computing particle paths in both spaces simultaneously, as illustrated in Figure 1.3.

Explicit numerical integration, such as a first-order explicit Euler scheme, works with P-space coordinates $\vec{x}_P \equiv \vec{x}$ and the original tangential vectors $\vec{v}_P \equiv \vec{v}$ to solve the particle-tracing equation. After each integration step, the corresponding position in D-space is computed. The vector field is no longer given on a P-space but a D-space domain, i.e., there are different representations for the vector components and the associated point on the surface. The modified particle-tracing equation then is

$$\vec{x}_P(t) = \vec{x}_P(t_0) + \int_{t_0}^{t_1} \vec{v}_P(\vec{x}_D(t), t)\mathrm{d}t, \tag{1.3}$$

where $\vec{x}_D$ and $\vec{x}_P$ represent the same position with respect to D-space and P-space, respectively.

The crucial step in making the integration process efficient is to reduce the 3D representation of the quantities to a 2D D-space representation when possible. Since flow fields are assumed to live on opaque surfaces, only the closest surface layer needs to be considered. Here, the depth component can be indirectly computed because the depth values of the surface on which the visualization is computed are known.

The algorithm consists of two major parts. In the first part, the 2D textures for starting positions $\vec{x}_P$ and the vector field $\vec{v}_P$ are initialized by rendering the mesh representation of the hypersurface. The closest depth layer is extracted by the z-test. The P-space positions are set according to the surface's object coordinates. The vector field texture is filled by $\vec{v}_P$, which originates from slicing through a 3D texture that holds the vector field data set. Because the vector field is usually not tangential from construction, it has to be made tangential by removing the normal component, which is computed according to the normal vectors of the surface mesh. In the second part, Equation 1.3 is solved by iterating over integration steps. This part works on the 2D $x - y$ subdomain of D-space, and it successively updates the coordinates $\vec{x}_P$ and $\vec{x}_D$ along the particle traces, while simultaneously accumulating contributions to the convolution integral.

The implementation of the complete visualization process can be split into three stages: the projection stage, which projects the surface geometry and the vector field onto the image plane, the LIC stage, which computes the line integral on the image plane, and the blending stage, which combines a LIC image with the rendered image of the surface geometry. All three stages are implemented by vertex and fragment programs to make use of the high processing speed of GPUs.

The projection stage produces three 2D textures as intermediate results: the projected vector field $\vec{v}_P$, the start point for particle tracing $\vec{x}_P$, and the rendered image of the illuminated surface. These three textures are filled in a single rendering pass by using multiple render-targets. The performance of the projection stage is comparable to the performance of rendering the surface with illumination being enabled—only a few instructions are added that project the vector field and store the initial coordinates for particle tracing.

In contrast, the LIC stage is computationally more expensive. This stage uses intermediate results from the projection part to compute the line integral. It solves the particle tracing Equation 1.3 to advance positions along stream lines. Simultaneously, contributions to the line integral are accumulated along the

stream line. The input noise is stored in a 3D texture. Potential aliasing due to perspective foreshortening is avoided by an adapted version of MIP mapping that compensates the effects of perspective projection: first, for each seed point, an independent scaling factor is computed to achieve a constant image-space noise frequency. Second, the noise scaling factor is discretized in order to achieve frame-to-frame coherence under camera motion. The result of this stage is written to a 2D texture that holds the gray-scale LIC image on the image plane. Particle tracing and integral accumulation are implemented in a shader loop that advances along the stream line. Typically, the number of iterations is between 40 and 300. Therefore, the overwhelming performance costs are associated with the LIC stage. Note that the LIC computation is only performed for pixels that are covered by the projected surface geometry, i.e., fragment processing is skipped for background pixels by means of the early z-test. The z-values for this masking are obtained from the projection stage. Masking leads to a rendering run time that is proportional to the number of visible pixels, i.e., output sensitivity is achieved to a large extent.

Finally, the blending stage combines the result of the LIC stage and the lit surface generated in the projection stage by blending and modulation. In this way, both the LIC texture and the surface geometry are visualized at the same time. This flow visualization technique completely recomputes the LIC image for each frame. Therefore, the rendering performance is not affected by deforming or changing the surface geometry, or by moving the camera. In this way, this approach is suited for interactive applications in which visualization parameters can be rapidly changed by the user.

1.5 Topology of Vector Fields

Classical vector field topology partitions a given vector field into areas of qualitatively different flow behavior. In order to segment a vector field in such a way, *critical points* must be identified. Locations $\vec{x}_c$ in the field must meet two conditions to be classified as a critical point of first order. First, for the vector field must apply $\vec{v}(\vec{x}_c) = 0$, thus, stream lines started at $\vec{x}_c$ degenerate to a single point. This single point is called stationary or, alternatively, constant orbit. The second condition requires that the determinant of the velocity gradient is non-zero at a stationary point. Then, this point represents an isolated zero and is called a critical point.

A critical point can be classified based on the behavior of the vector field around its location. This can be done by performing an eigenvector analysis of the velocity gradient at the corresponding location. Possible cases in two dimensions are sources, sinks, and saddles as well as focuses and centers. An overview is given in Table 1.1, where the classification of critical points in relation to their eigenvalues is shown.

two real eigenvalues	type
both positive sign	source
both negative sign	sink
opposite signs	saddle
two conjugate complex eigenvalues	type
positive real parts	focus source
negative real parts	focus sink
purely imaginary	center

Table 1.1: Classification of first-order critical points in 2D vector fields.

Saddle-type critical points play an important role in vector field topology since they give rise to distinguished stream lines—a stream line converging to a saddle-type critical point in forward-time (or converging to the same point in negative time, which means that the stream line diverges from the saddle). Please note that there is no stream line at the location of the critical point. To compute stream lines, a small step is made away from the critical point in the direction of the eigenvectors; the direction of a stream line is determined by the direction of the eigenvectors as well. These distinguished stream lines are called *separatrices*, since regions of qualitatively different flow behavior are separated by them.

Of further interest to vector field topology are periodic orbits. Periodic orbits are stream lines in a vector field that pass a certain point $\vec{x}_0$ more than once. More precisely, a periodic orbit exists in a vector field if there is a time span $T > 0$ such that $\vec{x}(t) = \vec{x}(t + T)$.

The visualization of all critical points, periodic orbits, and the corresponding separatrices is called the *topological skeleton*. This concept of vector field topology was introduced by Helman and Hesselink [HH89] to the visualization community.

1.5.1 Lagrangian Coherent Structures

Applying vector field topology to create a visualization that shows the topological skeleton of a vector field has proven useful for many applications. An inherent problem of vector field topology is, however, that it is built upon stream lines, and not path lines. Therefore, it is only able to create an instantaneous view of the vector field, which works well for stationary fields. In the case of non-stationary fields, the results of vector field topology are often inappropriate. This was shown, e.g., by Shadden et al. [SLM05] with their "double

gyre" example—separatrices are dislocated and do not show the flow separation properly.

Lagrangian coherent structures (LCS) are an alternative to vector field topology that work for time-dependent vector fields as well. LCS are not finally defined, which naturally leads to deviating definitions in the literature. Relevant for this thesis is the definition of Haller [Hal01], which states that LCS are ridges in the finite-time Lyapunov exponent (FTLE) field—a field that is computed based on path lines. Consequently, LCS are an appropriate alternative to vector field topology in unsteady vector fields, since they correctly reflect advection processes.

Since LCS are based on the FTLE, this concept is defined first. The Lyapunov exponent is employed to measure the growth of pertubations in a time-dependent field or dynamical system. Speaking informally, this value is obtained by considering two particles located closely together in the field—the advection of these two particles is then observed for a defined time span T after which the distance between the particles is measured. Non-positive separation indicates regions where the flow behaves in a predictable way, whereas positive separation is usually correlated to regions with chaotic behavior, as the particles are separated away from each other. The term "finite-time" in FTLE indicates that the practical computation of this value only approximates the limit solution, as it only considers a finite time interval for the advection analysis.

Once the FTLE is obtained for every location in the vector field, LCS can be computed based on this preliminary step. If the FTLE field is interpreted as a height field, then LCS are located at the center of elevational crests or ridges. In practice, there are no absolute values that can be used to identify ridges indiscriminantly, which means that the outcome of an LCS computation heavily depends on the time span T and the advection step size used for the computation of the FTLE field. In addition, the outcome depends on which ridges are regarded as "sharp" enough to assume a LCS at their center [SLM05]. As will be shown in Section 4, obtaining proper LCS is based on empirically finding appropriate parameter values.

1.6 Multi-Attribute Fields

Data generated in today's simulations is not only getting larger and larger, but also more complex with respect to data dimensionality. As an example, physics-based simulations offer several data dimensions that are correlated to each other. A vector field representing air flow can be accompanied by additional data channels containing e.g., air pressure, temperature, humidity, and so on. For such data sets, scatterplots are a highly useful tool to comprehend relations between these data dimensions. To name an example, the Bernoulli

effect can be easily identified by mapping velocity and pressure to the axis of a scatterplot. According to this effect, regions of low pressure are found if high velocities are present.

To start an analysis, the user has to choose two data dimensions which are then mapped to the axis of a 2D scatterplot. Many data sets contain more information than a vector for each data sample—these additional *data dimensions* can contain information like temperature, pressure, or humidity to name just a few. However, even such additional dimensions are not available, Kniss et al. [KKH02] discuss an approach that can be applied in such a case. They propose mapping a scalar value to the horizontal axis and deriving a second data dimension based on this first one. Data for the second data dimension is created by computing the magnitude of the gradient of the first data dimension. If there are boundaries present in the first data dimension, they manifest themselves in the scatterplot as arc-like structures. Such boundaries can be highly interesting, e.g., in medical data sets where several layers of organic matter are close to each other.

Based on the approach of Kniss et al., not only transfer functions can be designed to reveal boundaries. Similarly, *brushing and linking* [BC87, GRW+00] can be employed to select the arc-like structures in the scatterplot and highlight according regions in the spatial domain. Brushing and linking is a technique that connects the abstract data visualization of scatterplots with, e.g., the spatial visualization of volume rendering. Data samples that are selected in the data domain are identified in the spatial domain and highlighted there. In this way, the spatial origin of data samples selected in the data domain is visible.

1.7 Graphics Processing Units

GPUs were originally relatively simple-designed hardware used as graphical output buffer and only able to perform basic manipulations with the goal of transforming polygonal geometry into raster graphics. The programmable graphics pipeline [LKM01] introduced a considerable change as GPUs can now be used to execute arbitrary programs—even programs that are not related to graphical problem settings at all. One major difference to CPUs is the massively parallel nature of GPUs. In this thesis, CPUs with up to four cores were used to execute programs, while GPUs in the same system perform computations with 512 cores. This is highly efficient for rendering raster images, as pixels can often be colored independently from each other in a massively parallel fashion.

Generating images starts with an application executed on the CPU and ends with writing pixel data to the frame buffer on the GPU. This process, which includes several stages, is referred to as the *rendering pipeline*. A simplified version of the rendering pipeline is shown in Figure 1.4. The pipeline starts

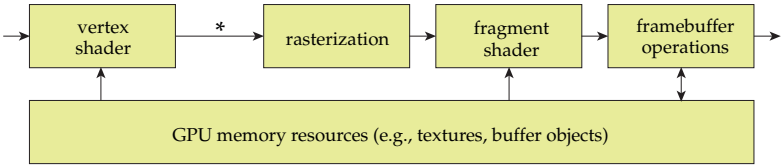


Figure 1.4: Simplified graphics pipeline with programmable vertex and fragment shaders. The asterisk (*) indicates the position of optional tesselation and geometry shaders.

with render calls that are executed on the CPU and define a set of vertices, their topology, and additional data like color, normals, or texture coordinates.

The next step in the pipeline is *vertex processing*. As the name indicates, this stage processes vertex data, however, for each vertex on an individual basis. This stage can be manipulated using a *vertex shader*, a short program which is used to transform vertices to so-called clip coordinates. However, this stage can also be used to generate any necessary data not directly related to vertices.

Recently, tesselation and geometry programs have been introduced that are located in the rendering pipeline between vertex and fragment shaders. Tesselation shaders make it possible to control the tesselation level depending on the distance to the virtual camera, allowing fine control in scenarios where dynamic level-of-detail is of interest. Geometry shaders are executed on individual output primitives and have therefore, in contrast to vertex programs, access to all vertices of a primitive. With geometry shaders it is possible to add or remove vertices as necessary, thus allowing mesh refinement or culling of unwanted primitives. In this thesis, tesselation and geometry shaders are not of interest since complex meshes are not used. Instead, textures are generated which are manipulated with fragment shaders that are located further down the pipeline.

Rasterization follows vertex processing and the optional tesselation and geometry shaders and filters its input of primitives by discarding those that are not contributing to the final image: primitives that are outside of the view frustum or those that are only visible from the back side. Recent generation graphics hardware allows one to program this stage as well.

One of the later steps in the rendering pipeline is *fragment processing*, where the final color and depth value of individual pixels is computed in a *fragment shader*. Here, access to texture memory also allows one to access pixel data of previous render steps. In this way, images can be created iteratively with a mechanism called *render-to-texture*, where the output is not written to the frame buffer, but into a *render target* instead.

Programming these stages with shader programs can be a challenge, especially when the desired algorithm is not related to graphic computations. Although the available programming languages like HLSL for DirectX applications or GLSL for OpenGL provide a high level of abstraction, certain limitations may require complicated approaches to create a solution which would otherwise be easy to implement, e.g., on a CPU.

To overcome the rigid programming model of the rendering pipeline and to provide hardware acceleration with GPUs at the same time, efforts were made that are now commonly referred to as *General-Purpose computation on Graphics Processing Units (GPGPU)*. Instead of forcing a general problem into a programming paradigm using only graphical primitives, the GPGPU perspective allows one to (almost) freely program GPUs. The Compute Unified Device Architecture (CUDA) [NVI11] is such an environment that allows massively parallel execution of code on the GPU. Since CUDA is dependent on the vendor NVIDIA, alternatives were developed. The Khronos group created the Open Computing Language (OpenCL) [Khr11], which is an independent standard for various kinds of GPUs and multicore CPUs.

1.8 Outline

This thesis is organized in three parts that describe techniques to visualize and analyze vector fields. These three parts are organized in a way that subsequent parts handle vector fields with an increasing level of abstraction.

The first part treats methods that directly visualize vector data in a dense manner—the field is visualized by showing as much information as possible and capitalizing on each pixel of the viewport. Chapters 2 and 3 describe these texture-based mapping and rendering techniques.

The second part of this thesis increases the level of abstraction since it presents structural methods: topological features of vector fields and their visualization are the focus of this part. These feature-based techniques are described in Chapters 4 and 5.

In the third part of this thesis, the abstraction is increased again by transitioning to the data domain. Here, at first glance, the spatial aspect of data samples in a field is neglected. Instead, two data dimensions are analyzed with respect to correlations using scatterplots. Despite the decoupling of the visualization from the spatial domain, techniques like brushing and linking can be used to restore this connection again. The third part is comprised of the Chapters 6, 7, and 8. Figure 1.5 summarizes how the methods described in each chapter are organized with respect to the previously mentioned level of abstraction.

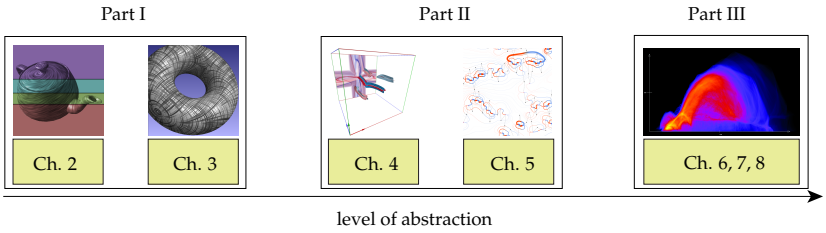


Figure 1.5: A structural organization of techniques described in this thesis.

1.9 Contributions

The focus of this thesis was the development of improved visualization and analysis techniques for different types of vector fields, with the following contributions. Please note that the papers that originated from my dissertation research were published in collaboration with my PhD advisor. Other collaborators are mentioned explicitly below. Parts of these publications and material thereof are used in this thesis.

Parallel Vector Field Visualization on Curved Surfaces (Chapter 2, [BSWE06])

GPU-based visualization techniques for vector fields on curved surfaces are an alternative to full 3D approaches that offer the completeness and flexibility of the visual representation, but avoid the perceptual problems of true 3D methods at the same time. However, the approach that was used as a basis to build on is strongly influenced by image resolution and suffers from low rendering rates for high resolution visualizations. The main contributions presented in this chapter offer increased rendering performance and increased memory for larger and more complex data sets by employing a cluster environment to create the final result. A hybrid sort-first sort-last rendering scheme is applied to distribute the workload evenly to all cluster nodes and to increase the available texture memory at the same time.

The method described in this chapter is based on my diploma thesis, but was reworked for publication with additional data sets. The resulting paper [BSWE06] was written in collaboration with my diploma thesis advisors Daniel Weiskopf, Magnus Strengert, and examiner Thomas Ertl.

Animation of Orthogonal Texture Patterns for Vector Field Visualization (Chapter 3, [BW08a])

Animation is an established technique to visualize not only direction but also the magnitude of flow. However, traditional techniques that use LIC-like approaches to visualize vector fields were not designed to observe results of per-

ception research with respect to animations. In contrast, the approach presented here decouples the line-like patterns used to visualize the vector field from the direction of animation. In addition, alternative visualization methods were developed that combine traditional LIC-like techniques with the method presented in this thesis.

Space-Time Visualization of Dynamics in Lagrangian Coherent Structures of Time-Dependent 2D Vector Fields (Chapter 4, [BSDW12])

Lagrangian coherent structures are a means to visualize the topology of time-dependent vector fields. In this thesis, a method is presented that allows one to visualize space-time manifolds that represent the topology of a vector field. In addition, hyperbolicity and the associated dynamics are visualized directly on the space-time manifolds. In this way, these properties are not only visualized in a qualitative, but also in a quantitative way.

The method described in this chapter was developed in collaboration with Filip Sadlo and Carsten Dachsbacher.

Magnetic Flux Topology of 2D Point Dipoles (Chapter 5, [BSW+12])

Traditional topological methods are not optimal to analyze vector fields that are based on magnetic dipoles. An alternative topological construct is presented in this thesis that allows one to visualize the existence and magnitude of magnetic flux between dipoles.

The technique which is presented in this chapter was developed in collaboration with Filip Sadlo. To demonstrate the usefulness of this technique, data sets were used that were created by our application domain collaborators Rudolf Weeber, Sophia Kantorovich, and Christian Holm, who also helped to analyze the resulting images of our visualization software.

Continuous Scatterplots (Chapters 6, 7, 8, [BW08b], [BW09])

Scatterplots are a useful tool to analyze multi-attribute fields. Traditional scatterplots, however, are not able to make use of implicit data that is available in scientific data sets through means of interpolation or reconstruction. A mathematical basis for continuous scatterplots is presented in this part of the thesis to overcome this issue. The following chapters improve the original idea of continuous scatterplots to increase their flexibility with respect to interpolation or reconstruction schemes and hardware acceleration as well as algorithmic alternatives are presented that reduce the time required to compute a continuous scatterplot.

Chapter 8, which is based on our publication [HBW11], computes continuous scatterplots in an iterative way. This paper also demonstrates that this idea can be used to create continuous parallel coordinates as well. However, this thesis focuses on the computation of continuous scatterplots. This paper was first-authored by Julian Heinrich, who was supported by me in the part about continuous scatterplots.