Inaugural-Dissertation

zur Erlangung der Doktorwürde der

Gesamtfakultät für Mathematik,
Ingenieur- und Naturwissenschaften

der

Ruprecht-Karls-Universität
Heidelberg

vorgelegt von

Antoni Sagristà Sellés, M.Sc.

aus Sant Pol de Mar, Spanien

Tag der mündlichen Prüfung: _____

Visualization of Astrometric and Astrophysical Data

by

Antoni Sagristà Sellés

Advisors: Prof. Dr. Filip Sadlo
Prof. Dr. Stefan Jordan

Abstract

Astronomy is as old as mankind. Its progress and evolution in history has been parallel to the very process of human development. The advent of modern physical sciences caused an exponential progress in astronomy and, by extension, astrophysics, which has seen amazing development over the twentieth century, and especially in the first decades of the new millennium. In this work we firmly set one foot in astrophysics and the other in scientific visualization, while we develop and use the latter to respond to scientific research questions posed by the former. This thesis has been interdisciplinary since its very inception. It bases on scientific visualization and computer graphics and aims at providing and developing methods and techniques particularly designed to help analyze and understand astrophysical systems and processes. It is composed of two distinct parts:

In the first part of this thesis, we borrow from the field of vector field topology, classically concerned with the non-inertial dynamics of static and time-dependent flows, and develop and extend it to enable the study of force-induced, inertial systems, like those governing most of our universe. We then focus on the various time dimensions ingrained in vector field topology, and introduce a framework for its analysis and exploration based on novel derived aggregation fields that capture various properties of the underlying system, and present them in digestible representations in order to aid in the interpretation and analysis of the time domain. Finally, we address the very underlying structures that define complex dynamical systems, and provide a quantitative approach for their extraction and subsequent analysis.

The second part of this thesis concerns itself with the exploration and representation of very-large astrometric and astrophysical datasets. In this part, we base on computer graphics and visualization, and develop a technique to efficiently and interactively navigate through catalogs of billions of objects, introduce a method to effectively use floating-point arithmetic in the representation of the known universe without suffering from precision loss, propose a novel logarithmic function to enable limited-resolution depth buffers to function for astronomically large scenes, and present an integrated visualization of relativistic effects, including relativistic aberration due to the observer's motion and gravitational waves.

Zusammenfassung

Die Astronomie ist so alt wie die Menschheit. Ihr Fortschritt und ihre Entwicklung in der Geschichte verliefen parallel zum Prozess der menschlichen Entwicklung selbst. Das Aufkommen der modernen physikalischen Wissenschaften führte zu einem exponentiellen Fortschritt in der Astronomie und damit auch in der Astrophysik, die sich im Laufe des zwanzigsten Jahrhundert und insbesondere in den ersten Jahrzehnten des neuen Jahrtausends eine erstaunliche Entwicklung genommen hat. In dieser Arbeit setzen wir einen Fuß in die Astrophysik und den anderen in die wissenschaftliche Visualisierung, während wir letztere entwickeln und nutzen, um auf wissenschaftliche Forschungsfragen zu antworten, die von ersterer aufgeworfen werden. Diese Arbeit ist von Anfang an interdisziplinär angelegt. Sie basiert auf wissenschaftlicher Visualisierung und Computergrafik und zielt darauf ab, Methoden und Techniken bereitzustellen und zu entwickeln, die speziell dazu dienen, astrophysikalische Systeme und Prozesse zu analysieren und zu verstehen. Sie besteht aus zwei verschiedenen Teilen:

Im ersten Teil dieser Arbeit nehmen wir Anleihen aus dem Bereich der Vektorfeldtopologie, die sich klassischerweise mit der nichtinertialen Dynamik statischer und zeitabhängiger Strömungen befasst, und entwickeln und erweitern sie, um die Untersuchung kräfteinduzierter, inertialer Systeme zu ermöglichen, wie sie den größten Teil unseres Universums bestimmen. Anschließend konzentrieren wir uns auf die verschiedenen Zeitdimensionen, die in der Vektorfeldtopologie enthalten sind, und führen einen Rahmen für ihre Analyse und Erforschung ein, der auf neuartigen abgeleiteten Aggregationsfeldern Aggregationsfelder, die verschiedene Eigenschaften des zugrundeliegenden Systems erfassen und sie in verdaulichen Darstellungen, um die Interpretation und Analyse des Zeitbereichs zu erleichtern. Schließlich werden wir befassen wir uns mit den zugrundeliegenden Strukturen, die komplexe dynamische Systeme definieren, und bieten einen quantitativen Ansatz für deren Extraktion und Analyse.

Der zweite Teil dieser Arbeit befasst sich mit der Erforschung und Visualisierung sehr großer astrometrischer und astrophysikalischer Datensätze. In diesem Teil stützen wir uns auf Computergrafik und Visualisierung und entwickeln eine Technik zur effizienten und interaktiven Navigation durch Kataloge mit Milliarden von Objekten, stellen eine Methode zur effektiven Nutzung der Fließkommaarithmetik bei der Darstellung des bekannten Universums ohne Präzisionsverlust vor, führen eine neuartige logarithmische Funktion ein, um die Nutzung des Tiefenpuffers mit begrenzter Auflösung für astronomisch große Szenen zu ermöglichen, und präsentieren eine integrierte Visualisierung relativistischer Effekte, einschließlich relativistischer Aberration aufgrund von Beobachterbewegungen und Gravitationswellen.

Acknowledgements

First of all, I would like to thank my principal advisor Filip Sadlo. He has been both an inspiration and a motivation, always providing invaluable and insightful guidance and criticism without ever being too harsh, even when he had reason to. Without him, this thesis would have never been possible. It has been fun working with you Filip, thank you. I would also like to thank my secondary advisor Stefan Jordan. He was the inception point with whom the idea of starting a doctoral thesis was developed. He has taught me a lot in astrometry and astrophysics, and his joyful criticism and spirited speech have surely made me step up my game in countless occasions. Thanks, Stefan. Also, I would like to thank Andreas Just, as he initially brought me in contact with Filip and also provided much appreciated critical assessment to some of the work presented here.

I would also like to thank the people at the Visual Computing Group in Heidelberg. Even though I have not seen most of you as often as I would have liked, some good times were had, especially when the impending doom of deadlines was looming above our heads.

Finally, I would like to thank my family, Eva, Lea and Dimas, for their support and understanding over these years, and my parents for always being there.

Contents

Computer science is no more about computers than astronomy is about telescopes.

— Edsger W. Dijkstra

1 Introduction

ASTRONOMY is arguably the oldest of all the natural sciences, with its origins in calendrical, cosmological, mythological and religious practices and beliefs. The constant presence of celestial lights in the night sky has been a fundamental and pivotal factor in the development of important human activities such as science and technology across history. Humans have wondered about the celestial spectacle and have tried to decipher its mechanics for as long as there has been a human race. Hundreds of thousands of years ago, ancient human beings looked at the skies and saw a plethora of bodies of different sizes and colors moving around in puzzling, albeit consistent and understandable patterns. The cosmic backdrop back then was not much different from what we can see today, if only a little bit less polluted by light, but the interpretations and explanations of the phenomena have changed fundamentally, initially driven by raw unaided observation and superstition, and now by the methodical and tireless process of scientific inquiry and discovery.

Some seven thousand years ago, a group of nomads living in the current location of the Nubian desert in Egypt became the first known humans to record the motion of the stars using a stone circle to mark the advent of the summer solstice. They were but the first of many more. For thousands of years, societies all over the world have been building massive stone calendars and aligning them with the Sun and the stars to mark the passage of seasons. The locations and monuments are so numerous that an entire field of study, Archaeoastronomy, is devoted to the study of the role that astronomy has had in society and culture across history, and how humans have understood the phenomena in the sky and developed science and technology around it.

In due time, Astronomy would develop in parts of China, India, Egypt, Europe, America, and the Middle East, rendering the acquired profound knowledge of the stars essential to the development of early agricultural societies. As time went by, civilizations all over the planet nurtured and developed the astronomical discipline, often times in an isolated

manner, becoming increasingly dependent on it. They would accurately record the positions of stars and other celestial bodies with respect to time, name them, and group them into constellations. Thousands of years would need to pass until such rigorous and methodical observations were applied to other fields like physics or biology.

Visual aids and representations have played a significant role in the development of astronomy during its history. The early rock-circle calendars can actually be understood as proto-visualization devices. These include the alignments of stones that indicate the rising of certain celestial bodies, or the calendar circles that point in the direction of the sunrise in the summer solstice.

Nowadays visualization is everywhere. It is, almost by definition, an inter-disciplinary enterprise used to help analyze and understand many phenomena in a multitude of fields. Today, we understand that stars are not small lights positioned on a celestial dome above us moving around in two-dimensional patterns, but physical objects subject to the same laws and processes that are under effect here on the surface of Earth. We understand that the movement of stars, galaxies, and even the fate of the universe itself is predominantly determined by one of the four fundamental forces of nature, the force of gravity.

This thesis is concerned with the topic of developing effective methods to help understand and analyze astrophysical processes and systems. The chaotic motion present in gravitational systems can be analyzed and understood with well-established topology-based flow visualization techniques. Such techniques typically deal with massless particles, leaving systems like most of those found in astronomy out. Currently, this is a flagrant gap in the visualization literature that we aim to mend. Additionally, astronomical systems have a strong dependence on the time dimension, not only in terms of when they happen, but also for how long do they last. Current flow visualization methods are typically ad-hoc in this respect, providing little support to visualize the time components. We also address this in the present work. Finally, direct visualizations of astronomical concepts such as star fields are becoming more and more relevant as new data from space missions such as Gaia become available. In these exciting times, Gaia crafts the largest multi-dimensional star catalog ever known to man, with star numbers in the billions. Representing, visualizing and interactively exploring such a dataset in an accurate and realistic manner is no trivial task, especially if we are to take into account not only the two- or three-dimensional positions of stars, but also their brightnesses, colors, and even motions across the sky.

Therefore, the first part of this thesis is dedicated to developing a visualization framework suitable for studying and analyzing inertial systems using established topology-based flow visualization techniques, and providing a toolset for analyzing the time component in such systems. The second part of this thesis is concerned with the realistic

representation and interactive exploration of very large astrometric star catalogs like the ones curated by the Gaia mission with its successive data releases.

1.1 Structure of this thesis

This thesis is logically structured into two distinct parts, wrapped between an introduction and a closure or conclusion. The introduction chapter, which you are reading right now, proceeds with a listing of the contributions of this thesis.

The first part comprises Chapters 2–5, and develops a methodology to apply classical topology and flow visualization techniques to the astrophysical world.

- Chapter 2 contains an introduction to the field of flow visualization, including steady and unsteady flows, vector fields, vector field topology, dynamical systems, Lagrangian coherent structures (LCS) and the finite-time Lyapunov exponent (FTLE).
- Chapter 3 builds upon the classical topology-based flow visualization field and develops a method to extend the traditional FTLE formulation to the inertial dynamics most commonly found in astrophysical systems.
- Chapter 4 establishes a visual analysis framework to aid in the study of the temporal dimensions of dynamical systems by developing a set of aggregation functions in a novel aggregation space.
- Chapter 5 presents an attempt to quantify, formalize and understand the many parameters involved in the extraction process of the features known as Lagrangian coherent structures, which separate regions with distinct dynamics. LCS are key to the successful use of topology to understand and analyze dynamical systems.

The second part contains Chapters 6 and 7, and presents large-scale astrometric and astrophysical representation and visualization techniques in the broader context of the Gaia mission and its data releases.

- Chapter 6 provides an introduction to computer graphics, rendering and acceleration strategies like culling and level-of-detail structures.
- Chapter 7 introduces Gaia Sky, a project that enables the interactive visualization of the largest star map ever created, containing close to two billion objects and provided by the Gaia data releases.

Finally, the conclusion chapter wraps everything up, provides a summary of the main contributions and a discussion, and ponders about the possible future work.

In this thesis, important terms are typeset in italics when they are first introduced. They are then described and cross-referenced in the index, at the end of the document. Acronyms are also defined when they are first introduced. A list of acronyms with their meanings is also provided at the end.

1.2 Contributions

Research on the field of vector field topology is well-established and ongoing. The main aim of this thesis is the development of methods to provide a better understanding of astrophysical processes and systems, and to enable the interactive exploration of large astrometric catalogs. The main contributions of this work are outlined and summarized in the following paragraphs:

Topology of Inertial Dynamics (Chapter 3, [Sag+17]). Traditional vector field topology is mainly concerned with the dynamics of massless particles and has a close focus on velocity. This approach leaves out the inertial, force-induced dynamics that dominate large parts of the universe. Therefore, we present a method that adapts traditional vector field visualization to inertial dynamics by means of a decomposition of the FTLE into the phase-space contributions of position spread and velocity spread, introduce novel techniques to aid in the visualization and interpretation of the resulting high-dimensional space, and develop a full framework to aid in the analysis of such complex systems.

Visual Analysis of the FTLE (Chapter 4, [SJS20]). The above approach deals with a constrained adaptation of FTLE to the inertial case, but it omits the analysis of the various time dimensions in a global, integrated way. Thus, we introduce a method that enables the detailed analysis and exploration of dynamical systems with respect to their initial and transport times. We do so by presenting a set of aggregation functions that measure and capture essential properties of the system and present them in a visual way. These aggregation functions include basic statistical properties, height ridge quantities, aliasing information and a measure of domain connectedness.

Robust Extraction of LCS (Chapter 5). LCS visualization is an essential part of vector field topology, as it constraints and separates the domain into regions with distinct dynamics, providing insight. Typically, LCS are extracted by means of height ridges in the FTLE, but this method is often unreliable and requires of much tweaking to produce satisfactory results. Therefore, in order to complement the previous works, we present a

method to the extraction of LCS with quantitative guarantees, aimed at simplifying the process of determining these structures in a dynamical system.

Gaia Sky (Chapter 7, [Sag+19]). While the previous contributions have offered novel methods and techniques with the aim of enabling the analysis and the extraction of concealed information in astrophysical systems, in this last part we adopt a more direct approach. Here, we look at the direct representation and interactive exploration of very large astrometric and astrophysical datasets, presenting a system capable of representing billions of objects reliably and interactively. To that effect, we introduce a new level-of-detail structure tailored to stars, and we present approaches to eliminate floating-point issues derived from the representation of astronomically large scenes and to improve the default sampling scheme of the depth buffer. We wrap it up by introducing an integrated representation of relativistic aberration and gravitational waves.

1.3 Notation

All mathematical symbols and equations in this document are introduced when they first appear. Relevant concepts are introduced and typeset in italics the first time they appear, and are cross-referenced to the rest of the document in the Index section at the end. All acronyms are also introduced when they first appear. The Acronyms section, at the end of the document, contains a list of all acronyms with their meaning.

1.4 Publications

Some parts of this thesis are based on the following peer-reviewed papers previously published in scientific journals:

A. Sagristà, S. Jordan, A. Just, F. Dias, L. G. Nonato, and F. Sadlo. "Topological Analysis of Inertial Dynamics". *IEEE Transactions on Visualization and Computer Graphics* 23:1, 2017, pp. 950–959

A. Sagristà, S. Jordan, T. Müller, and F. Sadlo. "Gaia Sky: Navigating the Gaia Catalog". *IEEE Transactions on Visualization and Computer Graphics* 25:1, 2019, pp. 1070–1079

A. Sagristà, S. Jordan, and F. Sadlo. "Visual Analysis of the Finite-Time Lyapunov Exponent". *Computer Graphics Forum* 39:3, 2020, pp. 331–342

Part I

Topological Analysis of Astrophysical Systems

National boundaries are not evident when we view the Earth from space. Fanatical ethnic or religious or national chauvinisms are a little difficult to maintain when we see our planet as a fragile blue crescent fading to become an inconspicuous point of light against the bastion of the stars.

— Carl Sagan

2 Flow Visualization Fundamentals

Flow visualization is the discipline of analyzing *vector field* data by means of visual presentation, exploration, and analysis. It attempts to make flow patterns visible with the aim of visually acquiring quantitative and qualitative information on the underlying properties of the field.

In this chapter, we provide the basic notions necessary to follow the rest of the first part of this thesis. We start with an overview on flow visualization and vector fields in Sections 2.1–2.4, and later cover in detail feature-based visualization methods, with steady and unsteady vector field topology in Section 2.5.

2.1 Vector Fields

Vector fields and their analysis are essential for many fields of science and engineering. They give rise to families of transformations of space called flows, and are typically used to represent the velocities of moving fluids or the evolution of forces in space. Vector fields can be pictured as a collection of arrows, each with its own direction and magnitude, embedded in space. We can distinguish two types of vector fields: *time-independent* vector fields, also referred to as *steady flows*, and *time-dependent* vector fields, also referred to as *unsteady flows*.

A vector field is a mapping $f : \mathbb{R}^n \mapsto \mathbb{R}^m$ that assigns each point $\mathbf{x} = (x_1, \ldots, x_n)^\top$ of a domain $\Omega \subseteq \mathbb{R}^n$ a vector in $\mathbb{R}^m$. Here, we focus on the case where both the space and the vectors are in Ω, i.e., they have the same dimensionality ($n = m$). Therefore, we can define a vector field as

$$\mathbf{u}(\mathbf{x}) = \mathbf{u}(x_1, \ldots, x_n) : \Omega \to \Omega. \tag{2.1}$$

We can make it explicit and assign a vector $\mathbf{u} = u_i = (u_1, \ldots, u_n)^\top$ to each position $\mathbf{x}$,

$$\mathbf{u}(\mathbf{x}) = \begin{pmatrix} u_1(x_1, \ldots, x_n) \\ \vdots \\ u_n(x_1, \ldots, x_n) \end{pmatrix}. \tag{2.2}$$

Steady flows are usually represented by instantaneous, time-independent vector fields. They are also called static vector fields. When all the time derivatives of a flow field vanish, then, the flow field is considered a steady flow.

An unsteady flow, on the other hand, does change over time, and thus is represented by a mapping that assigns a vector to each point of the domain and to each time. This kind of flow is usually given as a time-dependent vector field,

$$\mathbf{u}(\mathbf{x}, t) = \mathbf{u}(x_1, \ldots, x_n, t) : \Omega \times T \to \Omega. \tag{2.3}$$

In mathematics, vector fields and differential equations may be considered two faces of the same thing. Specifically, time-independent vector fields are the same thing as autonomous ordinary differential equations (ODE), which do not explicitly depend on time. The location of a *massless particle* $\mathbf{x}(\tau)$, seeded in a steady flow at $\mathbf{x}(t_0) = \mathbf{x}_0$ and observed after a certain integration time τ is the solution of the autonomous ODE

$$\dot{\mathbf{x}}(\tau) = \mathbf{u}(\mathbf{x}(\tau)). \tag{2.4}$$

By contrast, the location of that same particle in an unsteady flow is given by

$$\dot{\mathbf{x}}(t) = \mathbf{u}(\mathbf{x}(t), t), \tag{2.5}$$

which, together with an initial condition $\mathbf{x}(t_0) = \mathbf{x}_0$, becomes an *initial value problem* (IVP) whose solution is the integral curve

$$\mathbf{x}(t) = \mathbf{x}_0 + \int_{t_0}^{t} \mathbf{u}(\mathbf{x}(\tau), \tau)\, \mathrm{d}\tau. \tag{2.6}$$

As we will see shortly, this curve is called trajectory or pathline, and describes the path of a massless particle released in the vector field at position $\mathbf{x}_0$ and time t_0.

2.1.1 Flow Map

We can use the concept of pathline to introduce the *flow map*, represented by $\boldsymbol{\phi}_{t_0}^{T}(\mathbf{x})$. The flow map represents the location of particles seeded at position $\mathbf{x}$ and time t_0 after their integration in the vector field $\mathbf{u}$ over an advection time T. In other words, it is a mapping from the seed positions $\mathbf{x}$ of particles released at time t_0 to their final positions at the end of their respective pathlines after advection for time T. The flow map is defined as:

$$\boldsymbol{\phi}_{t_0}^{T}(\mathbf{x}) = \mathbf{x} + \int_{t_0}^{t_0+T} \mathbf{x}(\tau)\ \mathrm{d}\tau. \tag{2.7}$$

2.2 Flow Visualization Methods

There are essentially three different types of flow visualization method families. They are direct methods, geometry-based methods and feature-based methods. They are outlined below and developed further in the forthcoming sections.

Direct methods. Direct methods (Section 2.3) are characterized by the representation of a set of primitives that directly encode properties and features of the underlying field. Glyph and quiver plots (Figure 2.1) are examples of this. They tend to suffer from occlusion problems, especially with three and higher dimensions.

Geometry-based methods. Geometry-based methods represent the flow using geometric primitives such as lines and surfaces. Integral curves (Section 2.4) are a prominent example of geometry-based methods.

Feature-based methods. These are based on the identification of prominent or conspicuous parts or characteristics of the field. Among the most prominent feature-based methods in flow visualization are the so-called topology-based methods (Section 2.5). These methods are based on the determination of the topological features of the field, which describe the flow with respect to an ensemble of locations and areas with coherent behavior. The subsequent chapters in this first part are mostly concerned with topology-based methods.

2.3 Direct Visualization

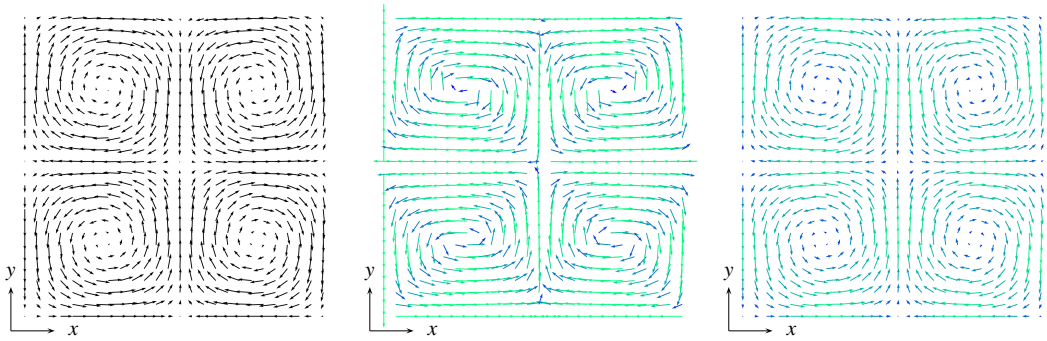
Vector fields can be visualized using direct techniques like glyphs, which are markers used to visually represent pieces of data, where the characteristics of a graphical entity

directly encode attributes of the data record. In glyph-based visualization, the dataset is typically presented as a collection of glyphs. Glyphs are a ubiquitous visualization tool found in a large variety of applications. They are intuitive, easy to implement, and convey relatively little uncertainty compared to other techniques. Sometimes also referred to as icons, glyphs are geometric primitives that can be used to visualize different data quantities by a set of visual properties including their size, color, shape, orientation, etc. The widespread consensus for a long time was that just knowing the basic properties of glyph-based visualization would be enough for its successful application and usage, but it has been shown that only well-designed glyphs are actually useful [CM84]. An effective glyph visualization should choose and combine different visual properties. General guidelines for successful glyph visualization have been proposed by Ward [War08] and Lie et al. [LKH09]. In the domain of flow visualization, vector fields can be represented with *quiver plots* (also known as hedgehog plots), where the vectors are expressed as glyph arrows located at their positions in the field, encoding orientation and magnitude. Figure 2.1 provides an example of a glyph-based visualization for a simple vector field and different quantities represented by different glyph properties. Even though the production and interpretation of these visualizations is straightforward, for our purposes, quiver plots tend to be unpractical and cumbersome, and provide little insight into the dynamics of the flow, especially in 3D and higher dimensional domains. Some of their issues are:

1. They require animation to represent time dependency.
2. It is often unclear whether the arrows represent vector values at their start, middle or endpoint. This needs to be specified as additional context.
3. Arrow length can represent the vector magnitude, but scaling is usually unsatisfactory. Too large scaling results in occlusion, too small scaling hides direction, and using a fixed length omits vector magnitude information.

2.4 Integral Curves

Direct methods are straightforward to produce and interpret, but are limited in application. An alternative common approach is the use of integral geometry, which is based on the trajectories of the particles themselves. The integration of such massless particles in vector fields leads to different types of curves, which are globally referred to as integral curves, or field lines. When the values of a vector field are interpreted as velocity, four types of curves are defined, namely *streamlines*, *pathlines*, *streaklines*, and *timelines*.



(a) (b) (c)

Figure 2.1: Example of an unsteady flow represented by arrow glyphs with different properties. In (a) the direction of the vector field at each location is represented by the orientation of the glyphs, and the magnitude is mapped to the length of the arrows. In this representation color is not used. In (b) the direction of the vectors in the field is also represented by the orientation and their magnitude is color-mapped. In this representation all arrows have the same length, leading to a cluttered visualization. Finally, (c) represents direction with the orientation and magnitude is mapped to both arrow length and color. In general, glyph representations like these have several additional problems like being able to only represent an isolated time snapshot or the difficulty of choosing the right scaling, which correctly balances occlusion and direction representation.

2.4.1 Streamlines

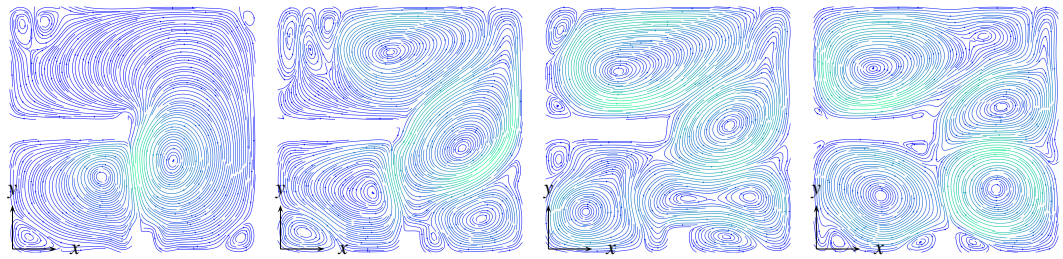
Streamlines (Figure 2.3a) represent contours of the field, showing the motion of the whole field at one instant of time. In time-dependent vector fields, instantaneous streamlines for a fixed time T are defined as

$$\mathbf{u}_T(\mathbf{x}) = \mathbf{u}(\mathbf{x}, T). \tag{2.8}$$

Streamlines are then the solution to the corresponding integral curve

$$\mathbf{x}(t) = \mathbf{x}_0 + \int_{t_0}^{t} \mathbf{u}(\mathbf{x}(\tau), T)\, \mathrm{d}\tau. \tag{2.9}$$

Streamlines are always tangent to the vectors of the field, and show the direction in which non-inertial particles will travel when released at any position and point in time. They represent the whole motion of the field at a single time snapshot, so they would still need animations to represent the time component. They are extracted in unsteady flows by integrating a particle trajectory at a constant time T. Streamlines are solutions to Equation 2.9, always exist and cannot cross each other, since they are unique. In non-steady flows streamlines are of lesser importance, especially when compared to pathlines,



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

Figure 2.2: Example of an unsteady flow represented with streamlines at different times. The time slices are shown at $t = 0.05$ s (a), $t = 0.1$ s (b), $t = 0.15$ s (c) and $t = 0.2$ s (d). The magnitude of the vector field at each location is mapped to color. As we move in time from (a) to (d), we see different regions formed by concentric lines appear and sometimes break up and evolve into multiple ones. This obviously provides some insight, but streamline plots can still only show a static vector field, or a single time slice of a time-dependent one. More advanced techniques can be used, like Line Integral Convolution, but they still represent only a static picture.

as they do not describe true particle trajectories. An example of streamline-based vector field visualization is shown in Figure 2.2.

2.4.2 Pathlines

Also referred to as trajectories, pathlines (Figure 2.3b) describe the path of massless particles released in a flow at time t_0 and at position $\mathbf{x}_0$. A pathline is the solution of Equation 2.6. Pathline-based visualization techniques applied to unsteady flows usually produce cluttered plots where the pathlines intersect and cross each other. They must be used with care in static visualization, but they are very well suited for dynamic visualization. In unsteady flows, pathlines are equal to the particles' trajectories as they move through the time-varying vector field. In a physical experiment, pathlines can be obtained by capturing long exposures of tracer particles as they advect with the flow.

2.4.3 Streaklines

Streaklines (Figure 2.3c) are obtained by continually releasing particles at a fixed location and taking a snapshot of the generated pattern at a fixed later time. The advected trail of particles forms a curve that visualizes the change of the flow at a certain location over time, also called temporal coherence.

Streaklines starting at a position $\mathbf{y}$ and captured at time t_n are generated using the following recipe:

1. For each time sample $t_0, \ldots, t_n$ solve the IVP

$$\dot{\mathbf{x}}(t) = \mathbf{u}(\mathbf{x}_i(t), t), \quad \mathbf{x}_i(t_i) = \mathbf{y}. \tag{2.10}$$

2. Extract the point $\mathbf{x}_i(t_n)$ from each integral curve $\mathbf{x}_i(t)$.
3. Connect the points to form the streakline.

Note that the time interval between the release of the particles must often be refined in an adaptive fashion to avoid consecutive particles diverging too much.

2.4.4 Timelines

Timelines (Figure 2.3d) are obtained by injecting particles densely on a seed curve $\mathbf{c}(u)$ at distinct times and taking a snapshot of the generated pattern at a fixed later time. They show the propagation of a line of massless particles in time, as well as the spatial coherence between particles seeded at the same time along a curve. In physical experiments they can be produced by placing a wire into a fluid and applying current pulses. Each pulse generates small bubbles which are advected with the flow, producing the timeline. They are typically defined as

$$\mathbf{t}(u) = \boldsymbol{\phi}_t^{\tau}(\mathbf{c}(u)). \tag{2.11}$$

Similarly to streaklines, timelines starting at seeds $\mathbf{y}_i$ on the curve and captured at time t_j can be generated as follows:

1. For each point sample $\mathbf{y}_0, \ldots, \mathbf{y}_n$ on the curve, solve the IVP

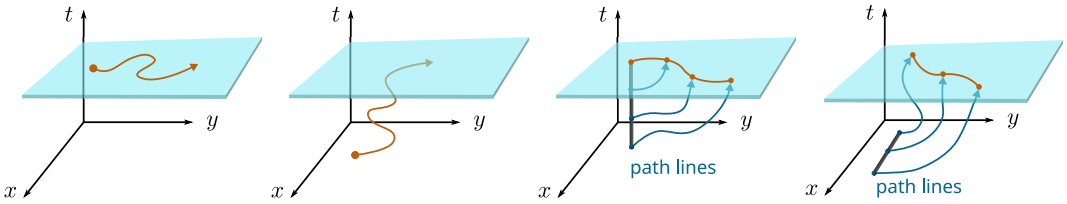
$$\dot{\mathbf{x}}(t) = \mathbf{u}(\mathbf{x}_i(t), t), \quad \mathbf{x}_i(t_0) = \mathbf{y}_i. \tag{2.12}$$

2. Extract the point $\mathbf{x}_i(t_j)$ from each integral curve $\mathbf{x}_i(t)$.
3. Finally, connect the points to form the timeline.

The result of this algorithm is a timeline for time t_j. Similarly to streaklines, the spatial interval in timelines often requires adaptive refinement to prevent close-by particles diverging too much.

2.4.5 Discussion

In steady flows, streamlines, pathlines, and streaklines are identical. Streaklines and timelines are computationally more expensive than only solving a single IVP, since they are based on releasing more than one particle and computing their pathlines. Computing



(a) streamline (b) pathline (c) streakline (d) timeline

Figure 2.3: Space-time illustrations of integral curves. In orange, the four main types: (a) streamlines, (b) pathlines, (c) streaklines, and (d) timelines. In the case of streaklines and timelines, their inception pathlines are also shown in blue.

streamlines, pathlines, and their derived streak and timelines, has several issues typically related to the appropriate choice of the integration method and step size, and the choice of the global and local point location method used. In this work, we use the *fourth-order Runge-Kutta* (RK4) integration scheme to compute all integral curves.

Similarly to the direct visualization methods, visualizations using integral curves also often suffer from occlusion and cluttering problems, especially in three dimensions, unless they are used locally and with measure to illustrate particular localized behaviors or properties. Seeding integral curves at the nodes in *regular grids* is typically not a good idea. Some works have been done on the placement of such lines, like the original work of Turk and Banks exploring the placement of streamlines using an adaptive image-space method based on maintaining line density to achieve equal spacing [TB96]. Additional occlusion mitigation and global placement approaches have been proposed for two-dimensional [JL97] and three-dimensional [FG98] flow visualization, also utilizing flow features in the dataset [VKP00] or context-based methods [Sch+07].

2.5 Vector Field Topology

Before Helman and Hesselink introduced vector field topology to the visualization community [HH89; HH91], the visualization of complex flows was accomplished with experimental techniques, e.g., seeding smoke or tracing patterns on the surface of a flow. These approaches are certainly not well suited for modern times, with datasets ever increasing in complexity, dimensionality and size. *Vector field topology* (VFT) aims to help in the endeavor of analyzing and understanding the transport in flows, and is often used to simplify the representation of a vector field, while enabling deeper insight into the very structure of the field. VFT addresses the challenge of representing vector fields in a comprehensive manner through the identification of topological features, which are elements that reveal the most essential structure of the vector field. Typically, this is achieved via

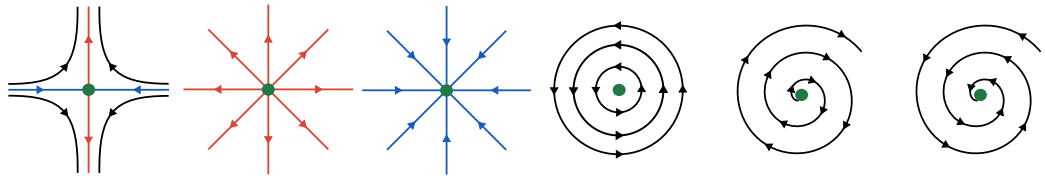
the detection and classification of *critical points*, which are isolated zeros of the vector field, and manifolds seeded in direction of the *eigenvectors* of the gradient matrix at these points, extracted by streamline or stream surface integration.

Among these, the codimension-one manifolds (surfaces in three-dimensional flows, or curves in two-dimensional flows) that separate the flow into regions that behave differently. Particularly, we distinguish two specially interesting scenarios where these structures are involved. First, the ones where the flow close to a structure attaches or separates from it, leading to attachment and separation lines. Second, the structures where streamlines that start arbitrarily close to each other at either side of the structure actually end in very different areas. These second type of structures is of paramount importance in VFT and, together with the analysis of critical points, conforms its basis. Asimov [Asi93] offers a very good starting point to the literature on steady vector field topology, and the next sections provide a short summary thereof.

2.5.1 Critical Points

Critical points are positions in the flow field domain where the velocity vanishes and the *Jacobian* has full rank (non-zero *eigenvalues* and non-zero determinant). The vectors at critical points are zero. Critical points can be characterized according to the behavior of the streamlines in the vicinity of the point. When a streamline leads into one of this points, the streamline converges, as the tangent is not defined (the magnitudes of the vectors around the point approach zero). In this case, a streamline that starts at a particular position $\mathbf{x}_0$ with $\mathbf{u}(\mathbf{x}_0) = \mathbf{0}$ degenerates to a point. This kind of point is called constant orbit or stationary point. These points are also called isolated if they are not adjacent to others, i.e., the vector field in the surrounding area takes all possible directions.

Critical points can be classified to a first-order approximation according to the real and imaginary parts of the eigenvalues of the Jacobian matrix in the neighborhood of the position of the critical point. Visualizations of *higher-order* critical points (i.e., points where the determinant of the Jacobian matrix at the point is zero) are addressed by Scheuermann et al. [Sch+97] and Weinkauf et al. [Wei+05]. In a first classification, hyperbolic critical points are those whose eigenvalues of the velocity gradient have nonzero real parts. Hyperbolic critical points imply local stability, meaning that they are stable against small perturbations of the vector field. In this case, perturbations do neither change the topology of the vector field nor the general trend and direction of the streamlines. In a more thorough classification based on the real and imaginary parts of the eigenvalues of the Jacobian (Table 2.1), we can define, for 2D vector fields, the six major types of critical points as saddles (positive and negative real parts, imaginary parts equal to zero), repelling



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

Figure 2.4: Illustrations of the types of critical points: saddle (a), source or repelling node (b), sink or attracting node (c), center (d), repelling focus (e) and attracting focus (f). Colored vectors represent eigenvectors corresponding to negative (blue) and positive (red) eigenvalues.

nodes (positive real parts, imaginary parts equal to zero), attracting nodes (negative real parts, imaginary parts equal to zero), centers (zero real parts, nonzero opposite signs in imaginary parts), repelling focuses, also known as sources (positive real parts, non-zero opposite signs in imaginary parts) and attracting focuses, also known as sinks (negative real parts, non-zero opposite signs in imaginary parts). Illustrations of the various types of critical points are provided in Figure 2.4 and the eigenvalue classification is condensed in Table 2.1. Figure 2.5 showcases a vector field with four centers and a saddle-type critical point. Qualitatively speaking, the imaginary parts of the eigenvalues represent circulating flow patterns, and the real parts represent the attracting and repelling behavior of the flow. For instance, a positive real part indicates a repelling node (source), while a negative real part indicates an attractive node (sink). Complex eigenvalues indicate rotation around the point. There are other types of points, like *attachment* and *detachment nodes* (see gray circles in Figure 2.5), related to the attachment and separation lines mentioned above, where streamlines end at walls where the velocity is zero, or boundary incoming and outgoing nodes, where streamlines leave the domain of available data of the vector field, but these have a comparatively minor importance (i.e., boundaries are somewhat artificial and often depend on the arbitrary region of study) and are not covered here.

Out of all of the critical point types, saddle points are special in the sense that only four streamlines converge at the point itself. At the saddle point, the streamlines are tangent to the eigenvectors of the Jacobian, with one eigenvector direction converging to the critical point and one converging in reverse time. These sets of streamlines are codimension-one (i.e. lines in 2D fields, two dimensional surfaces in 3D) stable and unstable manifolds [Asi93]. The stable manifolds converge to the critical point or a periodic orbit in forward time and separate regions of the domain where the dynamics of the flow are distinct. When we release two particles at either side of such a stable manifold and integrate them in forward time, they diverge from each other. In contrast, unstable manifolds converge to the critical point or a periodic orbit in backward time. If we release particles

at either side of an unstable manifold, they diverge from each other when integrated in backward (negative) time.

2.5.2 Separatrices

As we have seen in the previous section, the eigenvector orientations define the type of critical points. For example, saddle-type critical points have streamlines converging toward the critical point along one of the eigenvectors from both sides (magenta lines in Figure 2.5), and diverging from the critical point along the other eigenvector to both sides (green lines in Figure 2.5). In the neighborhood of a sink-type critical point, all streamlines converge to the critical point along all directions, and in a source-type critical point the streamlines diverge along all directions. Saddle-type critical points, then, are special because they are related to distinguished streamlines.

In two-dimensional vector fields, streamlines that converge to a saddle-type critical point in forward time and converge to it in backward time are known as the *separatrices* of the vector field, because they separate regions of qualitatively distinct behavior in the vector field. These separatrices are of much importance in vector field topology, as, together with the set of all critical points, make up the *topological skeleton* of the field.

In two-dimensional and three-dimensional vector fields, the set of all streamlines that converge to a critical point or periodic orbit in forward or backward time is called an invariant manifold.

Table 2.1: Classification of critical points by the real and imaginary parts of the eigenvalues of the Jacobian matrix around the point. $\mathfrak{R}_1$ and $\mathfrak{R}_2$ indicate the real parts of the eigenvalues, and $\mathfrak{I}_1$ and $\mathfrak{I}_2$ indicate their imaginary parts. When the imagniary parts are nonzero, we have complex-conjugate eigenvalues.

Type	$\mathfrak{R}_1$	$\mathfrak{R}_2$	$\mathfrak{I}_1$	$\mathfrak{I}_2$
saddle	< 0	> 0	0	0
repelling focus	> 0	> 0	$\neq 0$	$\neq 0$
attracting focus	< 0	< 0	$\neq 0$	$\neq 0$
center	0	0	$\neq 0$	$\neq 0$
source	> 0	> 0	0	0
sink	< 0	< 0	0	0

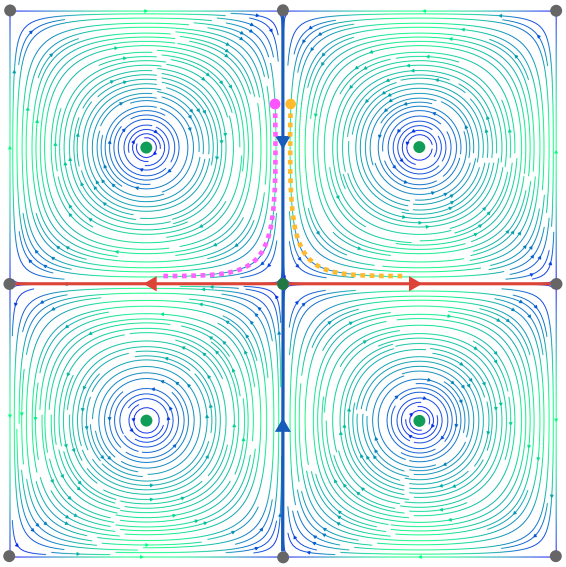


Figure 2.5: Streamline representation of the same vector field displayed in Figure 2.1 with a saddle-type critical point (light green circle), four centers (dark green circles) and eight boundary attachment nodes (gray circles). Stable manifolds (blue lines) converge towards the saddle-type critical points, while unstable manifolds (red lines) diverge from it. Two particles in magenta and yellow released at either side of the stable manifold diverge from each other when integrated in forward time. The stable manifold separates trajectories of particles in forward time, providing a boundary between regions with different behavior.

2.5.3 Lagrangian Coherent Structures

Since traditional VFT builds on streamlines rather than pathlines, it has the major drawback that it is only meaningful for steady vector fields. The major reason for that is that in unsteady vector fields, the pathlines are different to streamlines, and both the critical points and the separatrices tend to move about in a way that is not consistent with time-dependent advection. VFT is then able to give only an instantaneous view on the vector field, which is only directly interpretable for stationary vector fields or isolated time snapshots. In the past, unsteady vector fields were analyzed in this fashion by using vector field topology methods in isolated time steps, but this approach proved hard to interpret and did not result in useful insight into the real behavior of the flow along the temporal dimension. Shadden et al. [SLM05] showed that the separatrices extracted from time snapshots can actually diverge a lot from where the actual flow separation resides in time-dependent vector fields. The advantage of the coherent structures discussed here is that they reveal the true behavior of the flow in a physically motivated, easy to interpret man-

ner. Other approaches to the topology-related visualization of unsteady vector fields were presented by Theisel et al. [The+04] and Shi et al. [Shi+06].

Lagrangian coherent structures (LCS) are the time-dependent counterpart to separatrices. They are distinguished manifolds of coherent behavior in flows that exert a major influence on nearby trajectories over a time interval. These structures are solution trajectories that differ from their neighbors in their origin or destination, thus dividing dynamically distinct regions in the flow, revealing geometry which is usually hidden when analyzing the vector field with more direct visualization techniques. In contrast to the separatrices of VFT, they tend to be insensitive to short-term perturbations and small-scale noise due to their Lagrangian definition. LCS of unsteady flows are also unsteady, so they tend to move and deform over time. LCS, like separatrices, are attracting if infinitesimal perturbations converge to these structures in forward time and repelling if they are attracting in backward time. Therefore, these structures help in the analysis of time-dependent systems and are a good tool to understand transport. LCS have been proven along the years to be very useful in a broad range of applications. Nowadays, LCS are mainly extracted as height ridges in derived fields, like the *finite-time Lyapunov exponent* (FTLE). An excellent introduction to the field of time-dependent vector field topology is provided by Pobitzer et al. [Pob+11], and an overall survey on topology-based methods by Heine et al. [Hei+16]. Recent advances in streak-based topology are covered by Sadlo, Üffinger, and Machado et. al. [Mac+16; SW10; ÜSE13].

In the next sections, we first introduce the classical Lyapunov exponent, and later have a look at its time-dependent counterpart, the FTLE, to conclude with an overview on height ridges and their extraction mechanisms.

2.5.4 Lyapunov Exponent

Before properly introducing the FTLE, we must first quickly cover the classical *Lyapunov exponent* (LE). Its main motivation is that small differences in initial values of a dynamical system can grow into very large differences over time, which is an intrinsic property of deterministic chaos. The reason for this is the exponential growth of those small initial differences, or perturbations. The differences grow in respect to the size they have reached in the previous moment in time, so the larger they are, the faster they grow. The classical Lyapunov exponent, named after the Russian mathematician Aleksandr Mikhailovich Lyapunov, is a measure of the rate of growth of these generic perturbations. This exponent indicates the speed with which two initially close dynamics diverge (if the LE is positive) or converge (if it is negative) in phase space. Generally speaking, phase space is the space containing all possible states of a physical system. In this thesis, we use phase space

as the space containing the positions and velocities. If we have the perturbation $\boldsymbol{\delta}_{t_0}(\mathbf{x})$, which is applied at position $\mathbf{x}$ at time t_0, the LE represents the limit of its growth for $|T| \to \infty$ and $\|\boldsymbol{\delta}_{t_0}(\mathbf{x})\| \to 0$, and is defined as

$$\sigma_{t_0}(\mathbf{x}) = \lim_{|T|\to\infty} \lim_{\|\boldsymbol{\delta}_{t_0}(\mathbf{x})\|\to 0} \frac{1}{|T|} \ln \frac{\|\boldsymbol{\delta}^T_{t_0}(\mathbf{x})\|}{\|\boldsymbol{\delta}_{t_0}(\mathbf{x})\|} , \tag{2.13}$$

with $\boldsymbol{\delta}^T_{t_0}(\mathbf{x})$ being the (grown) perturbation at time $t_0 + T$, i.e., the distance between the endpoints of the two trajectories (Figure 2.7a). Notice that the limit $\|\boldsymbol{\delta}_{t_0}(\mathbf{x})\| \to 0$ ensures that the two trajectories stay sufficiently close to each other, i.e., that they allow for a linearization of the vector field between them, for the entire advection time T.

The LE hence indicates how rapidly a complex system of several interdependent dynamics tends to run up to deterministic chaos. The inverse value of the LE indicates the so-called *Lyapunov time*. This is the time it takes for an initial perturbation to reach a certain magnitude, thus allowing certain conclusions about the predictability of the system. The Lyapunov time provides an estimate for determining the time period a system is expected to be predictable.

2.5.5 Finite-Time Lyapunov Exponent

The classical LE is quite useful in the study of ergodic theory for time-independent dynamical systems. However, the dynamical systems of practical importance for this work are time-dependent and only computed or measured over a finite amount of time. Due to its asymptotic nature, the original LE is not well suited for analyzing time-dependent dynamical systems, or those that are not defined over an infinite time interval, so their practical value is rather limited in these scenarios. Lately, finite-time Lyapunov exponent fields have become the most popular tool to analyze and gain insight into the behavior of coherent structures in unsteady flows [Pob+11].

The FTLE (Figure 2.6) is a scalar measure of the separation of trajectories that are seeded simultaneously at time t_0 close to each other, i.e., at distance $\boldsymbol{\delta}_{t_0}(\mathbf{x})$, and integrated for an advection duration T (Figure 2.7a). For an n-dimensional vector field, there are n Lyapunov exponents. Notice, that in general, trajectories correspond to pathlines (Section 2.4) in this context, i.e., the paths of massless particles in a time-dependent vector field $\mathbf{u}(\mathbf{x},t)$, released at their seed points and obtained by solving the respective initial value problems. In that sense, one can consider $\mathbf{x}$ being the seed position of one trajectory and $\mathbf{x} + \boldsymbol{\delta}_{t_0}(\mathbf{x})$ the seed of another. The finite-time Lyapunov exponent, in contrast

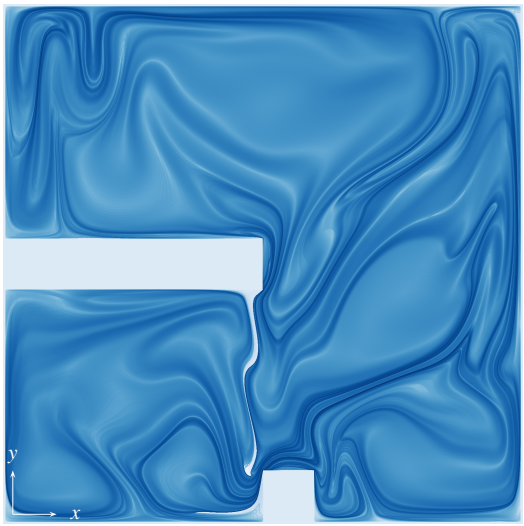


Figure 2.6: Example of a forward-time FTLE field extracted with a resolution of 1600×1600 and $t_0 = 0.09$ s and $T = 0.088$ s using the Buoyant Flow dataset. Darker regions indicate higher FTLE values, while lighter areas are mapped to low FTLE values. The sharper dark-blue lines are candidates to local maxima, and represent repelling LCS.

to the regular LE in Equation 2.13, measures this separation only for a predefined finite advection time T:

$$\sigma_{t_0}^{T}(\mathbf{x}) = \lim_{\|\boldsymbol{\delta}_{t_0}(\mathbf{x})\| \to 0} \frac{1}{|T|} \ln \frac{\|\boldsymbol{\delta}_{t_0}^{T}(\mathbf{x})\|}{\|\boldsymbol{\delta}_{t_0}(\mathbf{x})\|} \,. \tag{2.14}$$

Note that in both formulations (LE and FTLE), $\boldsymbol{\delta}_{t_0}(\mathbf{x})$ has to be oriented such that the obtained value is maximal.

For at least three decades, the FTLE has been indeed computed by testing different orientations of $\boldsymbol{\delta}_{t_0}(\mathbf{x})$, until Haller [Hal01] provided an estimate based on the flow map. As we have already seen in Section 2.1, the flow map $\boldsymbol{\phi}_{t_0}^{T} : \mathbf{x} \mapsto \boldsymbol{\phi}_{t_0}^{T}$ is a mapping from the seed points $\mathbf{x}$ of trajectories started at time t_0 to their respective endpoints $\boldsymbol{\phi}_{t_0}^{T}$ after advection for time T. Consequently, its gradient, i.e., Jacobian, $\nabla\boldsymbol{\phi}_{t_0}^{T}$ captures the spread of this mapping, i.e., captures the spread of the endpoints, and thus the norm of this tensor provides an estimate for $\|\boldsymbol{\delta}_{t_0}^{T}(\mathbf{x})\| / \|\boldsymbol{\delta}_{t_0}(\mathbf{x})\|$, leading to the following formulation of the FTLE:

$$\sigma_{t_0}^{T}(\mathbf{x}) = \frac{1}{|T|} \ln \left\| \nabla\boldsymbol{\phi}_{t_0}^{T}(\mathbf{x}) \right\|_2 \,, \tag{2.15}$$

with $\|A\|_2$ representing the spectral norm of tensor A, i.e., the square root of the largest eigenvalue of $A^{\top}A$ for a matrix A. This formulation of the FTLE is used in the major part of the visualization literature and is used here as well.

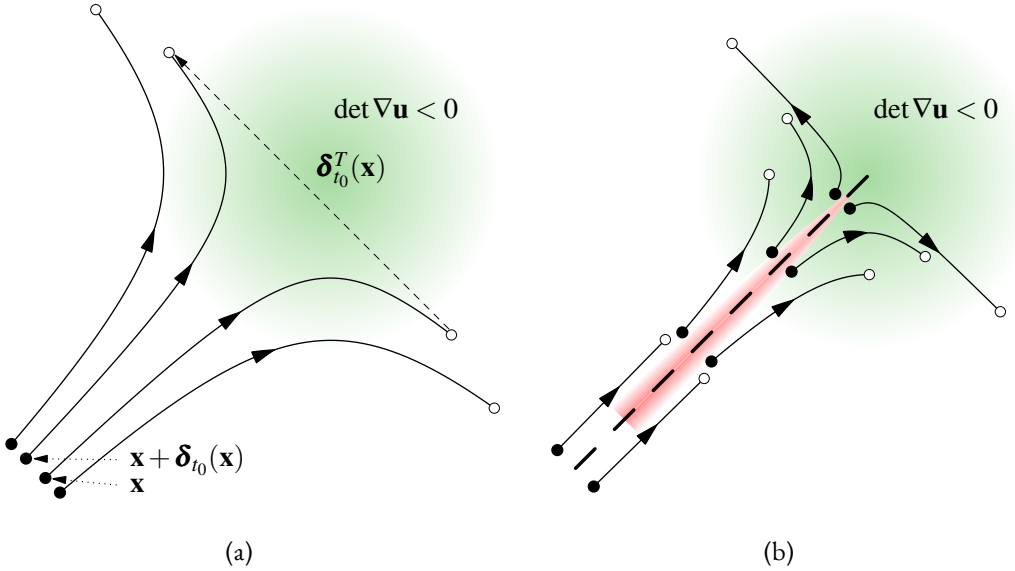


Figure 2.7: (a) Separation ($\|\boldsymbol{\delta}^T_{t_0}(\mathbf{x})\|/\|\boldsymbol{\delta}_{t_0}(\mathbf{x})\|$) of trajectories (black) by hyperbolic region ($\det \nabla \mathbf{u}(\mathbf{x},t) < 0$, green). Notice that the lifetime T_χ and the strength $|\chi|$ of the hyperbolic region have to be sufficient to cause such a separation. (b) As T increases, i.e., trajectories become longer, LCS tend to become longer too (in this case grow to bottom left, since further trajectories reach the hyperbolic region. At the same time, ridges (dashed) in the FTLE field (red, higher values by higher saturation) that were existing before (are closer to the hyperbolic region) become sharper because separation becomes stronger.

LCS visualization and analysis is nowadays predominantly based on the FTLE. Haller [Hal01] showed that LCS can be obtained by detecting local extrema in the FTLE. This was later confirmed with quantitative guarantees by Shadden et al. [SLM05]. According to Haller, attracting LCS can be obtained as ridges (approximated as height ridges, see Section 2.5.7) in the backward-time FTLE, and repelling LCS can be obtained as ridges in the forward-time FTLE. The resulting stable and unstable manifolds described in Sections 2.5.1 and 2.5.2 have their counterparts in steady, non time-dependent VFT in repelling and attracting LCS, respectively.

Four example FTLE fields captured using the Buoyant Flow dataset with the same t_0 and T, and different *resolutions* and height ridge extraction are shown in Figure 2.8.

FTLE and Inertial Dynamics. While traditional FTLE deals with the dynamics of massless, non-inertial particles in flows, in Chapter 3, we present a novel extension to the analysis of inertial systems based on topology in general, and the FTLE in particular. Similar approaches have been presented before, like those by Sapsis and Haller, Peng and Dabiri, and Günther and Theisel, all in the field of inertial particles driven by fluid flow.

Our approach, in contrast, is not restricted to flows, adopts a more general stance, and is able to accommodate systems driven by gravitational interactions and electromagnetic forces. Studying the flow-induced dynamics of inertial particles is a rather evolved topic with various applications from, e.g., meteorology [SH09], biology [SPH11], and multi-component or multi-phase flow [RRV14]. Recent works outside visualization include those by Sapsis and Haller [SPH11], and Peng and Dabiri [PD09]. Sudharsan al. [SBR16] gives a good introduction into this field. In the domain of visualization, the first work of Günther et al. [Gün+13] provides techniques and introduced new concepts for integral curves of flow-induced motion of inertial particles. Subsequently, Günther and Theisel extend the concept of vortex core lines to flow-induced motion of inertial particles [GT14]. More recently, they presented a counterpart [GT16a] to vector field topology for steady-state flow-induced motion of inertial particles, a respective concept [GT15] for time-dependent flow inspired by the FTLE with focus on the separation of flow-induced inertial trajectories due to mass, and a solution [GT16b] to the demanding source inversion problem in flow-induced transport of inertial particles.

Also recently, Garaboa-Paz and Pérez-Muñuzuri [GP15] extend FTLE for flow-induced inertial trajectories in incompressible flow to phase space, i.e., they investigate the impact of the variation of initial velocity, similar to our approach but as a whole (without separating position and velocity as we do in Chapter 3). Although all these works analyze the motion of inertial particles, they are all defined in terms of the dynamics of inertial particles induced by fluid flow. This motion is obtained by an empirical model ([Gün+13], Equation 1), defining acceleration as a function of the velocity of the particle, flow field velocity, and other accelerations. Thus, the problem of motion of flow-induced inertial particles is a special case of the problems that we address with our technique in Chapter 3, i.e., we have no specific constraints on the acceleration of inertial particles. Also, in all these previous works (except for the topology approach [GT16a] and the phase-space approach [GP15]), initial velocity is assumed constant, which does not make it necessary to treat the problem in $2n$-dimensional phase space, as we do.

FTLE Visual-Based Analysis. In Chapter 4, we put our focus on the interdependence between t_0 and T. Our approach is based on the formulation of aggregation fields in the t_0–T space, to help to analyze and explore dynamical datasets by means of the FTLE and the LCS. Probably due to the very high computational cost of the FTLE and the difficulties associated with making it interactive, this is the first work that applies *visual analytics* methods, which focus on the analytical reasoning enabled by interactive visual interfaces, to aid in the application and interpretation of FTLE-based flow analysis. We combine the FTLE and flow visualization with an integrated interactive visualization sys-

tem in Chapter 3, but this has also been attempted before, for example, in the realm fluid dynamics in general [Sch+99]. Our work here also borrows from Weiskopf's exploration of GPU-based interactive visualization techniques of scalar and vector fields [Wei07].

We also find some important work in the domain of systems research in visualization. For instance, the Visualization Toolkit [SML04], VisTrails [Bav+05], ParaView [AGL05], VisIt [Chi+12], the Insight Toolkit [JMI15], Diderot [Kin+16], VTK-m [Mor+16] or the very recent Topology Toolkit [Tie+18] are good candidates. In this field, the Topology Toolkit implements most of the techniques mentioned above and offers an open-source platform for topological analysis in visualization, while VisTrails offers a provenance infrastructure which maintains a detailed history of data and workflows in the visualization process. The latter would represent a fine companion to our system in order to ensure the persistence and reproducibility of results.

The visual analysis of time-dependent data and flow fields has also been attempted before. For instance, Aigner et al. [Aig+07] study the introduction of time as an additional dimension in visual analytics, Bürger et al. [Bür+07] integrate local feature detectors in the visual analysis of time-dependent flow simulations, Shi et al. [Shi+07] present an approach to visually analyze time-dependent flow fields through the behavior of pathlines, and Doleisch, Hauser and co-authors [DGH03; Dol+04] study the visual analysis of complex time-dependent flow simulations and real data. In this context, the technique introduced in Chapter 4 provides an interactive analysis of trajectories, and introduces the concept of FTLE aggregation fields, which attempt to capture global trends and features in the seeding and advection time dimensions.

2.5.6 Finite-Size Lyapunov Exponent

An extension to the concept of FTLE was proposed by Aurell et al. [Aur+97] where instead of measuring the growth rate of finite-time perturbations, a finite-size approach is adopted. The *finite-size Lyapunov exponent* (FSLE) limits the maximum separation s between pathlines and measures the time needed to reach it. It coincides with the FTLE for sufficiently small perturbations, but diverges significantly otherwise. The FSLE approach is motivated by the necessity of creating a measure that is independent of the advection time T, since different regions may require different choices of advection time. In the same fashion as Haller's formulation of the FTLE (Equation 2.15), we can formulate the FSLE $\sigma^s_{t_0}$ with a given separation factor s as:

$$\sigma^s_{t_0} = \frac{1}{|T_s|} \ln s, \tag{2.16}$$

with a minimal $|T_s|$ such that

$$\left\| \nabla \boldsymbol{\phi}_{t_0}^{T_s}(\mathbf{x}) \right\|_2 = s \tag{2.17}$$

holds true. In these equations, T_s is the time required for a small perturbation to reach a separation of s. The FSLE is undefined in the locations of the domain where a separation of s is never reached.

2.5.7 Height Ridges

Nowadays, the most prominent method for extracting LCS from a time-dependent vector field are height ridges in the FTLE field. A codimension-one ($(n-1)$-dimensional) *height ridge* in a scalar field $s(\mathbf{x})$, with $\mathbf{x} \in \Omega$, can be obtained according to Eberly [Ebe96] by extracting the zero-isocontour from a derived scalar field:

$$\nabla s(\mathbf{x}) \cdot \boldsymbol{\varepsilon}_{\min}(\mathbf{x}) = 0 \,, \tag{2.18}$$

where $\boldsymbol{\varepsilon}_{\min}(\mathbf{x})$ is the minor eigenvector of the Hessian $\nabla\nabla s(\mathbf{x})$, with the additional requirement of the minor eigenvalue $\lambda_{\min}$ of the Hessian being negative. In discrete (grid-based) settings, this isocontour extraction can be accomplished based on the marching squares algorithm [LC87], with an additional step that makes eigenvector orientation consistent [FP01]. Additionally, it is common to suppress spurious solutions (noise) by rejecting those parts of the solutions where $\lambda_{\min}$ is not sufficiently small, i.e., one requires $\lambda_{\min} < \tau_\lambda$ with a user-defined threshold $\tau_\lambda \leqslant 0$. When necessary, especially in Chapter 4, we also employ such filtering in our examples, and document our choice for τ_λ accordingly. Additionally, the work by Peikert and Sadlo [PS] explores and assesses further methods for height ridge computation and filtering.

In Section 5.1.1, we provide an introduction to *cross-flux* in the context of LCS, and show how it has been used as a measure to evaluate whether ridges are good candidates for LCS. In Section 5.1.2, we delve into more practical matters, and explore the interdependence in FTLE between the advection time, i.e., the time length we use as a basis to our analysis, and the sampling resolution, showing that higher advection times typically require higher resolutions.

As demonstrated in Figure 2.8, the main issue with height ridge extraction from FTLE fields is typically not (numerical) noise due to the involved second derivatives, but aliasing and ridges that are closer together than the support size of the discrete second-derivative operator, affecting estimation of the Hessian. We address both issues in Sections 4.2.3 and 4.2.4. Much work has been done on height ridges in the past. Garth et al., and Sadlo and Peikert present works on direct visualization of FTLE and LCS [Gar+09;

SP09]. Other approaches to ridge extraction attempt to solve the numerical noise problem, including the filtered AMR ridge extraction by Sadlo et al. [SP07], which is based on the determination of the gradient by least squares and also works for *unstructured grids* (grids with irregular connectivity), and the feature extraction method by Kindlmann et al. [Kin+18], implemented in Diderot [Kin+16], which solves the numerical noise issue by using ray casting and advanced interpolation schemes. Many applications and derivations have followed, such as the analysis of area-preserving maps [Tri+12a], tensor fields [Hla+11; Tri+12b], computational steering [Ame+11], dynamics of inertial particles [GT17], advection-diffusion flow [SKE14]. Other approaches to ridge extraction involved alternative computation methods, such as the local approach by Kasten et al. [Kas+09], cell tracking [Kuh+14], grid advection [SRP11], streak-based topology [SW10; ÜSE13], and higher-order evaluation [Üff+12]. Although some of these approaches formulate refinement criteria, these are similar to those used for adaptive refinement mentioned above, i.e., they are not quantitative with respect to LCS properties such as cross-flux. None of these approaches has, however, employed physically-quantitative measures for assessing the quality of LCS visualization, as we do in Chapter 5.

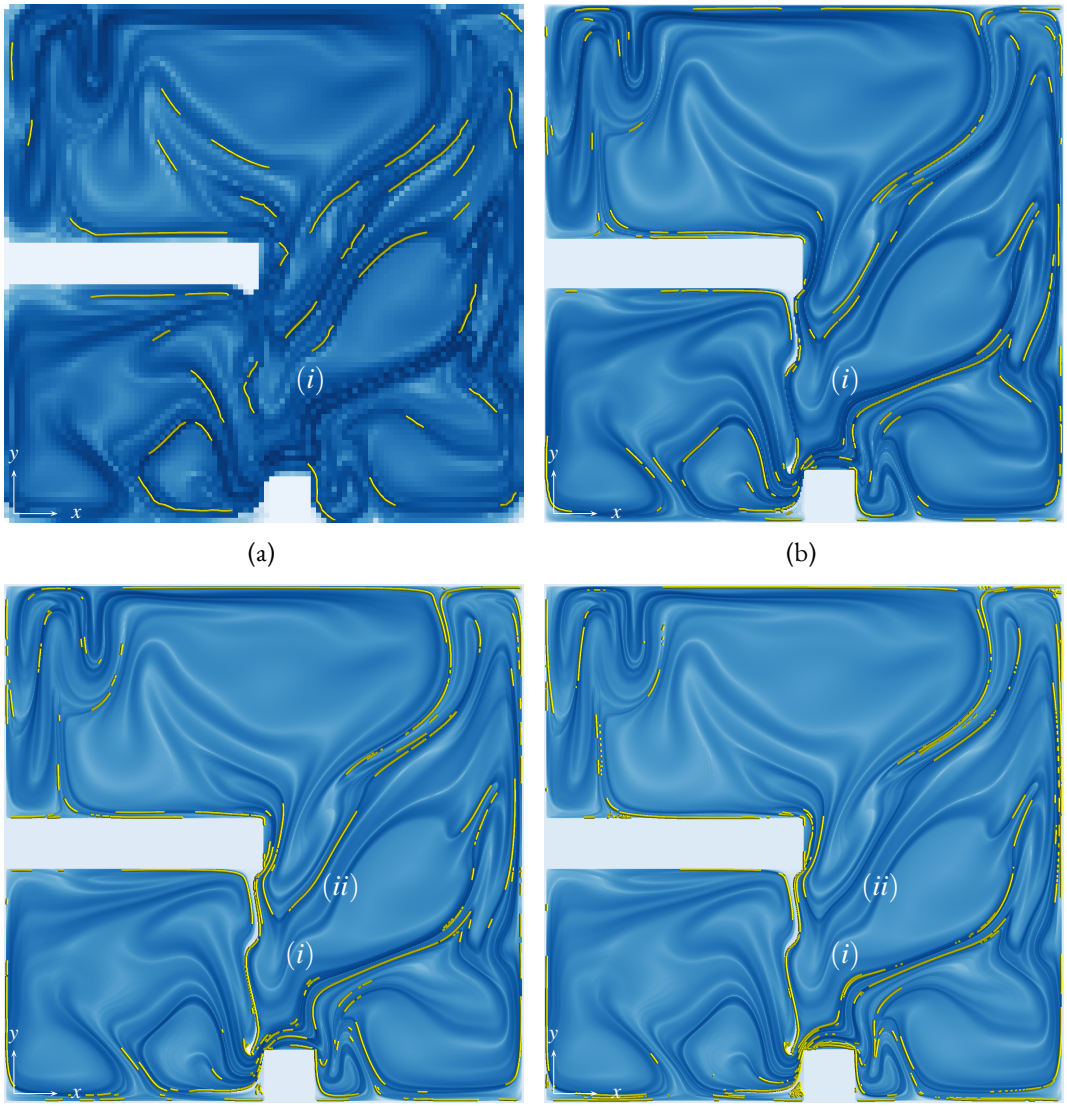


Figure 2.8: Height ridge extraction from the Buoyant Flow dataset with four different FTLE field resolutions of 100×100 (a), 400×400 (b), 800×800 (c), and 1600×1600 (d), with identical $t_0 = 0.09$ s and $T = 0.088$ s. In each case, the ridge filter threshold τ_λ has been adjusted according to an expert's choice to -0.07(a), -0.06(b), -0.04(c), and -0.03(d). The resolutions (a) and (b) are not able to capture the two aligned, very close ridges in the bottom right quadrant (i) in (a) and (b). However, the ridges are already present, albeit broken, when we increase the resolution (c), and they are much better captured at (d), although even higher resolution would be required. The ridge (ii) is present in (c) but disappears when we double the resolution (d) due to its insufficient sharpness with respect to the increased resolution.

Man must rise above the Earth—to the top of the atmosphere and beyond—for only thus will he fully understand the world in which he lives.

— Socrates

3 Topology of Inertial Dynamics

TRADITIONAL vector field visualization has a close focus on velocity, and is typically concerned about the dynamics of massless, non-inertial particles. This chapter develops a novel approach to the analysis of the force-induced dynamics of inertial particles. These forces can arise from acceleration fields such as gravitation, but also be dependent on the particle dynamics itself, as in the case of magnetism. Compared to massless particles, the velocity of an inertial particle is not determined solely by its position and time in a vector field. In contrast, its initial velocity can be arbitrary and impacts the dynamics over its entire lifetime. This results in a four-dimensional problem for 2D setups, and a six-dimensional problem for the 3D case. The approach presented here avoids this increase in dimensionality and tackles the visualization by an integrated topological analysis approach. The utility of this approach is demonstrated using a synthetic time-dependent acceleration field, a system of magnetic dipoles, and *N-body* particle systems both in 2D and 3D.

Our everyday life and large parts of the universe are dominated by masses, and the various forces acting upon them. In the continuous setup, prominent examples are acceleration caused by gravitation between bodies, electrostatics, and magnetism. Some of these accelerations can be represented as time-dependent vector fields. However, most traditional techniques for vector field visualization have either instantaneous (local) scope, or are based on the kinematics of massless particles, and thus cannot provide appropriate insight into the dynamics of inertial particles. Beyond that, phenomena like magnetic interaction cannot be represented by vector fields in terms of acceleration.

Topological analysis of vector fields is motivated by the aim to separate their spatiotemporal domain into regions of qualitatively different behavior. For steady vector fields, this is typically achieved by extraction of streamlines that converge in forward or reverse time direction to isolated zeros of the vector field, known as separatrices and critical points. For time-dependent vector fields, Lagrangian coherent structures, which can be obtained as

ridges in the FTLE field, are a counterpart to separatrices, separating regions of qualitatively different behavior for a prescribed time interval.

In this chapter, we present a technique for the analysis of the inertial dynamics of point masses. In contrast to traditional massless particles, whose velocity is given by the vector field at the respective position and time, velocity of inertial particles is also part of the underlying initial value problem. In other words, instead of solving an IVP in space only, it needs to be solved in phase space, which includes the degrees of freedom of velocity. This leads to an increase of dimension by a factor of two, leading to four-dimensional problems for 2D spatial problems, and six-dimensional problems for 3D spatial configurations. We avoid the difficulties with higher-dimensional visualization by separating position and velocity in our approach, which makes sense also from a purely practical point of view, as position and velocity have very different physical meanings and play distinct roles in dynamical systems.

The contributions in this chapter include:

- A counterpart to the finite-time Lyapunov exponent for the analysis of arbitrary inertial dynamics in phase space,
- derived concepts that constrain initial values for analysis,
- decomposition into spread due to velocity and position,
- dimensional stacking for phase-space navigation, and
- a concept to analyze the multiplicity in the underlying maps.

3.1 Method

This section describes the method to adapt regular non-inertial FTLE to inertial, acceleration-driven systems.

3.1.1 Inertial Dynamics

The subject of visualization in our approach is inertial dynamics in terms of accelerations. One source of accelerations are time-dependent acceleration fields

$$\mathbf{a}(\mathbf{x},t) = (a_1(x_1,\ldots,x_n,t),\ldots,a_n(x_1,\ldots,x_n,t))^\top \tag{3.1}$$

assigning each point $\mathbf{x} := (x_1,\ldots,x_n)^\top$ at time t in the domain $\Omega \subseteq \mathbb{R}^n \times \mathbb{R}$ a vector $\mathbf{a} \in \mathbb{R}^n$ with accelerations a_i, $i = 1,\ldots,n$ in the respective dimensions. The trajectory

$\mathbf{x}(t)$ is fully determined by an IVP starting at time t_0 at initial position $\mathbf{x}_0 := \mathbf{x}(t_0)$ with initial velocity

$$\dot{\mathbf{x}}_0 := \dot{\mathbf{x}}(t_0) = \left(\frac{\mathrm{d}x_1(t)}{\mathrm{d}t}, \ldots, \frac{\mathrm{d}x_n(t)}{\mathrm{d}t} \right)^\top \Bigg|_{t=t_0}. \tag{3.2}$$

We first reformulate the IVP in phase space, which is the space composed by the position and velocity spaces:

$$\boldsymbol{\xi}(t_0) := \begin{pmatrix} \mathbf{x}_0 \\ \dot{\mathbf{x}}_0 \end{pmatrix}, \quad \frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\xi}(t) = \begin{pmatrix} \dot{\mathbf{x}}(t) \\ \ddot{\mathbf{x}}(t) \end{pmatrix} = \begin{pmatrix} \dot{\mathbf{x}}(t) \\ \mathbf{a}(\mathbf{x}(t), t) \end{pmatrix}, \tag{3.3}$$

with $\boldsymbol{\xi}(t)$ representing the path of a point mass in phase space. Using the representation in phase space, we now can formulate the inertial dynamics of such a particle in the acceleration field:

$$\mathrm{d}t\boldsymbol{\xi}(t) = \begin{pmatrix} \dot{\mathbf{x}}(t) \\ \mathbf{a}(\mathbf{x}(t), t) \end{pmatrix}. \tag{3.4}$$

The resulting integral formulation

$$\boldsymbol{\xi}(t) = \boldsymbol{\xi}(t_0) + \int_{t_0}^{t} \begin{pmatrix} \dot{\mathbf{x}}(\tau) \\ \mathbf{a}(\mathbf{x}(\tau), \tau) \end{pmatrix} \mathrm{d}\tau \tag{3.5}$$

represents a coupled system, i.e., one needs to solve

$$\mathbf{x}(t) = \mathbf{x}_0 + \int_{t_0}^{t} \dot{\mathbf{x}}(\tau)\, \mathrm{d}\tau \tag{3.6}$$

and concurrently

$$\dot{\mathbf{x}}(t) = \dot{\mathbf{x}}_0 + \int_{t_0}^{t} \mathbf{a}(\mathbf{x}(\tau), \tau)\, \mathrm{d}\tau, \tag{3.7}$$

i.e., Equation 3.6 is needed in Equation 3.7, and vice versa. In our implementation, we accomplish this for time t and time step Δt by means of a coupled fourth-order Runge-Kutta integration scheme:

$$\mathbf{x}_1 = \mathbf{x}(t), \qquad \dot{\mathbf{x}}_1 = \dot{\mathbf{x}}(t), \qquad \mathbf{a}_1 = \mathbf{a}(\mathbf{x}_1, t), \tag{3.8}$$

$$\mathbf{x}_2 = \mathbf{x}_1 + \dot{\mathbf{x}}_1 \Delta t/2, \qquad \dot{\mathbf{x}}_2 = \dot{\mathbf{x}}_1 + \mathbf{a}_1 \Delta t/2, \qquad \mathbf{a}_2 = \mathbf{a}(\mathbf{x}_2, t + \Delta t/2), \tag{3.9}$$

$$\mathbf{x}_3 = \mathbf{x}_1 + \dot{\mathbf{x}}_2 \Delta t/2, \qquad \dot{\mathbf{x}}_3 = \dot{\mathbf{x}}_1 + \mathbf{a}_2 \Delta t/2, \qquad \mathbf{a}_3 = \mathbf{a}(\mathbf{x}_3, t + \Delta t/2), \tag{3.10}$$

$$\mathbf{x}_4 = \mathbf{x}_1 + \dot{\mathbf{x}}_3 \Delta t, \qquad \dot{\mathbf{x}}_4 = \dot{\mathbf{x}}_1 + \mathbf{a}_3 \Delta t, \qquad \mathbf{a}_4 = \mathbf{a}(\mathbf{x}_4, t + \Delta t). \tag{3.11}$$

From this, we obtain the new position $\mathbf{x}(t+\Delta t)$ and velocity $\dot{\mathbf{x}}(t+\Delta t)$ from the previous $\mathbf{x}(t)$ and $\dot{\mathbf{x}}(t)$ as follows:

$$\mathbf{x}(t+\Delta t) = \mathbf{x}_1 + (\dot{\mathbf{x}}_1 + 2\dot{\mathbf{x}}_2 + 2\dot{\mathbf{x}}_3 + \dot{\mathbf{x}}_4)\Delta t/6, \tag{3.12}$$

$$\dot{\mathbf{x}}(t+\Delta t) = \dot{\mathbf{x}}_1 + (\mathbf{a}_1 + 2\mathbf{a}_2 + 2\mathbf{a}_3 + \mathbf{a}_4)\Delta t/6. \tag{3.13}$$

In our experiments, we obtained sufficiently accurate results with fixed step size RK4, although the application of adaptive approaches, such as the Runge-Kutta-Fehlberg method, would be likewise possible.

For phenomena, such as magnetic interaction, where acceleration cannot be represented by a vector field, we replace $\mathbf{a}(\mathbf{x},t)$ with an appropriate function, e.g., $\mathbf{a}(t,\mathbf{x}(t),\dot{\mathbf{x}}(t),\dots)$, in the above formulation.

3.1.2 Finite-Time Mapping

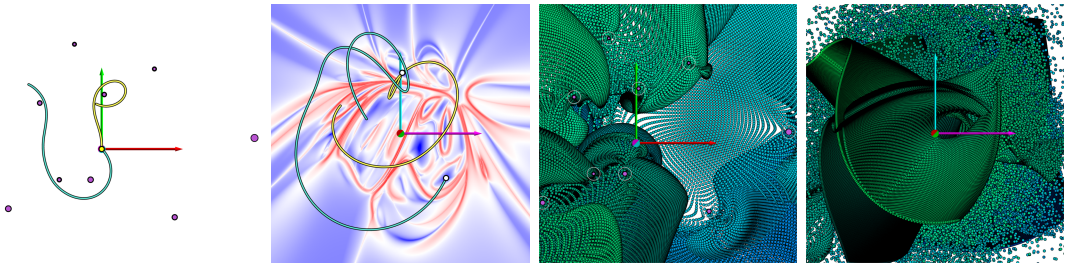
Now that we can solve for the dynamics of point masses over finite advection times T, we can investigate some of their properties. We assume that the underlying accelerations $\mathbf{a}(\cdot)$ are continuous (which is, e.g., the case for N-body systems, magnetism, and tensor-product linearly interpolated fields) both in space and time. In analogy to streamlines in vector fields, this leads to the fact that the pathlines or trajectories $\boldsymbol{\xi}(t)$ of point masses cannot intersect in phase space. Thus, trajectories can converge but cannot reach the same point in phase space within finite time intervals. In other words, the phase-space mapping

$$\boldsymbol{\Phi}_{t_0}^{T} : \boldsymbol{\xi}(t_0) \mapsto \boldsymbol{\xi}(t_0+T) \tag{3.14}$$

is bijective. As a consequence, in phase space, manifolds of point masses can only deform due to inertial dynamics, but not self-intersect or tear apart or merge, i.e., they are topologically invariant.

Figure 3.1 shows an example of a two-manifold of point masses (varying in initial velocity) after transport for a finite time interval. In the spatial projection (Figure 3.1c) and in the velocity projection (Figure 3.1d), one can observe overlaps, but there are no self-intersections in four-dimensional phase space. In contrast, Figure 3.2 shows the same configuration but with a constrained initial velocity. Overlaps are also present in the final configurations of this example.

The insight that the mapping is bijective in phase space has a practical impact: we can always reverse the direction of integration, i.e., we can integrate in reverse direction from $\boldsymbol{\xi}(t_0+T)$ to $\boldsymbol{\xi}(t_0)$. We will exploit this property in Section 3.1.7 to find correspondences.



(a) initial position (b) initial velocity (c) final position (d) final velocity

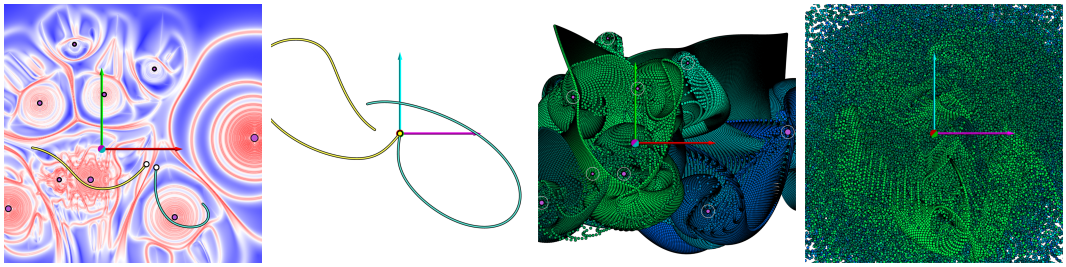
Figure 3.1: Analysis of gravitation-induced inertial dynamics, for the 2D Nine-Body example, with bodies (purple dots, mass-proportional), trajectories (colored), and their seeds (white dots). Views: Initial position (a), initial velocity (b), final position (c), and final velocity (d), with axes denoting x-position (red), y-position (green), x-velocity (magenta), and y-velocity (cyan). Initial position has been constrained to $\mathbf{0}$ (yellow dot in (a)), resulting in degrees of freedom of initial velocity only, which is visualized with PS-FTLE-V (b). All samples of the 600×600 PS-FTLE-V grid after inertial transport for time T shown by dots in (c) and (d) (initial x-velocity mapped to blue channel, initial y-velocity to green channel). © 2017 IEEE.

3.1.3 Phase-Space Finite-Time Lyapunov Exponent

As we have already covered in Section 2.5, topological analysis aims at providing a partitioning of the domain into parts of qualitatively different behavior. For time-dependent vector fields, where massless particles are assumed to strictly follow the vector field, the finite-time Lyapunov exponent [Hal01] field represents the maximum spatial spread of massless particles started at time t_0 infinitesimally close to position $\mathbf{x}$, and has proven successful in revealing their topology. The FTLE has the formulation shown in Equation 2.15, repeated here for convenience:

$$\sigma_{t_0}^{T}(\mathbf{x}) = \frac{1}{|T|} \ln \left\| \nabla \boldsymbol{\phi}_{t_0}^{T}(\mathbf{x}) \right\|_2 . \tag{3.15}$$

with $\boldsymbol{\phi}_{t_0}^{T}(\mathbf{x})$ representing the flow map which maps massless particles started at $\mathbf{x}$ and time t_0 to their position after advection for time T, and $\| \cdot \|_2$ representing the spectral norm (i.e., for a matrix A the square root of the largest eigenvalue of $A^\top A$). Ridges in this field indicate Lagrangian coherent structures (LCS) [SLM05], which are typically codimension-one subsets (ridges) of the domain and separate qualitatively different regions for the finite advection time T.



(a) initial position (b) initial velocity (c) final position (d) final velocity

Figure 3.2: Same as Figure 3.1, but with initial velocity constrained to **0** (yellow point in (b)) and thus PS-FTLE-P field in (a). © 2017 IEEE.

A straightforward approach to analyze inertial dynamics is to apply the FTLE concept to the *inertial flow map*

$$\mathbf{\Phi}_{t_0}^{T}(\boldsymbol{\xi}) := \boldsymbol{\xi} + \int_{t_0}^{t_0+T} \begin{pmatrix} \dot{\mathbf{x}}(\tau) \\ \mathbf{a}(\mathbf{x}(\tau),\tau) \end{pmatrix} \mathrm{d}\tau \tag{3.16}$$

in phase space, leading to the *phase-space finite-time Lyapunov exponent* (PS-FTLE):

$$\varsigma_{t_0}^{T}(\boldsymbol{\xi}) := \frac{1}{|T|} \ln \left\| \nabla \mathbf{\Phi}_{t_0}^{T}(\boldsymbol{\xi}) \right\|_2. \tag{3.17}$$

The PS-FTLE is a (time-dependent) scalar field in the $2n$-dimensional phase-space domain, i.e., for 2D spatial problems it is a 4D scalar field, and for 3D cases it is 6D scalar field. These fields can't be directly visualized, we must explore more elaborate methods.

Multidimensional projection (MP) methods [LV10] have been one of the main alternatives to visualize data residing in spaces with dimension larger than three, and could thus provide solutions to visualize the $2n$-dimensional phase-space finite-time Lyapunov exponent field that we introduce in Chapter 3. Projection methods aim to map data to a visual space such that distances are preserved as much as possible, thus making possible the *visual analysis* of neighborhood structures present in the original data. In fact, in the context of visualization, MP techniques have mainly been used for the visual inspection of clusters and their properties [JPN15], since the analysis of more complex structures is not so straightforward with MP methods. However, MP techniques are difficult to apply and even more difficult to interpret in the context of phase space of inertial particles.

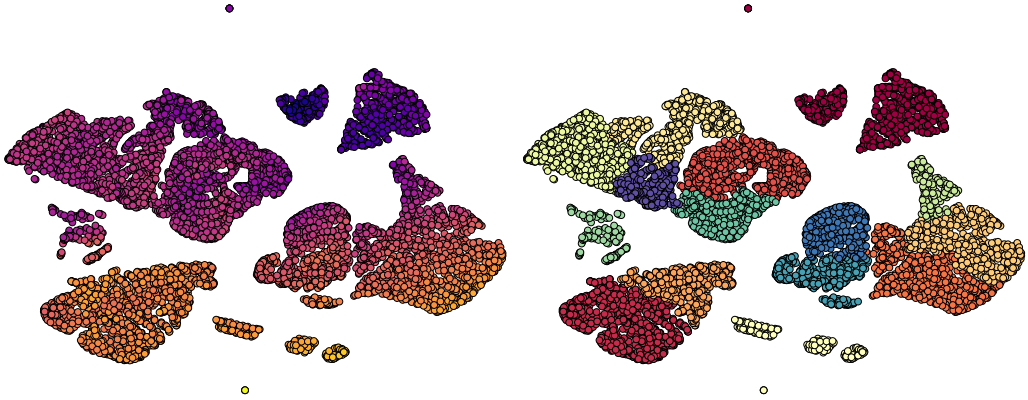
In this fashion, these higher-dimensional fields could be visualized by interactive *hyperslicing* [WL93] or similar approaches, but in these cases the overall picture would need to be "constructed" mentally, which would be cumbersome due to the typically very complex structure of (PS-)FTLE fields. Another approach would be to make use of dimen-

sionality reduction techniques, which aim to map points from 4D (or 6D) to a visual space preserving distances as much as possible [LV10]. Figure 3.3 shows the point cloud resulting from a projection of "ridge points" in 4D to a two-dimensional visual space using t-SNE [VH08] as projection mechanism. The ridge points approximate the ridges by selecting those nodes of a regular 50^4 sampling grid (resolution limited by memory requirements of t-SNE) with PS-FTLE value larger than 0.85. The *t-SNE projection* method has been chosen as projection mechanism due to its ability to reveal groups of similar instances in the visual space. Figure 3.3a shows the resulting projected points colored according to their proximity in 4D (close points have similar colors). As one can see, t-SNE is performing well in terms of neighborhood preservation. Figure 3.3b depicts the same point cloud as in Figure 3.3a, but colored according to cluster labels so as to facilitate group identification. Clusters were computed in the visual space using an agglomerative clustering algorithm [XW05]. Although some clusters clearly show up in the projection, it is difficult to figure out from the projection layout the interplay among the ridges, that is, the resulting clusters do not provide a direct notion of the true separating structures in phase space, nor their properties with respect to inertial dynamics. Moreover, t-SNE took about 36 s to perform the mapping of 5.726 ridge points, hampering interactive applications. Computationally more efficient projection methods could be used, but interactive rates would hardly be reached by this means only.

3.1.4 Constrained PS-FTLE

To avoid the abovementioned difficulties with visualization of higher-dimensional fields, we decouple, to the necessary extent, the analysis of the phase space with respect to position and velocity, which allows us to avoid an increase of dimensionality. Thus, our approach requires only 2D visualization for 2D problems, and 3D visualization for 3D problems. The central idea of our approach is to constrain the degrees of freedom of the initial condition during interactive analysis by selection, but to enable exploration of this choice, supported by a representation that provides overall context and a notion of the impact of the choice. By definition, and also with respect to most research questions, position and velocity play different roles. Thus, in our approach, either initial position or initial velocity are constrained during the interactive process, and the remaining degrees of freedom are those of initial velocity or initial position, respectively.

Assume that we want to shoot a mass ballistically (governed by gravitation only) from a given point. In this case, the initial position is determined by the launching pad (yellow point in Figure 3.1a). Thus, the remaining degrees of freedom are the initial velocity, the starting time, and the duration of flight. Starting time t_0 and duration of flight T are



(a) (b)

Figure 3.3: Ridge points mapped from 4D to 2D using t-SNE projection. (a) Neighbor points in 4D are close in the projection. (b) Agglomerative clustering cannot provide a direct notion of the original separating structures in 4D phase space. © 2017 IEEE.

treated as in FTLE-based examinations described above, i.e., they are typically explored by the user or given by the problem under investigation. This means, they are not treated as a degree of freedom during these types of analysis. Thus, initial velocity are the only degrees of freedom that would need to be examined in this example. The space spanned by t_0 and T is examined in detail in Chapter 4 instead.

In our example, the degrees of freedom of initial velocity are represented by the range of the initial velocity view (Figure 3.1b), and interactive exploration in that view can be accomplished by inertial trajectories starting with the respective velocity and (predetermined initial position) from Figure 3.1a. Since the true trajectory is in phase space, it of course also has a velocity component, which represents a respective trajectory in the initial velocity view. Of course, exhaustive exploration by such interactive trajectories would be cumbersome. But by employing the concept of the PS-FTLE with respect to the remaining degrees of freedom (in this example initial velocity), we can provide a concise representation of the structure of the problem, i.e., of the regions with similar inertial dynamics (Figure 3.1b).

We accomplish this by constraining the PS-FTLE to a fixed initial velocity $\dot{\mathbf{x}}_0$, resulting in the *PS-FTLE-P*:

$$ {}^{\dot{\mathbf{x}}_0}\varsigma^T_{t_0}(\mathbf{x}) := \varsigma^T_{t_0}\begin{pmatrix}\mathbf{x}\\ \dot{\mathbf{x}}_0\end{pmatrix} := \frac{1}{|T|}\ln\left\|\left(\frac{\partial}{\partial x_1},\ldots,\frac{\partial}{\partial x_n}\right)\boldsymbol{\Phi}^T_{t_0}\begin{pmatrix}\mathbf{x}\\ \dot{\mathbf{x}}_0\end{pmatrix}\right\|_2 \tag{3.18}$$

or by constraining the PS-FTLE to a fixed initial position $\mathbf{x}_0$, resulting in the *PS-FTLE-V*:

$$ {}^{\mathbf{x}_0}\varsigma_{t_0}^{T}(\dot{\mathbf{x}}) := \varsigma_{t_0}^{T}\begin{pmatrix}\mathbf{x}_0\\ \dot{\mathbf{x}}\end{pmatrix} := \frac{1}{|T|}\ln\left\| \left(\frac{\partial}{\partial \dot{x}_1},\ldots,\frac{\partial}{\partial \dot{x}_n}\right)\boldsymbol{\Phi}_{t_0}^{T}\begin{pmatrix}\mathbf{x}_0\\ \dot{\mathbf{x}}\end{pmatrix}\right\|_2 . \tag{3.19} $$

Note that

$$ \left(\frac{\partial}{\partial x_1},\ldots,\frac{\partial}{\partial x_n}\right)\boldsymbol{\Phi}_{t_0}^{T}\begin{pmatrix}\mathbf{x}\\ \dot{\mathbf{x}}_0\end{pmatrix} \quad \text{and} \quad \left(\frac{\partial}{\partial \dot{x}_1},\ldots,\frac{\partial}{\partial \dot{x}_n}\right)\boldsymbol{\Phi}_{t_0}^{T}\begin{pmatrix}\mathbf{x}_0\\ \dot{\mathbf{x}}\end{pmatrix} \tag{3.20} $$

are $2n \times n$ matrices both leading to an $n \times n$ matrix and thus n eigenvalues during spectral norm computation. Note also that both variants capture spread in phase space, since in general, $\boldsymbol{\Phi}_{t_0}^{T}(\boldsymbol{\xi})$ maps to varying position and varying velocity, irrespective if initial position or initial velocity are kept constant or not. We denote ${}^{\dot{\mathbf{x}}_0}\varsigma_{t_0}^{T}(\mathbf{x})$ PS-FTLE-P because it captures phase-space spread in terms of varying initial position, and ${}^{\mathbf{x}_0}\varsigma_{t_0}^{T}(\dot{\mathbf{x}})$ PS-FTLE-V since it represents phase-space spread due to varying initial velocity. Please see Figure 3.2a for an example of PS-FTLE-P, and Figure 3.1b for a respective example of PS-FTLE-V. We omit a color legend, because for topological analysis based on FTLE variants, only a qualitative view is needed. We employ a colormap that maps low values to blue, medium to white, and large to red. One can nicely see in these illustrative examples how the constrained PS-FTLE captures the topological structure of inertial dynamics, i.e., how ridges in these fields separate regions of qualitatively different inertial dynamics (exemplified with selected inertial trajectories).

3.1.5 Decomposition of PS-FTLE

The ridges in the PS-FTLE-P and the PS-FTLE-V separate qualitatively different regions in initial position or initial velocity, respectively. By this, they provide a concise representation of inertial dynamics with respect to initial position and velocity. Nevertheless, these fields (and thus also their ridges) represent combined spread only: it is not possible to tell to what extent the detected spread is due to varying final position, and to what extent it is due to varying final velocity.

This motivates, as a complementary visualization component, to decompose PS-FTLE into a part that represents position spread, and another part that represents velocity spread. Due to the abovementioned difficulties with higher-dimensional visualization, we do not provide examples for decomposing the PS-FTLE itself. Instead, we decompose both the PS-FTLE-P and PS-FTLE-V into two fields each. To this end, we first decom-

pose the inertial flow map $\boldsymbol{\Phi}_{t_0}^{T}(\boldsymbol{\xi})$ into a part $\bar{\boldsymbol{\Phi}}_{t_0}^{T}(\boldsymbol{\xi})$ that maps to position, and a part $\tilde{\boldsymbol{\Phi}}_{t_0}^{T}(\boldsymbol{\xi})$ that maps to velocity:

$$\boldsymbol{\Phi}_{t_0}^{T}(\boldsymbol{\xi}) =: \begin{pmatrix} [1.4]\bar{\boldsymbol{\Phi}}_{t_0}^{T}(\boldsymbol{\xi}) \\ \tilde{\boldsymbol{\Phi}}_{t_0}^{T}(\boldsymbol{\xi}) \end{pmatrix}. \tag{3.21}$$

This allows us to decompose PS-FTLE-P into position separation:

$${}^{\dot{\mathbf{x}}_0}\bar{\varsigma}_{t_0}^{T}(\mathbf{x}) := \frac{1}{|T|} \ln \left\| \left(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n} \right) \bar{\boldsymbol{\Phi}}_{t_0}^{T} \begin{pmatrix} \mathbf{x} \\ \dot{\mathbf{x}}_0 \end{pmatrix} \right\|_2 \tag{3.22}$$

(being identical to IFTLE [PD09]), and velocity separation:

$${}^{\dot{\mathbf{x}}_0}\tilde{\varsigma}_{t_0}^{T}(\mathbf{x}) := \frac{1}{|T|} \ln \left\| \left(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n} \right) \tilde{\boldsymbol{\Phi}}_{t_0}^{T} \begin{pmatrix} \mathbf{x} \\ \dot{\mathbf{x}}_0 \end{pmatrix} \right\|_2 . \tag{3.23}$$

Accordingly, we also decompose PS-FTLE-V into position separation:

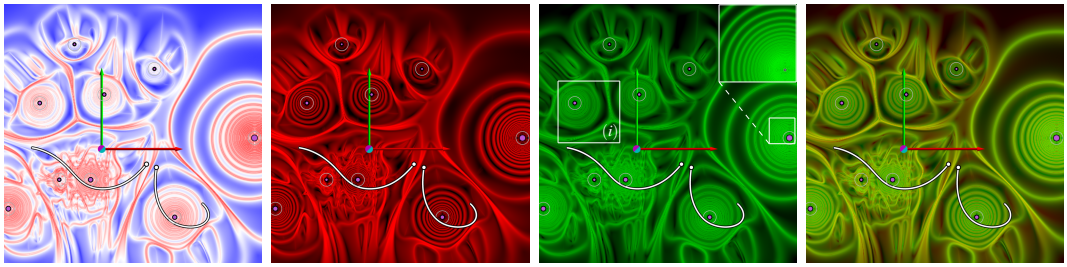
$${}^{\mathbf{x}_0}\bar{\varsigma}_{t_0}^{T}(\dot{\mathbf{x}}) := \frac{1}{|T|} \ln \left\| \left(\frac{\partial}{\partial \dot{x}_1}, \ldots, \frac{\partial}{\partial \dot{x}_n} \right) \bar{\boldsymbol{\Phi}}_{t_0}^{T} \begin{pmatrix} \mathbf{x}_0 \\ \dot{\mathbf{x}} \end{pmatrix} \right\|_2 \tag{3.24}$$

and velocity separation:

$${}^{\mathbf{x}_0}\tilde{\varsigma}_{t_0}^{T}(\dot{\mathbf{x}}) := \frac{1}{|T|} \ln \left\| \left(\frac{\partial}{\partial \dot{x}_1}, \ldots, \frac{\partial}{\partial \dot{x}_n} \right) \tilde{\boldsymbol{\Phi}}_{t_0}^{T} \begin{pmatrix} \mathbf{x}_0 \\ \dot{\mathbf{x}} \end{pmatrix} \right\|_2 . \tag{3.25}$$

Figure 3.4 exemplifies the utility of these decompositions for the case of the 2D Nine-Body example. Note that these measures have different units and thus their combined visualization cannot provide a quantitative view: ${}^{\dot{\mathbf{x}}_0}\bar{\varsigma}_{t_0}^{T}(\mathbf{x})$ has unit 1, ${}^{\dot{\mathbf{x}}_0}\tilde{\varsigma}_{t_0}^{T}(\mathbf{x})$ has unit s^{-1}, ${}^{\mathbf{x}_0}\bar{\varsigma}_{t_0}^{T}(\dot{\mathbf{x}})$ has unit s, and ${}^{\mathbf{x}_0}\tilde{\varsigma}_{t_0}^{T}(\dot{\mathbf{x}})$ has unit 1. Their combination to color channels (red and green in our implementation) can only provide qualitative insights and relations. We therefore normalize them prior to mapping to color. For Figure 3.4, the maximum of position spread of 264.478 has been mapped to maximum red level, and the maximum of velocity spread of 342.153 s^{-1} to maximum green level. Observe that the nested ring structure around the bodies is more clear in Figure 3.4b than in Figure 3.4c, revealing that different types of orbits (different periodicities and thus different Kepler orbital times) are more pronounced with respect to position than velocity.

On the other hand, close inspection of Figure 3.4c reveals that the respective ridges enclosing the bodies reveal a very thin valley line at their center. We examined this case, but



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

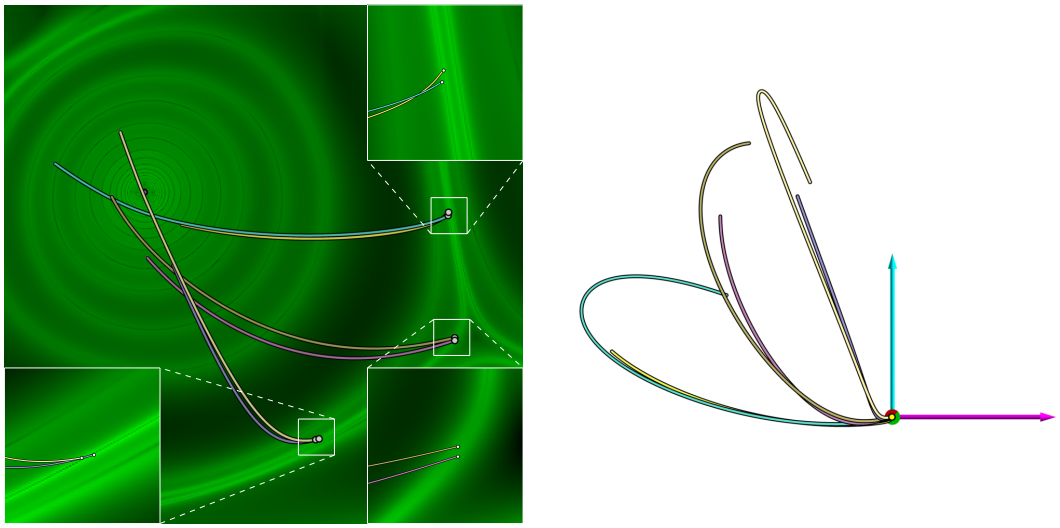
Figure 3.4: Decomposition of PS-FTLE-P ((a) from Figure 3.2a) into position separation $\dot{\mathbf{x}}_0\tilde{\boldsymbol{\varsigma}}^T_{t_0}(\mathbf{x})$ (b) (mapped to red channel), and velocity separation $\dot{\mathbf{x}}_0\tilde{\boldsymbol{\varsigma}}^T_{t_0}(\mathbf{x})$ (c) (mapped to green channel). (d) Combination of both channels reveals relative contribution of position spread and velocity spread. Observe that the ridge where the two trajectories are seeded is more yellowish, and thus position spread is there stronger in relation to velocity spread. This is consistent with the distance of their endpoints in position space (e.g., (d)) and velocity space (Figure 3.2b). Note that both position spread and velocity spread have been normalized because they have different units (1 for $\dot{\mathbf{x}}_0\tilde{\boldsymbol{\varsigma}}^T_{t_0}(\mathbf{x})$, and s^{-1} for $\dot{\mathbf{x}}_0\tilde{\boldsymbol{\varsigma}}^T_{t_0}(\mathbf{x})$). The very thin valleys within the ridges in region (i) are examined in detail in Figure 3.5. © 2017 IEEE.

zooming in did not yield any significant increase the gap width. Nevertheless, Figure 3.5 shows that the valleys are caused by trajectories "returning" in velocity space, caused by the deceleration of particles as they pass a nearby mass. This example demonstrates that structures that can be observed in the PS-FTLE and represent a superposition of position and velocity spreads can often be analyzed in more detail by our decomposition technique.

3.1.6 Stacked PS-FTLE

So far, we have shown how phase space can be analyzed using PS-FTLE-P and PS-FTLE-V, i.e., by constraining the underlying IVP either by selecting an initial position or by selecting an initial velocity. A limitation with this approach alone would be, however, that this selection would not be supported, i.e., that the user would need to explore this choice "blindly" without guidance.

This motivates our stacked PS-FTLE (SPS-FTLE) approach. The SPS-FTLE is inspired by dimensional stacking [LWW90; WBT97] of discrete data in information visualization. Assume we are in the configuration where PS-FTLE-V is used, i.e., where initial position is constrained and the remaining degrees of freedom are initial velocity (Figure 3.1). In this setup, PS-FTLE-V is displayed in the initial velocity view, and the initial position view contained so far only a point representing the selected initial position. To



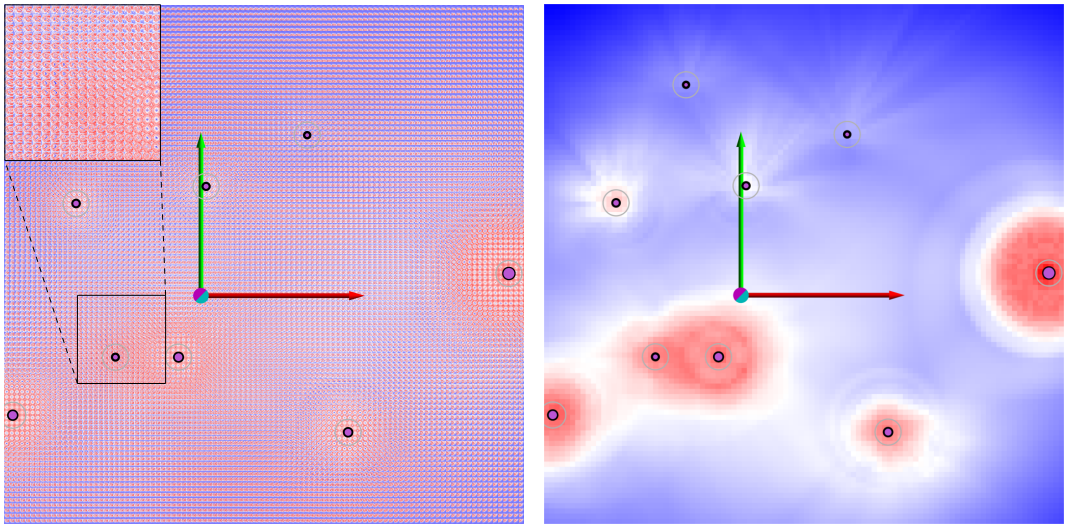
(a) (b)

Figure 3.5: (a) Valleys within ridges of velocity separation (green), within range (i) of Figure 3.4c, with trajectories (colored lines). (b) Same trajectories in velocity space. One can see (zoomed regions) that the valleys separate trajectories with increasing velocity magnitude from those where velocity magnitude decreases again ("returning" velocity curves). The point of maximum velocity magnitude (farthest from origin in (b)) corresponds to shortest distance from mass body (purple point in (a)).

support this selection, we provide context in the respective view (in this setup the initial position view) by presenting there the respective stacked PS-FTLE field.

For the described configuration, the SPS-FTLE representation in the initial position view consists of a grid that discretizes the initial position range, and within each cell of this grid, the PS-FTLE-V field is represented that results if the center of the respective cell is used as initial position (see Figure 3.6a). To avoid unnecessarily long computation times, the resolution of the PS-FTLE-V fields within the cells is kept low (100×100 in this case). For the opposite configuration, i.e., where initial velocity is constrained, the SPS-FTLE field discretizes the range of initial velocity into cells instead, and each cell contains the respective PS-FTLE-P field.

Because the large-scale structure of the SPS-FTLE representation is hard to perceive at medium zoom levels (i.e., one has to zoom out rather far to see the overall structure), we additionally generate a discretized version of the SPS-FTLE representation. This representation is obtained by "merging" the PS-FTLE field within each cell "into a single pixel" which is then shown at the respective resolution, i.e., each cell (or "pixel") of the discretized version represents the respective PS-FTLE-P or PS-FTLE-V field (see Figure 3.6b). In our implementation, the field within a cell is "merged" to a single value



(a) initial position (b) initial position

Figure 3.6: Stacked PS-FTLE at the example of the 2D Nine-Body example. (a) Stacked PS-FTLE with zoomed region (17×17 cells of the stacked PS-FTLE grid). Observe that each cell contains the respective PS-FTLE-V field (e.g., Figure 3.1b). (b) Discretized version of (a), using the average operator, for better context at medium zoom levels. In our interactive implementation, there is a transition from (b) to (a) as the user zooms into the stacked PS-FTLE representation.

using the maximum or average operator. Whereas taking the maximum provides a more conservative view, i.e., high values of the resulting field indicate the maximum separation that appears in the PS-FTLE and thus regions with low values are likely to be of inferior interest, the average operator provides a more quantitative view and, in our experiments, it provided more specific guidance.

3.1.7 Multiplicity Maps

So far, we focused on the views of initial position (e.g., Figure 3.1a) and initial velocity (e.g., Figure 3.1b). There, the constrained PS-FTLE, its decomposition, and its stacking enable the analysis of inertial dynamics with respect to the initial values of the IVP. As exemplified by the trajectories in Figure 3.1 and Figure 3.2, such analysis can answer various research questions (see also Section 3.3). This is similar to traditional (advection-based) FTLE visualization, which is considering the initial view only, i.e., traditional FTLE "resides" at time t_0; the state at time $t_0 + T$, that the flow map maps to, is typically not analyzed. One reason why this final state is typically not analyzed is that the traditional flow map $\boldsymbol{\phi}_{t_0}^{T}(\mathbf{x})$ only represents a deformation without overlap (which follows from an argu-

mentation similar to that in Section 3.1.2), i.e., it is a continuous bijective mapping from the seeds $\mathbf{x}(t_0)$ to the end points $\mathbf{x}(t_0+T)$ of the respective trajectories.

As discussed in Section 3.1.2, the inertial flow map $\boldsymbol{\Phi}_{t_0}^{T}(\boldsymbol{\xi})$ represents a bijective mapping in *phase space*. However, as discussed there too, this mapping is typically not bijective anymore in the respective spatial projections (e.g., Figures 3.1c and 3.2c) and velocity projections (e.g., Figures 3.1d and 3.2d): the n-dimensional manifold defined by the degrees of freedom of initial velocity or initial position, respectively, typically exhibits overlap, i.e., folds. In other words, a point in the final position view or final velocity view is often reached by more than one IVP. As we will see, analyzing these properties provides answers to relevant research questions and thus motivates our final component for analysis of inertial dynamics: *multiplicity maps*So far, we visualized in the final position view (e.g., Figure 3.1c) and final velocity view (e.g., Figure 3.1d) the spatial and velocity projections of the sample points of the constrained PS-FTLE. Since constrained PS-FTLE is sampled in the domain of initial position or initial velocity, it represents an n-manifold with a parametrization induced by these degrees of freedom. Figures 3.1c and 3.1d already provide an impression how the n-manifold resides when projected to the space or velocity domain, since the x-component of the initial value is mapped to the blue channel and the y-component to green. Nevertheless, due to the discrete sampling and due to occlusion of the points, this kind of visualization does not provide the multiplicity of the mapping, i.e., it does not show how many IVPs reach a given point in the projections of the final state. The position multiplicity map

$$\mu(\mathbf{x})_{t_0}^{T} := \left| \left\{ \boldsymbol{\xi} \mid \bar{\boldsymbol{\Phi}}_{t_0}^{T}(\boldsymbol{\xi}) = \mathbf{x} \right\} \right| \tag{3.26}$$

represents how many IVPs starting at time t_0 reach a given position $\mathbf{x}$ after time T. Its counterpart is the velocity multiplicity map

$$\mu(\dot{\mathbf{x}})_{t_0}^{T} := \left| \left\{ \boldsymbol{\xi} \mid \tilde{\boldsymbol{\Phi}}_{t_0}^{T}(\boldsymbol{\xi}) = \dot{\mathbf{x}} \right\} \right| \tag{3.27}$$

which counts how many IVPs starting at time t_0 reach a given velocity $\dot{\mathbf{x}}$ after inertial transport for time T.

In the context of constrained PS-FTLE, both the PS-FTLE-P and the PS-FTLE-V lead each to a respective position multiplicity map and a velocity multiplicity map. The PS-FTLE-P leads to the position multiplicity map

$${}^{\dot{\mathbf{x}}_0}\mu(\mathbf{x})_{t_0}^{T} := \left| \left\{ \mathbf{y} \mid \bar{\boldsymbol{\Phi}}_{t_0}^{T}\begin{pmatrix} \mathbf{y} \\ \dot{\mathbf{x}}_0 \end{pmatrix} = \mathbf{x} \right\} \right| \tag{3.28}$$

and velocity multiplicity map

$$ {}^{\dot{\mathbf{x}}_0}\mu(\dot{\mathbf{x}})_{t_0}^{T} := \left| \left\{ \mathbf{y} \mid \tilde{\mathbf{\Phi}}_{t_0}^{T}\begin{pmatrix} \mathbf{y} \\ \dot{\mathbf{x}}_0 \end{pmatrix} = \dot{\mathbf{x}} \right\} \right|, \tag{3.29} $$

whereas PS-FTLE-V leads to the position multiplicity map

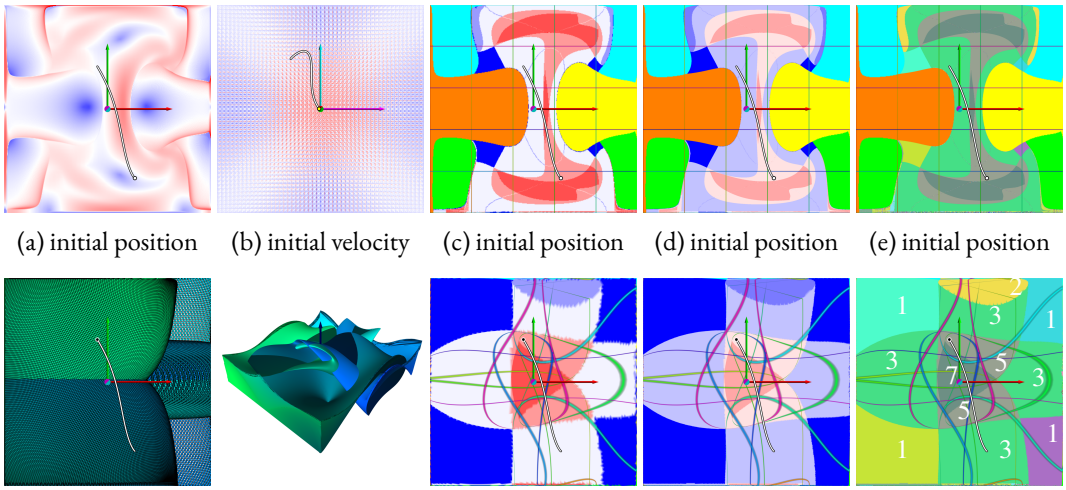
$$ {}^{\mathbf{x}_0}\mu(\mathbf{x})_{t_0}^{T} := \left| \left\{ \dot{\mathbf{y}} \mid \bar{\mathbf{\Phi}}_{t_0}^{T}\begin{pmatrix} \mathbf{x}_0 \\ \dot{\mathbf{y}} \end{pmatrix} = \mathbf{x} \right\} \right| \tag{3.30} $$

and velocity multiplicity map

$$ {}^{\mathbf{x}_0}\mu(\dot{\mathbf{x}})_{t_0}^{T} := \left| \left\{ \dot{\mathbf{y}} \mid \tilde{\mathbf{\Phi}}_{t_0}^{T}\begin{pmatrix} \mathbf{x}_0 \\ \dot{\mathbf{y}} \end{pmatrix} = \dot{\mathbf{x}} \right\} \right|. \tag{3.31} $$

Figure 3.7i shows a position multiplicity map ${}^{\dot{\mathbf{x}}_0}\mu(\mathbf{x})_{t_0}^{T}$ for the Quad-Gyre example, induced by PS-FTLE-P. For better display, we apply a logarithmic scaling and use the same colormap as for the constrained PS-FTLE results. Such representations of the multiplicity map provide insight into how many IVPs reach a respective point or velocity, but they do not reveal which IVPs reach which region. To this end, we perform a connected component labeling of the multiplicity maps, i.e., we detect connected components in the map that exhibit the same multiplicity, i.e., that are reached by the same number of IVPs. Figure 3.7j shows the connected component labeling for the multiplicity map from Figure 3.7i. To reveal the correspondences between the initial values and the connected components in the projection of the final state, we map the connected component labels back to the initial state. We achieve this by looking up for each sample in its final state in phase space the connected component label from the multiplicity field at its projection, and then apply the label to the respective initial value, resulting in a corresponding labeling in the initial position space or velocity space (Figure 3.7e). Together with Figure 3.7j), this reveals the correspondences of the inertial IVPs, i.e., one can see which connected components in the multiplicity map are caused by which regions of initial values.

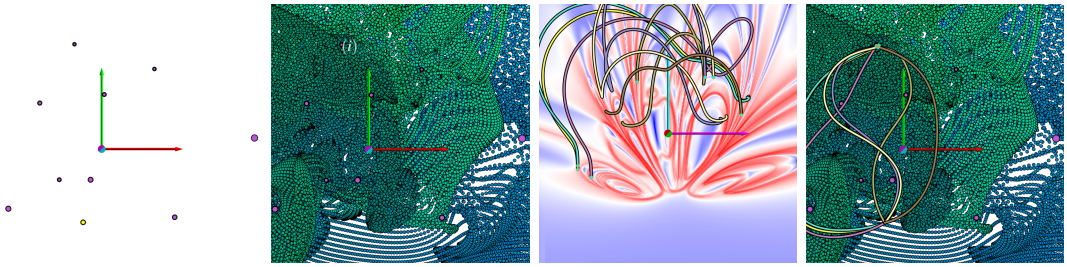
Nevertheless, since both the connected components in the multiplicity map and the corresponding initial regions can be comparably large, this visualization does not provide information on exactly which initial values map to which final state. We address this issue by two complementary approaches. First, we add grid lines that are also mapped to the corresponding view. Second, and more specific, we allow the user to interactively select regions in the final state and display the corresponding initial values (sample points) that reach those regions (Figures 3.8b and 3.8c). This way, the labels and grid lines provide



(a) initial position (b) initial velocity (c) initial position (d) initial position (e) initial position

(f) final position (g) final position × speed (h) final position (i) final position (j) final position

Figure 3.7: PS-FTLE and multiplicity for in Quad-Gyre example. Seed of inertial trajectory by white dot, final state by black dot. (a) PS-FTLE-P. (b) Selected initial velocity (yellow dot) with stacked PS-FTLE-P for context. (f) Final position of phase-space samples. (g) 3D mesh representation with final position × final speed (velocity magnitude, blue axis). (h) Position multiplicity map ${}^{\dot{\mathbf{x}}_0}\mu(\mathbf{x})_{t_0}^{T}$ induced by PS-FTLE-P shows how many IVPs reach a given final position. (c) Multiplicities from (h) mapped to respective initial position view show how many other IVPs reach the same final position (e.g., dark blue (1) means that no other IVP reaches the same final state). Boundaries are mapped: left → orange, right → yellow, top → cyan, bottom → green. (d),(i) Same as (c),(h) but with two levels of adaptive refinement. See how adaptive refinement improves the map. Observe also that some edges were already sharp in (h) because they originate from silhouettes of the folded 2-manifold. The color mapping (but not the multiplicities) typically changes during refinement because silhouettes representing almost "oblique" manifold regions are subject to "discretization noise" (e.g., at bottom boundary), typically caused by particles that are stopped at the domain boundary. (j) Connected components in (i), with multiplicity numbers. Note that in our prototype, these numbers are provided by hovering the mouse over the respective region. (e) Labels from (j) mapped to corresponding initial position. Grid lines (regular in initial space (c), (d), and (e)) provide visual cue to the mapping.



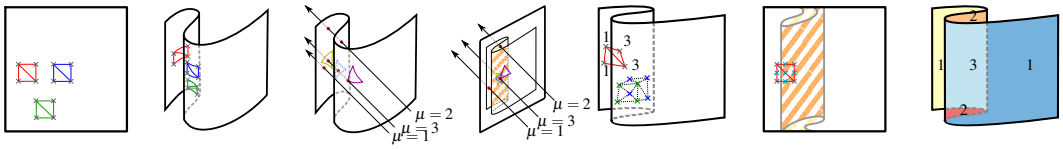
(a) initial position (b) final position (c) initial velocity (d) final position

Figure 3.8: Analysis of inertial dynamics at the example of the 2D Nine-Body example. (a) Nine stationary bodies (purple) inducing gravitational field. The initial position of our 400^2 inertial particles has been manually selected (yellow point), whereas their initial velocity is given by a uniform sampling of the initial velocity space (c). (b) Positions of the samples after inertial transport for time T. We select a small region of interest (i) that we would like to reach from the initial position. This provides the initial velocities (cyan dots in (c)) that lead to the desired region of interest. (c) Inertial trajectories in velocity space, together with phase-space finite-time Lyapunov exponent (color-coded), providing the topological structure and thus regions of similar inertial dynamics. (d) The trajectories in position space show how the desired location can be reached.

context information, and the interactive selection of samples provides explicit information on which IVPs reach which state due to inertial dynamics.

Computation of Multiplicity Maps

Figure 3.9 illustrates the computation of multiplicity maps and the connected components therein. As depicted in Figure 3.9a, the n degrees of freedom (i.e., initial position or velocity range) of the constrained PS-FTLE field are sampled on a *uniform grid* and a mesh is obtained using Delaunay triangulation. We then transport each sample by the inertial IVP to its final state in phase space (Figure 3.9b). For a position multiplicity map, these points are then projected to position space, otherwise they are projected to velocity space (in both cases, the n-manifold is projected from $2n$-dimensional phase space to nD). Let us assume, without loss of generality, that we construct a position multiplicity map. A ray is shot through each of the projected samples and the intersections between the ray and the triangles of the projected mesh are counted, providing the multiplicity count for each sample (Figures 3.9c and 3.9d). This count can then be directly visualized in the initial space (e.g., Figure 3.7d). However, visualization of this map in the final state (e.g., Figure 3.7i) is nontrivial because the n-manifold typically exhibits overlap, and rendering, e.g., using blending, would result in artifacts due to strongly stretched triangles which are generated due to the typically very strong distortions due to finite-time dynamics. There-



(a) (b) (c) (d) (e) (f) (g)

Figure 3.9: Multiplicity map computation. (a) Regular discretization and triangulation of initial space, yielding a flat manifold. (b) Manifold is transformed after transport, third dimension here by mapping to a quantity, e.g., velocity magnitude. (c) Vertex multiplicities represent number of ray intersections with triangles. (d) Intersections are determined in projected space. (e) Refinement (red) is performed for quads with edges that join vertices with different multiplicities (numbers). Projected vertices (from green and blue quad) are triangulated (dotted) to compute connected components in final space. (f) The quad to be refined is subdivided into four subcells by adding five new samples in the initial space. The process (triangulation, multiplicity computation) is then repeated with the new samples. (g) Connected components labeling by traversal in final space, with connectivity defined by equality of multiplicity. © 2017 IEEE.

fore, we instead employ Delaunay triangulation of the projected samples after transport (Figure 3.9e, dotted edges). Then, the planar mesh containing multiplicity values defined at the samples can be rendered (based on barycentric interpolation). The connected component labeling (Figure 3.9g) is achieved by identifying the connected subgraphs where all the vertices have the same multiplicity count. After labeling, the connected components can be visualized in the final state (Figure 3.7j), or mapped back to the initial value space (Figure 3.7e). The optional step of refinement can be processed once the sample multiplicity counts are in place. In that case, we identify the edges that join samples with different multiplicity and subdivide the quads in the initial uniform grid into four subcells, and then repeat the process until convergence or a maximum depth is reached (Figures 3.9e and 3.9f, red quad and green/blue samples).

3.2 Implementation Details

Our implementation uses CUDA (through PyCuda bindings [Klö+13]), when possible, for the processing, and OpenGL (through VisPy gloo bindings [Vis]) for the rendering. We use Python as a general-purpose scripting language to launch the CUDA kernels, fetch the data and pass them to OpenGL for rendering, and to construct and manage the GUI. Some functions cannot be fully implemented in the parallel paradigm that CUDA offers. In such cases we rely on the CPU.

A special use case is the processing of the PS-FTLE field, which is computed in each time step. In this case, a CPU synchronization step between the flow map and the PS-FTLE kernels is needed. Some operations are fully implemented in CUDA (the flow map

computation, all variants of the PS-FTLE, and the interactive trajectories), while others are implemented on the CPU and partially accelerated using CUDA kernels at some key points like the multiplicity maps, the adaptive sampling, and the stacked PS-FTLE. Finally, we have a few operations fully implemented on the CPU, such as the Delaunay triangulation (which uses the Qhull [BDH96] library through SciPy [JOP+01]).

In order to access the vector field or N-body simulation data on the GPU, we use textures. These textures consist of a $T \times N \times M$ matrix in the case of 2D vector fields, and a $T \times N \times M \times L$ matrix in the case of 3D vector fields, where T is the number of time steps and N, M, L are the dimensions of the vector field. In the case of N-body systems, the textures consist of a $T \times N$ matrix, where T is the number of time steps in the original N-body simulation and N is the number of bodies.

Due to the flexibility and symmetry of our approach, we found writing a single CUDA kernel for each operation would render the system unmaintainable. In order to tackle this issue, we set up a templating structure which generates and compiles all possible combinations of kernel files. These kernel files are then accessed and run through indexed tables which are queried using the application's current internal state model. So, for example, the kernel which computes the force acting on a test particle is used to compute the trajectories but also the flow map, and the code is different depending on the input field in use (N-body simulation or vector field).

We have a total of 30 template files (2 046 code lines) which produce, after templating, 40 compilable kernel files. The number of code lines of all final kernel files combined after templating is 9 107.

3.3 Results

We exemplify our approach using one analytic example (Section 3.3.1), a two-dimensional N-body system (Section 3.3.2), a system based on two-dimensional electromagnetic interactions (Section 3.3.3), and a three-dimensional N-body system (Section 3.3.4).

3.3.1 Quad-Gyre

Our first example has been presented by Shadden et al. [SLM05] for the analysis of Lagrangian coherent structures. It represents a time-dependent vector field given in the following analytic representation:

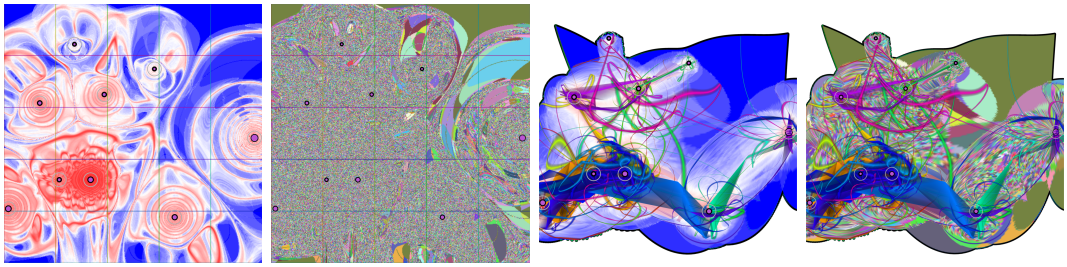
$$\mathbf{a}(\mathbf{x},t) = \begin{pmatrix} -\pi A \sin(\pi f(x,t)) \cos(\pi y) \\ \pi A \cos(\pi f(x,t)) \sin(\pi y) \frac{\mathrm{d}f}{\mathrm{d}x} \end{pmatrix} \tag{3.32}$$

with $f(x,t) = a(t)x^2 + b(t)x$, $a(t) = \varepsilon \sin(\omega t)$, $b(t) = 1 - 2\varepsilon \sin(\omega t)$, $\mathbf{x} = (x,y)^\top$, $\varepsilon = 0.25$, $\omega = \pi/5$, and $A = 0.1$. We sampled this field at a resolution of 50×50 nodes and 100 time steps in the spatial range $\mathbf{x} \in [-1,1] \times [-1,1]$ and temporal range $[0,20]$. Note that we use this y-range instead of the often employed y-range $[0,1]$, leading to a y-symmetric field with four time-dependent vortices instead of two.

We interpret this vector field as an acceleration field and employ our analysis technique. Initial time $t_0 = 0\,\mathrm{s}$ and transport time $T = 3.046\,\mathrm{s}$, with zero initial velocity, resulting in a PS-FTLE-P visualization (Figure 3.7a). One can see from the inertial trajectory that mass released at the respective position reaches the upper half, which would not be possible by advection. Looking at the multiplicity map in initial position space (Figure 3.7d), one can see that many of the ridges in the PS-FTLE-P are caused by the domain boundaries (orange, yellow, cyan, or green color) of our *sampled* vector field. The reason for this is that inertial particles reaching the domain boundary are stopped and accumulate there. The shown trajectory is seeded within one of the regions with low PS-FTLE-P value, representing trajectories that do not reach the boundary. The refined multiplicity map (Figure 3.7i) and in particular its connected component labeling with multiplicity numbers (Figure 3.7j) provide insight into the folding of the manifold at final position and thus inertial dynamics, which would not be possible from direct visualization of the final positions by points (Figure 3.7f).

3.3.2 2D Nine-Body System

A traditional problem that involves inertial dynamics is the motion of masses in gravitation fields. To this end, we integrated a N-body simulation in our interactive implementation. The 2D Nine-Body example consists of nine bodies with different masses at fixed positions. To avoid singularities at the body centers, a softening length [Deh01; RS] is employed and visualized with gray circles. Our approach is used to analyze the gravitational field induced by the bodies. Please see the accompanying video for further examples, where the bodies also undergo inertial dynamics. In all these cases, the initial value problems (point masses) underlying our technique are influenced only by the gravitational field of the bodies, i.e., the test particles do not influence each other. Although this could be accomplished at some additional computational cost, it would for most research questions be inappropriate, because our approach, in accordance with traditional FTLE, aims at analyzing the choice of an IVP, not the analysis of mutually dependent test particles interacting with the bodies and themselves. Figures 3.1, 3.2 and 3.8 provide an analysis using PS-FTLE-V and PS-FTLE-P fields. It can be nicely seen how ridges in the PS-FTLE-V field separate different inertial dynamics with respect to initial velocity, whereas



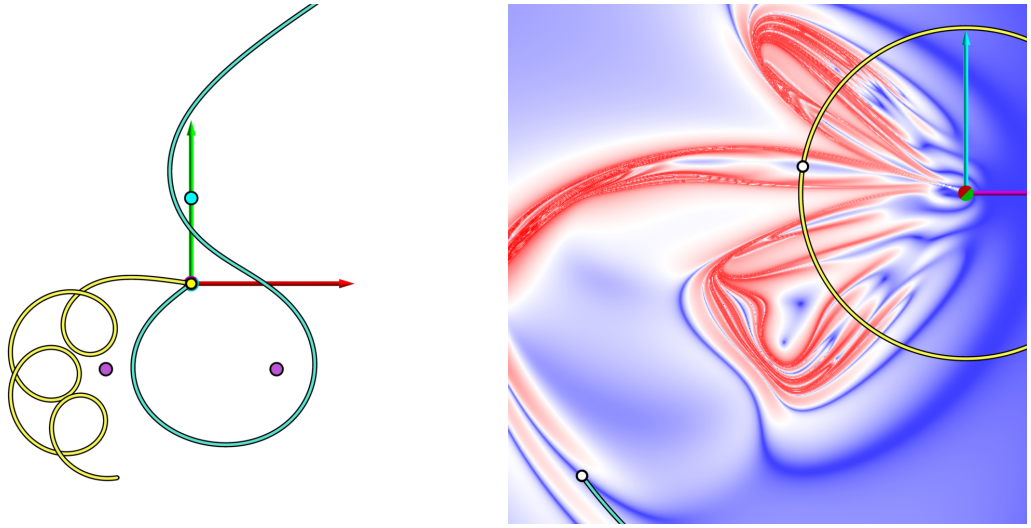
(a) initial position (b) initial position (c) final position (d) final position

Figure 3.10: Position multiplicity map in 2D Nine-Body example. (c) Position multiplicity map ${}^{\dot{\mathbf{x}}_0}\mu(\mathbf{x})_{t_0}^{T}$, induced by PS-FTLE-P. (a) Multiplicities from (c) mapped to corresponding initial values. (d) Connected components in (c). (b) Component labels from (d) at respective initial positions. © 2017 IEEE.

the PS-FTLE-P shows regions of different inertial dynamics due to varying initial position. Since these constrained PS-FTLE fields represent a superposition of position spread and velocity spread, we investigated these spreads individually by PS-FTLE decomposition (Figure 3.4). This analysis provided a surprising result: position spread captures the variation in dynamics around the gravitational bodies better than velocity spread. On the other hand, the large-scale ridges in velocity spread exhibit very thin valley lines (discussed in Section 3.1.5 and Figure 3.5). Figure 3.6a shows stacked PS-FTLE-V that supports the choice of an initial position, i.e., helps navigate the $2n$-dimensional phase space. Finally, we investigated multiplicity maps (Figure 3.10). We have chosen for this the same transport time $T = 4.246\,\mathrm{s}$, as for the other result images. Figure 3.10b reveals the chaotic inertial dynamics of this example at this transport time: one can observe "islands of stability in a sea of chaos". Due to the extremely strong stretching (see stretched grid lines in Figures 3.10c and 3.10d) and mixing of trajectories (i.e., chaotic transport), the connected components of the position multiplicity map are disrupted in chaotic "noise", too (Figure 3.10d), and the grid is distorted to an extent that would require extremely high refinement. Nevertheless, as can be seen in Figure 3.10a, the multiplicity-labeled initial position representation is still able to provide a good notion of IVP multiplicity: it shows for each initial value the count of other IVPs reaching the same final state (observe the quantization of this image which represents these counts).

3.3.3 Magnetic Dipoles

Our approach lends itself for the analysis of any type of inertial dynamics. We exemplify this with a special case that cannot be represented by an acceleration (vector) field: the motion of charged particles due to Lorenz forces, i.e., the motion of such particles in



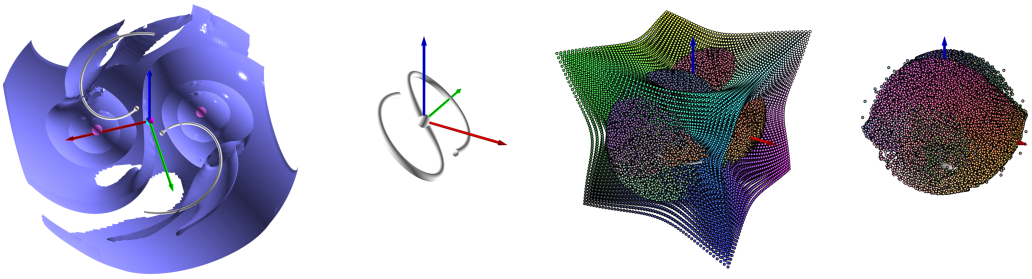
(a) initial position (b) initial velocity

Figure 3.11: Magnetic Dipoles example. (a) Initial position set to **0** (yellow dot), with magnetic dipoles (magnetic moment by dot color) and trajectories (colored lines). (b) PS-FTLE-V visualizing regions of qualitatively different dynamics induced by Lorenz force, with seeds (white dots).

magnetic fields. Such dynamics exhibits "helical" motion which not only depends on the strength of the magnetic field, but also on the velocity of the charged particle. For clarity and ease of representation, we have chosen a configuration that results in purely 2D dynamics. Nevertheless, it would be straightforward to adapt the next example (Section 3.3.4), which examines 3D N-body systems, accordingly.

Our example follows the 2D Nine-Body example in that it exhibits discrete objects (in this case magnetic dipoles) with varying properties (in this case different magnetic moment), and it also employs a softening length to avoid numerical issues due to singularities at the dipole centers. To obtain purely 2D inertial dynamics, all dipoles are located on, and oriented perpendicular to, the xy-plane, and we examine only initial xy-positions and initial xy-velocities. Note that, as in the case of the 2D Nine-Body example, this simulation is integrated in our interactive visualization system and thus does not involve artificial domain boundaries and is fully parametrizable.

Because initial velocity plays the more important role, we fix initial position to **0** and investigate the problem using PS-FTLE-V (Figure 3.11b). We chose two inertial trajectories by selecting different initial velocities (see also their spatial representation in Figure 3.11a) to exemplify the different dynamics of the regions in the PS-FTLE-V field, which are separated by respective ridges. We do not provide here the point-based visualization of final



(a) initial position (b) initial velocity (c) final position (d) final velocity

Figure 3.12: 3D Two-Body example. (a) Height ridge surfaces of PS-FTLE-P nicely separate different types (periodicities) of orbits (bodies by purple spheres, trajectories by white tubes, seeds by white spheres). (b) Respective trajectories in velocity space (note that we did not introduce a new color scheme for 3D velocity axes). (c) Final positions of PS-FTLE-P samples. (d) Final velocities.

position and final velocity, because they are both heavily convoluted and provide little insight.

3.3.4 3D Two-Body System

As our last example, we exemplify that our approach lends itself equally well for the visualization of 3D problems. To this end, we set up a 3D variant of the Nine-Body example. In this case, we have two bodies that are not fixed but move according to an N-body system in 3D, i.e., they orbit each other. Figure 3.12 shows our results for an initial velocity of $\mathbf{0}$, transport time $T = 12\,\mathrm{s}$, with a PS-FTLE-P sampling grid of 120^3, running at about 3.1 FPS, without including the N-body simulation, which is pre-computed. Instead of direct visualization of the constrained PS-FTLE field, we extracted height ridge surfaces [Ebe96].

3.4 Performance Analysis

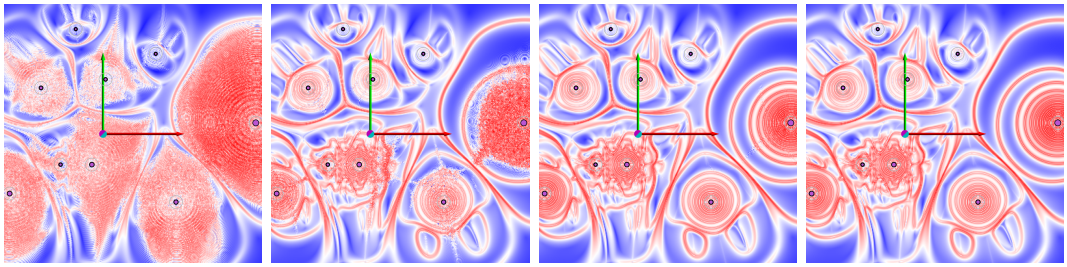
We have run two series of performance tests to analyze how the number of time steps and the grid resolution affect the compute time. To run the tests, we have used the 2D Nine-Body example with a softening length $e^2 = 0.05$. All tests have been run on a system with Manjaro Linux 15.12, kernel 4.4.5.1, 16 GB of RAM memory, an Intel Core i7-4790K CPU at 4.4 GHz, and a GeForce GTX 970 4 GB graphics card. For the performance analysis, if not stated differently, we have used the 2D Nine-Body example with a transport time $T = 4.0\,\mathrm{s}$, a time step $\Delta t = 0.001\,\mathrm{s}$, and a grid of 400×400 particles. We have found that the computing time scales linearly with the number of steps, see Table 3.1 for

the performance results. To analyze the impact of grid resolution, we have chosen four representative resolutions, 200×200, 400×400, 600×600, and 800×800. The results show (Table 3.1) that the computing time also scales linearly with the number of samples, as expected.

The stacked PS-FTLE example in Figure 3.6 took 2682.239 s to compute, of which 2653.038 s where spent running CUDA kernels. The rest was invested in the CPU loop. The multiplicity maps without adaptive sampling with the Quad-Gyre dataset shown in Figures 3.7c and 3.7h took 177.121 s to compute, of which 1.584 s were spent computing vertex multiplicity, 149.454 s in Delaunay triangulation, and 0.733 s in the final color mapping. The two levels of adaptive sampling used to produce Figures 3.7d, 3.7e, 3.7i and 3.7j took 6866.671 s to compute. The multiplicity maps without connectivity of this same example, after the adaptive sampling, took 202.827 s, of which 6.394 s were for computing vertex multiplicity, 96.855 s for Delaunay triangulation, and 1.460 s for the final color mapping. Finally, the version with connected components of the same example took 245.657 s to compute, of which 6.375 s was needed for computing vertex multiplicity, 94.764 s for Delaunay triangulation, and 48.478 s were spent in connected component labeling. Figure 3.13 shows the impact of the variation of the step size, as used for the measurements. Figure 3.14 shows the resolutions used for the measurements.

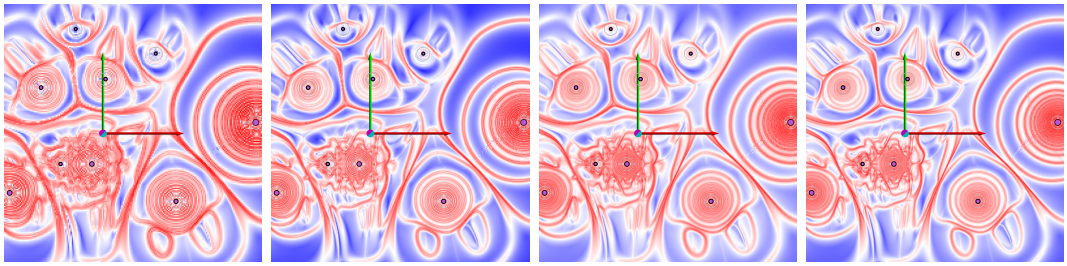
Table 3.1: Performance results in frames per second for different grid resolutions and time steps. $T = 4.0$ s in all tests. © 2017 IEEE.

Test	Fig.	Grid	Δt	Steps	Samples	FPS
Grid	3.14a	200×200	0.001	4 000	40 000	1.0080
	3.14b	400×400			160 000	0.2899
	3.14c	600×600			360 000	0.1322
	3.14d	800×800			640 000	0.0748
Δt	3.13a	400×400	0.1	40	160 000	77.230
	3.13b		0.01	400		10.120
	3.13c		0.001	4 000		1.008
	3.13d		0.0001	40 000		0.107



(a) $\Delta t = 0.1\,\mathrm{s}$ (b) $\Delta t = 0.01\,\mathrm{s}$ (c) $\Delta t = 0.001\,\mathrm{s}$ (d) $\Delta t = 0.0001\,\mathrm{s}$

Figure 3.13: PS-FTLE-P with different step sizes using the Nine-Body example (grid 400×400, $T = 4.0\,\mathrm{s}$). (a) Integration with $\Delta t = 0.1\,\mathrm{s}$ results in severe artifacts, whereas with (d), no artifacts are perceivable. In the accompanying video, we set Δt between $0.001\,\mathrm{s}$ and $0.02\,\mathrm{s}$ depending on the experiment. See Table 3.1 for respective performance measurements. © 2017 IEEE.



(a) 200×200 (b) 400×400 (c) 600×600 (d) 800×800

Figure 3.14: PS-FTLE-P with different grid resolutions using the Nine-Body example ($T = 4.0\,\mathrm{s}$, $\Delta t = 0.001\,\mathrm{s}$). (a) With a grid resolution of 200×200, samples can clearly be seen and fine structures cannot be resolved, whereas with (d), very fine structures can be resolved. In the accompanying video, we used a grid resolution of 200×200 in the introduction and multiplicity maps sections, and a grid resolution of 100×100 in the stacked PS-FTLE section. © 2017 IEEE.

3.5 Discussion

In this chapter, we presented a novel approach for the analysis of inertial dynamics in terms of initial value problems in n-dimensional space. For this, we extended the concept of the finite-time Lyapunov exponent (FTLE) to $2n$-dimensional phase space, leading to phase-space FTLE (PS-FTLE). By introducing constrained PS-FTLE, we are able to avoid direct visualization of this higher-dimensional space, leading to visualization in nD. To enable the analysis of the contribution of position spread versus velocity spread, we introduced decomposition of the PS-FTLE. To provide guidance for the exploration of the $2n$-dimensional space of initial values, we presented stacked PS-FTLE. Finally, for the analysis of the interrelation of initial value problems in phase space, we presented multiplicity maps, their effective refinement, computation of connected components therein,

and complemented this approach with interactive selection of initial values with respect to final states.

As future work, we plan to apply our approach to various fields in science and engineering, and investigate novel approaches for analysis of systems with dimension larger than three, including time.

The history of astronomy is a history of receding horizons.

— Edwin P. Hubble

4 Visual Analysis of the FTLE

In the previous chapter, we have extended the regular, non-inertial FTLE formulation to be able to capture and analyze the complexity of the inertial motion found in most astrophysical systems. As mentioned in Section 3.1.4, we have treated neither the starting time t_0 nor the advection or transport time T as degrees of freedom, and thus we do not currently have a mechanism at our disposal to truly explore these dimensions. However, astrophysical systems, such as the stars in an open cluster, whose movement is only governed by gravitational interactions, are very dependent on these two quantities. In this chapter, we build upon the previous and, focusing in the time domain, we develop a methodology to visually analyze dynamical systems in the context of their various time dimensions.

In time-dependent flow, advection and inertia-based transport do not only depend on *where* a tracer material is injected into the flow and for *what duration* it is transported, but also *at what time* it is released. Essentially, and omitting initial velocity for simplicity's sake, for an n-dimensional time-dependent vector field $\mathbf{u}(\mathbf{x},t)$ which assigns a vector $\mathbf{u} \in \mathbb{R}^n$ to each position $\mathbf{x} \in \Omega \subset \mathbb{R}^n$ within the spatial domain Ω at time t, advection is defined by the seeding position $\mathbf{x}$, the chosen seeding time t_0, and the advection duration T. Thus, it has $n+2$ degrees of freedom, or in other words, $n+2$ dimensions.

Since exploring the entire (n+2)-dimensional space would be prohibitively tedious, summarization and reduction strategies have proven very useful. As we have seen in Chapter 2, in steady (time-independent) vector fields, this is achieved with traditional vector field topology, which is based on instantaneous advection curves, known as streamlines (Section 2.4.1). Those streamlines that converge in forward or reverse time to saddle-type critical points, which are isolated zeros of $\mathbf{u}(\mathbf{x})$, separate such regions, and are thus named separatrices. In time-dependent flow, LCS (Section 2.5.3) take over the role of separatrices. As we have shown in Section 2.5.7, and motivated by Haller [Hal01] and shown by Shadden et al. [SLM05], LCS can be obtained as ridges, i.e., typically codimension-one

manifolds with locally highest value, in the finite-time Lyapunov exponent (FTLE) field. The FTLE field $\sigma_{t_0}^{T}(\mathbf{x})$ is a scalar field in the n-dimensional domain Ω, and the $(n-1)$-dimensional LCS provide such a summarization and reduction—however, only for the seeding time t_0 and advection time T used to compute the FTLE field.

Although the FTLE is a very powerful tool for understanding advection in time-dependent flow, it has not reached wide application yet. Its comparably high computational cost (a trajectory has to be computed for each of its sample points in space and time) might not be the main reason for that, since several acceleration strategies have been proposed [Gar+07; SP07; SRP11]. It is much more its highly intricate parametrization and difficult interpretation that impede successful application by the non-expert. For example, T has to be chosen large enough to capture the phenomenon of interest, but small enough to prevent aliasing at the chosen spatial resolution of $\sigma_{t_0}^{T}(\mathbf{x})$. Also, analysis of the two-dimensional space spanned by t_0 and T for relevant structure is not amenable by direct interactive exploration without supporting context. It is the goal of this chapter to provide an approach that overcomes such issues and helps FTLE-based flow analysis reach its deserved applications by making it easier to apply and interpret. Although large parts of our technique generalize to any dimension, we focus on two-dimensional vector fields and leave the extension to 3D for future work.

The contributions of this chapter include:

- The concept of *aggregation* functions in the t_0–T space, and
- aggregation function definitions to analyze: basic properties,
- height ridge configurations,
- aliasing and resolution issues, and
- the overall "connectedness" of FTLE fields.
- Finally, an integrated framework to aid in the exploration and analysis of these aggregation functions and the FTLE.

4.1 Basic concepts

Fundamental concepts of LCS and FTLE are well covered in Sections 2.5.3 and 2.5.5. An introduction to height ridges is provided in Section 2.5.7. In this present section we only discuss the points relevant only to this chapter. As we have seen in Section 2.5.5, FTLE is a measure of separation of particles seeded at nearby points at time t_0 after advection time T, and has the formulation shown in Equation 2.15, repeated here for convenience:

$$\sigma_{t_0}^{T}(\mathbf{x}) = \frac{1}{|T|} \ln \left\| \nabla \boldsymbol{\phi}_{t_0}^{T}(\mathbf{x}) \right\|_2 . \tag{4.1}$$

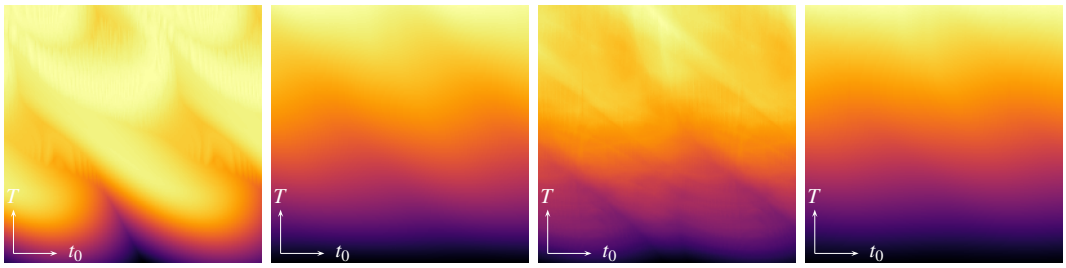
Note that increasing T typically leads to a decrease of the overall FTLE values, since the particles would need to diverge exponentially over the entire duration T to keep the overall FTLE values constant. We counteract this effect, which can hinder visual analysis of FTLE behavior, by optional multiplication of the FTLE values with $|T|$, prior to, e.g., applying aggregation functions (see Section 4.2). Notice also that the sampling of the FTLE grid does not need to (and typically does not) coincide with the grid nodes of the vector field $\mathbf{u}(\mathbf{x}, \mathbf{t})$. In fact, the spatial resolution of the FTLE grid is an important parameter, which needs to be chosen sufficiently high to prevent aliasing, or in other words, to appropriately capture the structure of the FTLE. Furthermore, we do not employ acceleration strategies for FTLE computation (except for parallel computation on the GPU), and we sample it for a region of interest of the space spanned by t_0 and T, including derived measures, which leads to very expensive computation. Thus, in this chapter we employ pre-processing, which, however, is carried out as a batch process since it requires no user interaction whatsoever.

4.2 Method

We now describe our visual analytics approach in terms of derived measures and their visual representation. In Section 4.3, we focus on interaction and implementation aspects, followed by an evaluation in Section 4.4. Performance and optimization considerations are discussed in detail in Section 4.5.

4.2.1 Aggregation Fields

The most basic challenge with FTLE-based flow visualization is the selection of a seeding time t_0 together with an appropriate advection time T. In addition, the temporal structure of vector fields can be very rich, making it difficult to determine relevant instants of time t_0. Notice that it is common practice in FTLE-based visualization to choose a T and then use it to compute the FTLE field $\sigma_{t_0}^{T}(\mathbf{x})$ for a finely resolved sequence of t_0, leading to respective animations. Whereas such animated visualizations might work quite well once the spatiotemporal region of interest, as well as the appropriate advection time, has been determined, they do not provide temporal context, nor do they support finding such relevant regions of interest. Additionally, the exploration of t_0 is complicated by the fact that t_0 and T are intertwined—their selections impact each other. And furthermore, processes at different locations and different times t_0 in a given vector field typically require different choices of T, imposing hard challenges in choosing appropriate combinations of t_0 and T.



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

Figure 4.1: Basic aggregation functions for the Quad-Gyre dataset with aggregation field resolution $G_{\text{agg}} = 200 \times 200$, FTLE resolution $G = 200 \times 200$, $t_0 \in [0,8]$, and $T \in [1,8]$. $f_{\sigma_{\text{avg}}}$ (b) displays a very soft structure apart from the direct proportionality with T. By contrast, $f_{\sigma_{\text{max}}}$ (a) captures the periodicity of the dataset surprisingly well. We also provide, for comparison, the 95th percentile (c), and the sum of squares (d) aggregation fields.

We support the exploration of regions of interest in the space spanned by t_0 and T (denoted $\Omega_{t_0,T}$) by introducing *aggregation fields* $f_\alpha(\mathbf{y})$:

$$f_\alpha : \Omega_{t_0,T} \to \mathbb{R}\,, \tag{4.2}$$

which map each point $\mathbf{y} \in \Omega_{t_0,T} \subset \mathbb{R}^2$ to the scalar result of an aggregation function α. That is, each point $\mathbf{y}$ corresponds to an FTLE field $\sigma_{t_0}^T(\mathbf{x})$, and the aggregation function

$$\alpha : (\Omega \to \mathbb{R}) \to \mathbb{R} \tag{4.3}$$

takes the *field* $\sigma_{t_0}^T(\mathbf{x})$ as input and outputs a single scalar value. Aggregation fields are displayed in the aggregation panel (Figure 4.9) of our system, providing insight into $\Omega_{t_0,T}$ in FTLE-based flow analysis. Notice that we map t_0 to the abscissa and T to the ordinate in the visual analytics framework.

We found the following aggregation functions particularly useful for summarizing important properties and trends of FTLE fields and LCS in a single scalar value. We divide them into four distinct groups according to their use: basic aggregation (Section 4.2.2), ridge aggregation (Section 4.2.3), aliasing aggregation (Section 4.2.4), and region aggregation (Section 4.2.5).

4.2.2 Basic Aggregation Functions

As motivated above, a basic need in FTLE-based flow visualization is to support the exploration of combinations of t_0 and T. We evaluated the summarization of an FTLE field for each $\mathbf{y} \in \Omega_{t_0,T}$ by aggregation functions computing its minimum, maximum,

average, median, sum of squares, root mean square, and 95th percentile—and identified the maximum and average as the generally most useful ones. The maximum aggregation function is defined as follows:

$$\sigma_{\max}(\sigma_{t_0}^T(\mathbf{x})) := \max_{\hat{\mathbf{x}} \in \hat{\Omega}} \sigma_{t_0}^T(\hat{\mathbf{x}}) \,, \tag{4.4}$$

with $\hat{\Omega}$ being the discrete domain of the FTLE field, i.e., the set of sampling grid nodes of our node-based representation of $\sigma_{t_0}^T(\mathbf{x})$, and $\hat{\mathbf{x}}$ representing such a node. An example of the resulting aggregation field $f_{\sigma_{\max}}(\mathbf{y})$ is shown in Figure 4.1a.

The average aggregation function is defined as:

$$\sigma_{\text{avg}}(\sigma_{t_0}^T(\mathbf{x})) := \frac{1}{|\hat{\Omega}|} \sum_{\hat{\mathbf{x}} \in \hat{\Omega}} \sigma_{t_0}^T(\hat{\mathbf{x}}) \,, \tag{4.5}$$

with $|\hat{\Omega}|$ being the cardinality of $\hat{\Omega}$, i.e., the number of nodes in the FTLE sampling grid. An example for the aggregation field $f_{\sigma_{\text{avg}}}(\mathbf{y})$ resulting from the average aggregation function $\sigma_{\text{avg}}(\cdot)$ is shown in Figure 4.1b.

4.2.3 Ridge Aggregation Functions

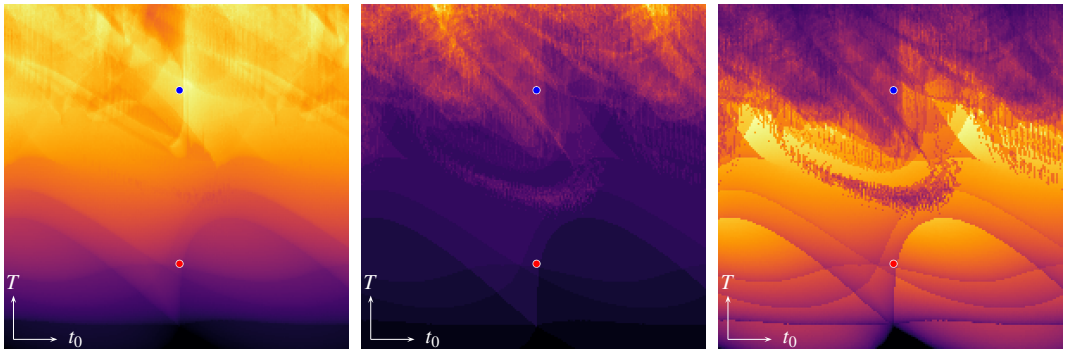
Since FTLE ridges represent LCS, which in turn are the topological features of time-dependent vector fields, the total amount of ridges is a basic measure for the topological structure of a time-dependent vector field. Thus, our first ridge aggregation function

$$\rho_{\text{len}}(\sigma_{t_0}^T(\mathbf{x})) := \sum_{r \in \mathscr{R}} \mu(r) \,, \tag{4.6}$$

simply measures the total length of all height ridges extracted from the FTLE field $\sigma_{t_0}^T(\mathbf{x})$, with $\mathscr{R}$ being the set of all ridges (in polyline representation), and $\mu(r)$ measuring the length of ridge r. Figure 4.3 shows an example result of the ridge extraction process and is annotated with the respective ρ_{len} values. Figure 4.2a shows the aggregation field $f_{\rho_{\text{len}}}(\mathbf{y})$, resulting from the ridge length aggregation function $\rho_{\text{len}}(\cdot)$.

The number of ridges, on the other hand, can capture if ridges are disrupted, e.g., due to insufficient resolution of the FTLE field. This is a major issue in FTLE-based visualization (Figure 2.8). Thus, our next ridge aggregation function

$$\rho_{\text{cnt}}(\sigma_{t_0}^T(\mathbf{x})) := |\mathscr{R}| \,, \tag{4.7}$$



(a) (b) (c)

Figure 4.2: $f_{\rho_{\text{len}}}$ (a), $f_{\rho_{\text{cnt}}}$ (b), and $f_{\hat{\rho}_{\text{len}(k)}}$ (c) aggregation fields, applied to the Quad-Gyre dataset, with $\tau_\lambda = -0.05$ and the same parameters as for Figure 4.1. Note that (a) shows a rather continuous field, while (b) contains only integer numbers of ridges, and (c) displays a similar structure due to its dependency on the number of ridges. Additionally, while ρ_{len} increases with T due to longer, sharper ridges, $\hat{\rho}_{\text{len}(k)}$ exhibits in (c), toward higher T values, a sharp drop in the quality of extracted ridges caused by aliasing. The rather noisy top part in all fields shows that the chosen spatial resolution is unable to capture the ridges at high T values well, so they break up in an unpredictable manner causing these patterns.

counts the number of height ridges extracted from the FTLE field $\sigma_{t_0}^T(\mathbf{x})$, with $|\mathscr{R}|$ being the number of elements in $\mathscr{R}$. Figure 4.2b shows the respective aggregation field $f_{\rho_{\text{cnt}}}(\mathbf{y})$.

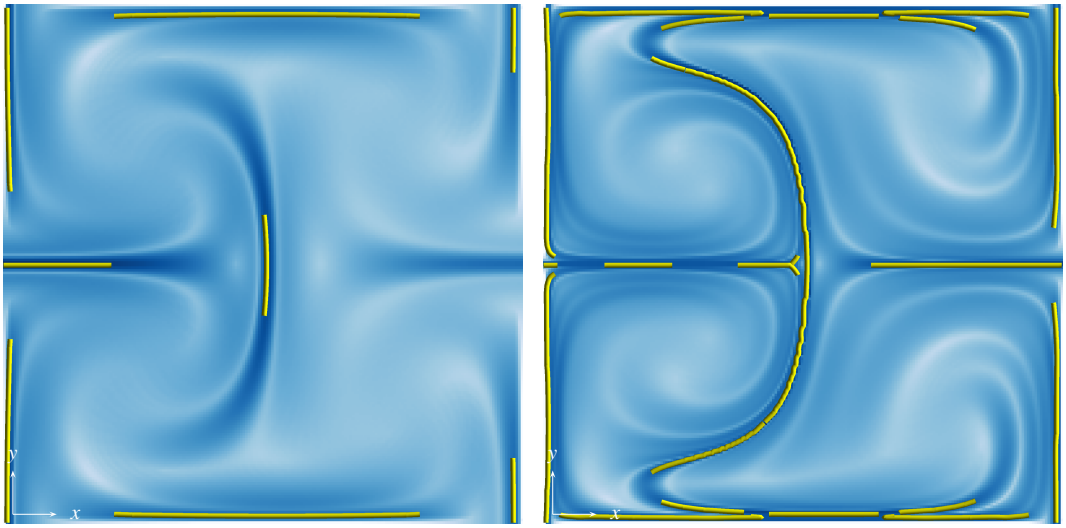
Since LCS, i.e., FTLE ridges, separate regions of qualitatively different time-dependent advection, their individual length is also important—a long ridge represents a larger barrier in the domain and thus also represents a more significant topological structure (Figure 4.3). This motivates our last ridge aggregation function

$$\hat{\rho}_{\text{len}(k)}(\sigma_{t_0}^T(\mathbf{x})) := \frac{1}{|\mathscr{R}|} \sum_{r \in \mathscr{R}} \mu(r)^k \,, \tag{4.8}$$

which measures the averaged kth power of the ridge lengths. It provides a good measure of the overall quality of the ridge extraction stage, and, implicitly, discretization quality (appropriateness of the resolution) of the FTLE field. Figure 4.2c shows the respective aggregation field $f_{\hat{\rho}_{\text{len}(k)}}(\mathbf{y})$ for $k = 1$.

4.2.4 Aliasing Aggregation Function

The ridge count $\rho_{\text{cnt}}(\cdot)$ and ridge length power $\hat{\rho}_{\text{len}(k)}(\cdot)$ aggregation functions are already able to indicate aliasing issues in the FTLE field $\sigma_{t_0}^T(\mathbf{x})$. However, they do that only indirectly via the properties of the ridges extracted from $\sigma_{t_0}^T(\mathbf{x})$. Since these ridge aggregation functions also capture the structure of the LCS and are thus superimposed



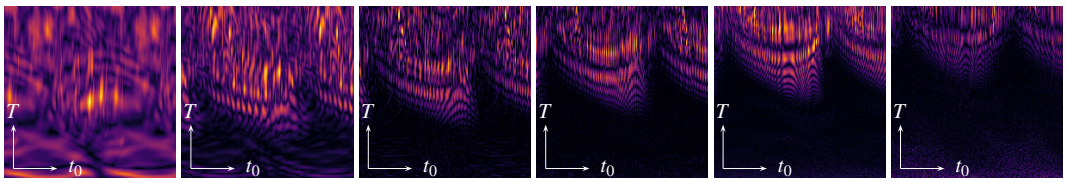
(a) (b)

Figure 4.3: Same dataset as Figures 4.1 and 4.2, showing the result (yellow) of the ridge extraction process with seeding time $t_0 = 4.0$ s and advection time $T = 2.75$ s (a), corresponding to the red dots in Figure 4.2, and $T = 6.25$ s (b), corresponding to the blue dots in Figure 4.2. The value of ρ_{len} in (a) is 5.09 , and 12.18 in (b). The aggregation fields show trends which would otherwise be impossible to observe just by looking at the single FTLE fields (higher values by higher saturation).

by LCS properties, we complement them with a new aggregation function that quantifies aliasing in a direct way.

Aliasing in FTLE fields, as apparent in Figure 2.8a, is caused by a too low FTLE field resolution compared to the gradients (frequencies) of the FTLE field. It is in particular the very low width of FTLE ridges that makes the FTLE field so hard to sample. This is all the more challenging as this width is related inversely to the advection duration T. That is, the larger T, the longer typically the LCS, and the sharper the FTLE ridges become. In confined domains, the flow has to turn at some point and will cause the ridges to fold, leading to closely adjacent ridges by the so-called stretching and folding mechanism [Sma67]. To the best of our knowledge, there is no analytic result on the relation between FTLE advection time and ridge width, but due to the underlying assumption of exponential divergence, and motivated by our observations, we assume that ridge width decreases exponentially, too, which imposes a very hard challenge on appropriate FTLE sampling. That is, increasing T typically requires much stronger increase of the spatial sampling resolution, otherwise aliasing will appear.

Motivated by the Nyquist–Shannon sampling theorem, we indicate aliasing by analyzing the frequency spectrum of the FTLE field. That is, we use the Fourier transform to obtain the 2D spectrum of the field, and measure the amplitude of the highest frequen-



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

Figure 4.4: Aliasing aggregation field $f_{v_{\text{sum}(2)}}$ for the Quad-Gyre dataset with resolution $G = 25 \times 25$ (a), 50×50 (b), 100×100 (c), 200×200 (d), 400×400 (e), and 800×800 (f). G_{agg} is, in all cases, 200 , $T \in [1,8]$, and $t_0 \in [0,8]$. Note the dependency of the aliasing aggregation field on advection time T, and how the aliasing pattern moves to larger T (top) as the field resolution G increases.

cies in that spectrum. Our motivation to do so is that if the original signal (in our case the underlying true FTLE field that we are discretizing using the discrete flow map) had frequencies higher than the Nyquist frequency, it would be very likely that the highest frequencies in the discretized signal (our sampled FTLE field) have non-negligible amplitudes. Conversely, negligible amplitudes in the highest frequencies in the FTLE can indicate appropriate sampling.

We realize such a quantification of the amplitude of the highest frequencies of an FTLE field $\sigma_{t_0}^T(\mathbf{x})$ by first transforming it to the 2D frequency domain $\tilde{\Omega} \subset \mathbb{R}^2$ using the fast Fourier transform [CT65], resulting in the spectrum $\tilde{\sigma}_{t_0}^T(\boldsymbol{\xi})$ for $\boldsymbol{\xi} \in \tilde{\Omega}$. Notice that in this representation, zero frequency is at the origin $\boldsymbol{\xi}_0 = \mathbf{0}$ and $\tilde{\Omega}$ is symmetric about the origin. As a consequence, points $\boldsymbol{\xi}$ at equal distance ν from the origin, i.e., $\|\boldsymbol{\xi}\| = \nu$, represent equal frequencies, or in other words, circles in $\tilde{\Omega} \subset \mathbb{R}^2$ (about the origin with radius ν) represent all amplitudes belonging to the frequency ν. Thus, to quantify the amplitude of the m highest percent of frequencies in the FTLE field $\sigma_{t_0}^T(\mathbf{x})$, we define the following aliasing aggregation function, which integrates the spectrum $\tilde{\sigma}_{t_0}^T(\boldsymbol{\xi})$ along these circles:

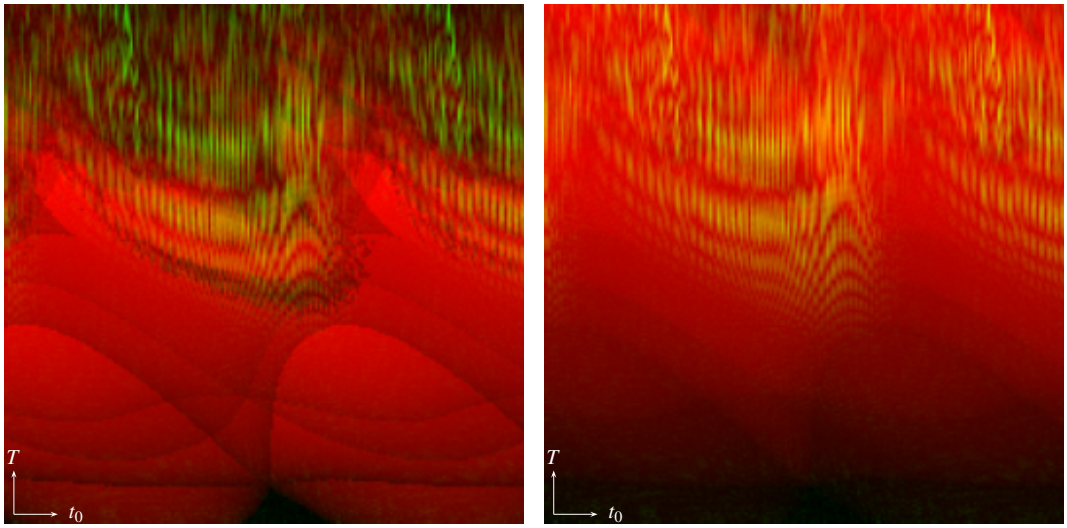
$$v_{\text{sum}(m)} := \iint_{\boldsymbol{\xi} \in \mathscr{X}} \tilde{\sigma}_{t_0}^T(\boldsymbol{\xi}) \, \mathrm{d}\boldsymbol{\xi} \ , \tag{4.9}$$

with

$$\mathscr{X} := \left\{ \boldsymbol{\xi}' \;\middle|\; \|\boldsymbol{\xi}'\| \geq \tfrac{100-m}{100} \hat{\nu} \right\} , \tag{4.10}$$

and with $\boldsymbol{\xi}' \in \tilde{\Omega}$, and $\hat{\nu}$ being the Nyquist frequency, i.e., the spatial resolution of our discretized $\sigma_{t_0}^T(\mathbf{x})$.

Figure 4.4 shows the resulting aliasing aggregation field $f_{v_{\text{sum}(m)}}(\mathbf{y})$ for choices of increasing FTLE resolution. One can see that as the FTLE resolution increases, the area within $\Omega_{t_0,T}$ with negligible highest frequencies (low aliasing) extends to the top, i.e., our aggregation indicates that longer advection times T can be used.



(a) (b)

Figure 4.5: Aliasing aggregation field $f_{v_{\text{sum}(2)}}$ mapped to the green channel, and $f_{\hat{\rho}_{\text{len}(k)}}$ (a), and $f_{\rho_{\text{len}}}$ (b) mapped to the red channel for the dataset from Figures 4.2a, 4.2c, and 4.4d. Note that in (a), the fields are somewhat complementary because the ridge quality decreases as the aliasing problems increase, whereas in (b), they both show the same increasing trend with T.

Typically, a correlation between the amount of aliasing and the ridge length aggregation $\rho_{\text{len}}(\cdot)$ can be observed. The main cause for this is that aliasing is typically caused by very sharp (thin) ridges, that cannot be resolved with the chosen spatial resolution. As discussed above, such long and sharp ridges appear in particular with longer advection times—and if the resolution is chosen too low, ridge extraction typically produces disrupted lines, which reflects in low $\hat{\rho}_{\text{len}(k)}$ values (top in Figure 4.2c). Our system provides the option to map aggregation fields to different color channels for combined analysis for such purposes, as demonstrated in Figure 4.5.

4.2.5 Region Aggregation Functions

Our final aggregation function goes one step further and focuses on the interpretation of FTLE fields. As discussed, FTLE ridges, the LCS, separate regions of qualitatively different time-dependent flow behavior. However, whereas separatrices in steady vector field topology indeed separate such regions entirely (assuming integration time of the streamlines goes to infinity), LCS typically do not. That is, LCS do generally not partition the domain—they typically leave gaps (see Figure 4.3b). The reason for that is that topology of aperiodic time-dependent flow does usually not have the opportunity to consider the limit case $T \to \infty$. Instead, finite-time considerations (with respect to finite advection

time T) have to be employed, leaving gaps between LCS due to the time-local nature of the topological processes.

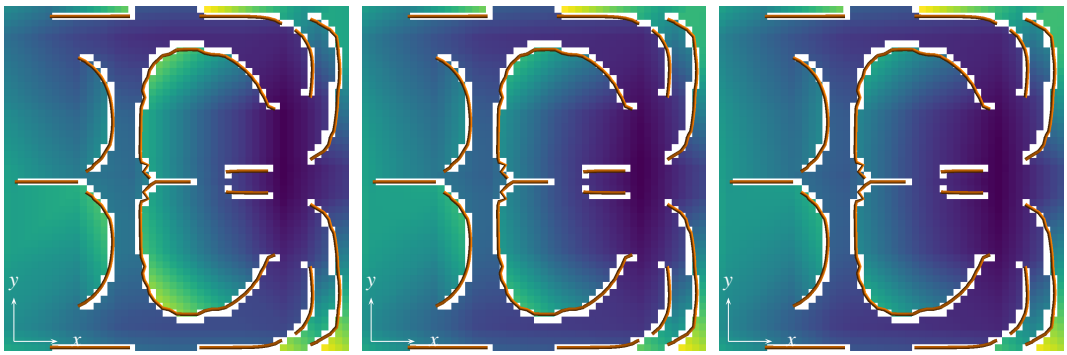
We account for these circumstances by introducing a measure to quantify the *connectedness* of the domain Ω with respect to "obstacles", which in our case are the FTLE ridges. Due to the high computational complexity of this measure, we define a separate (typically lower-resolution) sampling grid in the FTLE domain Ω, whose grid nodes we denote $\hat{\mathbf{y}} \in \mathscr{D}$ (Figure 4.6 contains a comparison of different $\hat{\mathbf{y}}$-resolutions). We then discretize the FTLE ridge set $\mathscr{R}$ at the resolution of the FTLE (at nodes $\hat{\mathbf{z}}$), by setting ridge nodes as "background" (white in Figure 4.6) and non-ridge nodes to "foreground". In other words, we binarize the ridge lines into the grid $\hat{\mathbf{z}}$, and collect its foreground nodes in a set $\mathscr{F}$ and its background nodes in a set $\mathscr{B}$. Based on that, we define our connectedness field $\gamma(\hat{\mathbf{z}})$ at a given node $\hat{\mathbf{z}}$ as follows:

$$\gamma(\hat{\mathbf{z}}) := \frac{1}{|\mathscr{F}|} \sum_{\hat{\mathbf{y}} \in \mathscr{D}} \mu(A^*(\hat{\mathbf{z}}, \hat{\mathbf{y}})) \,, \tag{4.11}$$

with $|\mathscr{F}|$ being the number of foreground, i.e., non-ridge nodes, $A^*(\hat{\mathbf{z}}, \hat{\mathbf{y}})$ being the shortest path between nodes $\hat{\mathbf{z}}$ and $\hat{\mathbf{y}}$ according to the A* algorithm (avoiding the background, i.e., ridge obstacles), and $\mu(\cdot)$ being the length of such a path. If such a shortest path between two foreground nodes $\hat{\mathbf{z}}$ and $\hat{\mathbf{y}}$ does not exist (e.g., because it is completely blocked by LCS and/or boundaries), the distance is set to the perimeter of the domain Ω. If one of the nodes (or both) are contained in $\mathscr{B}$, the respective length is set to zero. Figures 4.7c and 4.7d show an example of the connectedness field resulting from the respective ridge configurations (orange lines).

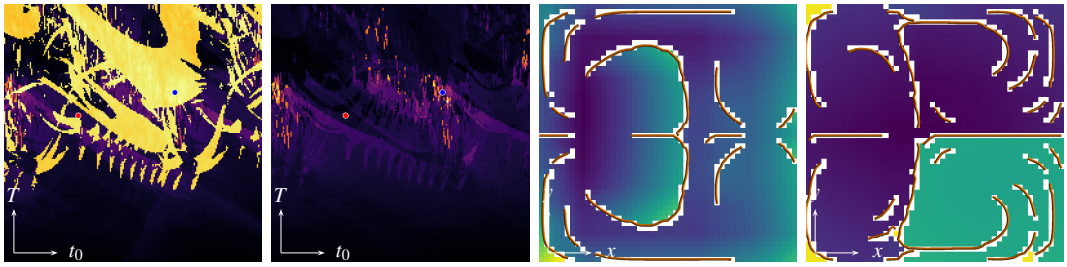
To obtain an aggregation field, we need to combine all values of the connectedness field $\gamma(\hat{\mathbf{z}})$ into a single scalar value. We accomplish this by employing the basic aggregation functions (Section 4.2.2) to $\gamma(\hat{\mathbf{z}})$ instead of $\sigma_{t_0}^{T}(\mathbf{x})$, resulting in the region aggregation fields $f_{\gamma_{\max}}(\mathbf{y})$ (Figure 4.7a) and $f_{\gamma_{\text{avg}}}(\mathbf{y})$ (Figure 4.7b).

Even though there is a correlation between the total ridge length aggregation field $f_{\rho_{\text{len}}}(\mathbf{y})$ and the connectedness aggregation field $f_{\gamma}(\mathbf{y})$ (Figure 4.8), the latter is able to account for almost isolated or even closed regions, in contrast. Additionally, $f_{\rho_{\text{len}}}(\mathbf{y})$ does not take into account the configuration and layout of the ridges in terms of topological structure.



(a) (b) (c)

Figure 4.6: The same connectedness field of the Quad-Gyre dataset with FTLE resolution $G = 50 \times 50$ and different $\hat{\mathbf{y}}$-resolutions $G_s = 5 \times 5$ (a), $G_s = 10 \times 10$ (b), and the same as G, $G_s = 50 \times 50$ (c). Observe the small difference between the version where $G_s = G$ (c) and the version using a hundred times less samples (a). This example shows that the connectedness of the dataset is equally well captured by all sampling resolutions.



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

Figure 4.7: Region aggregation fields (a)–(b) and connectedness fields (c)–(d) for the Quad-Gyre dataset. Resolutions are $G_{\text{agg}} = 200 \times 200$ and $G = 50 \times 50$. $f_{\gamma_{\text{avg}}}$ (b) shows a strong dependency on T, but also some changes along t_0, indicating different degrees of connectedness for different time settings. By contrast, $f_{\gamma_{\text{max}}}$ (a) clearly displays the fields which have closed areas in yellow. The red marker corresponds to the connectedness field (c), at $t_0 = 2.3$ s and $T = 5.0$ s, which exhibits no closed regions. On the other hand, the blue marker in (a) corresponds to (d), at $t_0 = 5.3$ s and $T = 5.6$ s, which has four isolated regions. In the connectedness fields, the ridge lines (orange) are rasterized in white.

4.3 System Description

The system is composed of a modular, interactive and customizable front-end and a versatile computing back-end, which enable the user to analyze, examine, and understand the temporal and spatial properties of the system under study. The software can be set up using command line arguments, to compute a user-selected list of aggregation fields, or it can load precomputed runs from files. Additional options include starting an interactive session or only compute in the background, and persisting the results to disk. The

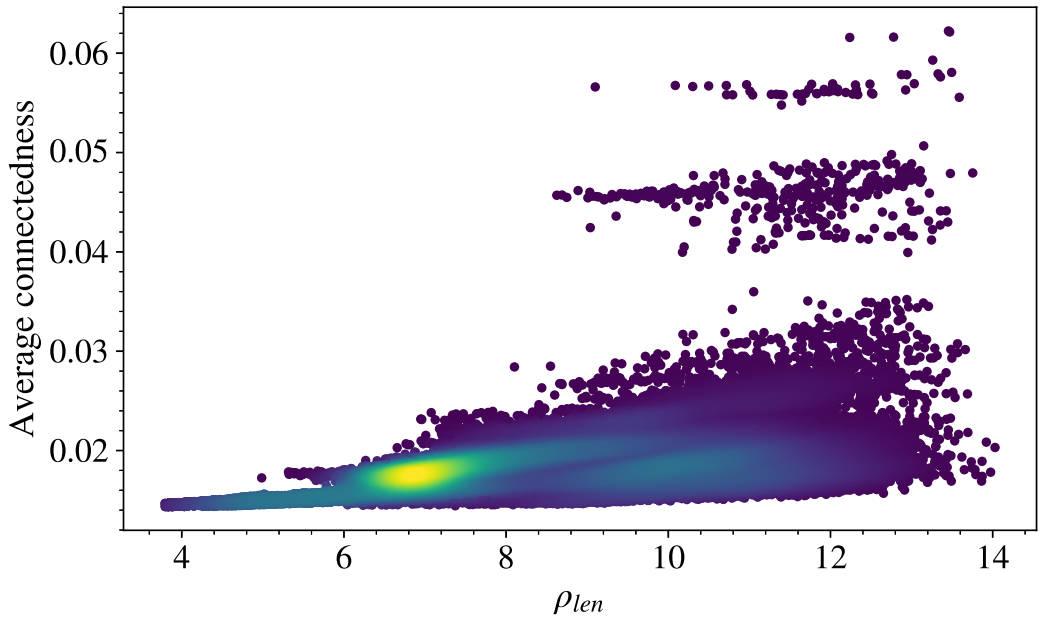


Figure 4.8: Scatterplot with density coloring of ρ_{len} (yellow = high), with respect to the average region aggregation field $f_{\gamma_{\text{avg}}}$, for the Quad-Gyre dataset. Note that for shorter ridge lengths, the relation is quasi-linear, but as the total ridge length increases, the values of the average region aggregation field spread out, giving rise to very different connectedness configurations for fields with the same total ridge length. The clusters centered around different average connectedness values (0.02, 0.045, 0.055) correspond to different closed-region configurations. For instance, the topmost cluster contains all configurations where the map is vertically divided into two similarly-sized disconnected regions. The second cluster corresponds to configurations where roughly a quarter of the map is a closed region (see Figure 4.7d). The rest corresponds to different configurations of smaller-sized closed regions. Note that within each cluster, there is a considerable deviation in average connectedness, proving that this aggregation function is actually able to measure the connectedness quite well, even when closed regions are present.

software will be open sourced and published at a later date. In the following, we describe its capabilities and features in detail.

Main layout. Our system consists of two panels laid out side-by-side (Figures 4.9 and 4.10), plus a set of UI elements and controls in charge of receiving and presenting information to the user. The left panel ① contains the aggregation field displaying one or more of the aggregation functions, optionally with logarithmic scale. The panel right next to it ② displays the FTLE field for a selected location in the aggregation space. The FTLE field panel is automatically updated as the user hovers the cursor over the aggregation field.

Aggregation panel. The aggregation panel ① contains the currently selected aggregation field with the user-selected resolution $G_{\text{agg}} = N_{t_0} \times N_T$. Two timelines are available to the left and bottom of the aggregation panel ③. These contain six "field icons" at different positions in t_0 (bottom) and T (left) space. The timelines are also automatically updated with the active mouse position in the aggregation field. Additionally, we display integrated hyperbolicity $\det(\nabla \mathbf{u}(\mathbf{x}))$ of the vector field (which tends to quantify the separating dynamics causing LCS) aligned with the seeding time on the horizontal axis, below the timeline ④. A color legend is provided to the right of the aggregation panel ⑤, and it is updated automatically when the aggregation field changes.

FTLE panel. The FTLE panel ② displays the finite-time Lyapunov exponent field for the active coordinate in the aggregation panel, with the user-selected resolution $G = N \times M$. It is recomputed and updated automatically whenever this coordinate changes. Moving the pointer within the FTLE panel, while pressing the control key, spawns a trajectory seed, which follows the pointer movement. Clicking on the field while pressing the control key creates permanent trajectory seeds. All used colormap schemes can be changed at runtime using the *View* menu. The system allows for global and local scaling of colormaps. All images in this chapter use local scaling. The colormaps used by default include the *blues* colormap (white to blue, used in regular FTLE fields), the *inferno* colormap (black to purple to orange to yellow, used in the aggregation fields) and the *viridis* colormap (purple to blue to green to yellow, used in the region aggregation fields).

Contours and ridges. Similarity contours can be computed in the aggregation panel in order to aid in the visual exploration of the aggregation field. Ridge extraction can be applied to either the aggregation or the FTLE fields. The ridge extraction threshold (τ_λ) defaults to the one used for the computation of the ridge aggregation fields. The system also supports interactive exploration of different values for τ_λ.

Three-way exploration. The system supports a three-way simultaneous exploration mode (Figure 4.5) for aggregation fields. Any two or three aggregation fields may be mapped to the red, green, and blue channels, in order to explore their combined properties. If the desired exploration concerns only two aggregation fields, a default 'zero' aggregation field is provided, ready to be selected and mapped to any color channel to mute it. More interestingly, a three-way exploration mechanism is also offered in the FTLE field panel. Once enabled, the user can successively click on positions in the aggregation panel to map the corresponding fields to the red, green, and blue channels. The mapping se-

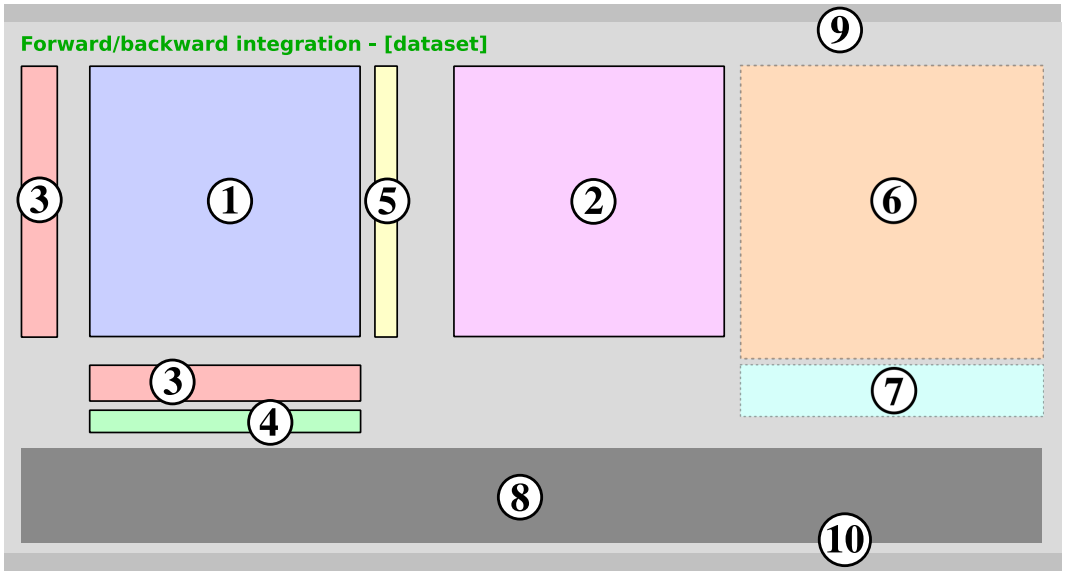


Figure 4.9: Illustration of the components in our system. Aggregation panel ①, FTLE panel ②, vertical and horizontal timelines ③, vector field hyperbolicity panel ④, aggregation field color legend ⑤, UI controls ⑥, contextual information pane ⑦, output console ⑧, main menu ⑨, and status bar ⑩.

quence starts over with the fourth click. Any trajectory seeds present in the field view are updated using the coordinates last FTLE field computed by the system.

Caching capabilities. The aggregators are computed once per dataset and set of initial parameters and persisted to a file. The fields are typically computed in real time, especially when their computational complexity is low enough that the available hardware can perform the task in an interactive manner. For the cases in which that is not possible, fields, ridges and other elements can be cached during the computing phase and persisted along with the aggregators in order to enable fast exploration and analysis. This is, in fact, the default mode when distance maps are enabled, as they are far too costly to be computed in real time during an interactive session. Along with the aggregators and possibly the fields and ridges, the full state of the application is also persisted, containing all the starting parameters and options.

Input/output controls. The controls are positioned to the right of the FTLE panel ⑥, and contain the necessary elements to manage the behavior and settings of the current session. In order to guide the user through the visual analysis process and improve understanding, the system offers a contextual information box ⑦, which shows comprehensive information of the currently selected/highlighted elements. The current output log is

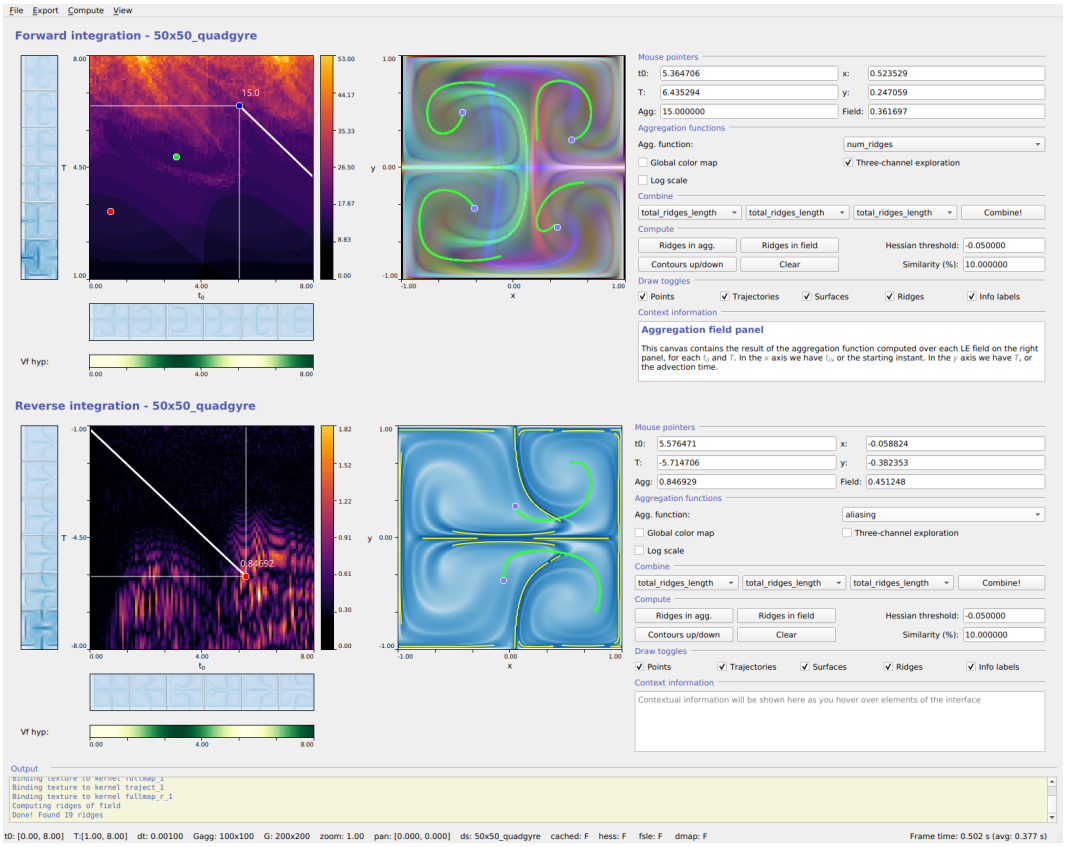


Figure 4.10: Screen shot with the layout and appearance of our framework. The user interface is divided into three main vertical areas: the forward integration, the reverse integration and the console output.

shown in the console output panel ⑧. The top menu bar ⑨ contains shortcuts to many functions, such as saving the current session, exporting views to image files, changing color maps or pausing the update thread. Finally, a summary of the state of the current session is displayed at the bottom of the window, in the status bar ⑩. The resolution of the aggregation and FTLE fields, the extents of t_0 and T, the caching status, and some global state flags are displayed here.

Contextual information box. The information is updated live as the user hovers over different UI elements, and offers straightforward notes on what the element is and does, as well as tips on how to use it to maximize its usefulness during the exploratory process.

Export functions. In order to produce publication-quality visuals, the system offers a handy export feature which is able to export the contents of any panel to images in PNG

and EPS formats. The export process renders the different layers (surface, primitive elements, text, etc.) of the selected view to high-resolution off-screen buffers and produces high-quality files. Additionally, it also blends all the layers together to produce the final high- and low-resolution images of the scene. Sometimes, some aggregation fields display strong dependencies on either t_0 or T. In order to aid in the analysis of such cases, projection profile plots of either aggregation field axes for any aggregation function can be automatically computed and exported to files as well.

4.4 Results

Our recommended exploration flow, as gathered by the experts' experience with the tool, starts by analyzing the basic aggregation in order to get a sense of the general trends that may be captured by the fast statistical functions, followed by the ridge and aliasing aggregation fields, which help narrow down the most interesting parameter subspaces, to end with the region aggregation to gain insight into the connectedness properties of the dataset. The system allows, if required, interactive exploration of the FTLE fields in context with the aggregation space, as well as interactive fine-tuning of key parameters such as the ridge extraction threshold. Additionally, recomputation of interesting sub-spaces at different resolutions and with different parameters may also be required. With these guidelines in mind, we analyze an analytic example (Section 4.4.1), a 2D fluid dynamics simulation (Section 4.4.2), a constructed dataset based on appearing Gaussian vortices (Section 4.4.3), an inertial N-body system (Section 4.4.4), and an atmospheric wind dataset based on real measurements (Section 4.4.5).

4.4.1 Quad-Gyre

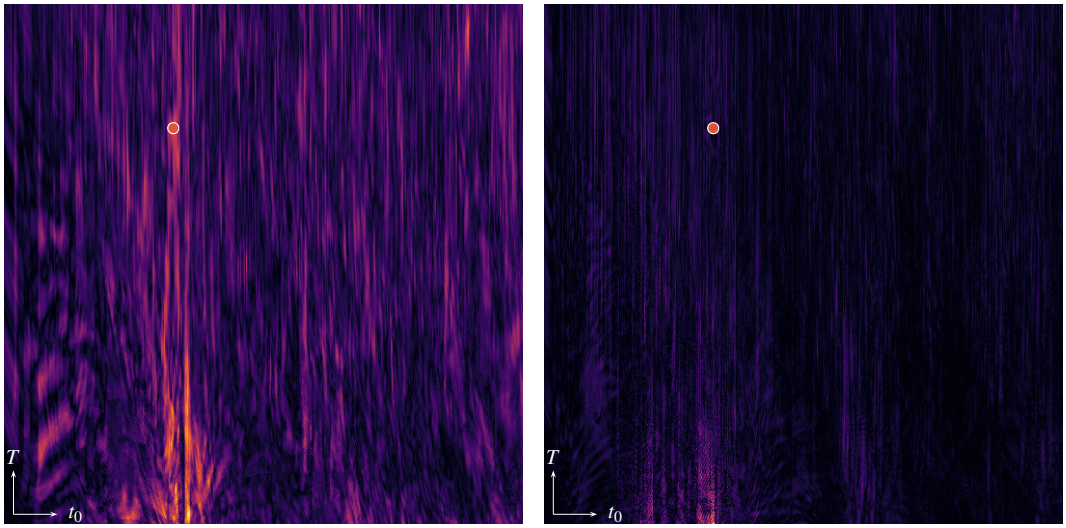
We have already used this example in Section 3.3.1. It was introduced by Shadden et al. [SLM05] for the analysis of Lagrangian coherent structures. The vector field is defined by the analytic representation shown in Equation 3.32. We sampled the field at a resolution of 50×50 and 100 time steps in the spatial range $\mathbf{x} \in [-1,1] \times [-1,1]$ and temporal range $[0,20]$. Note that we use this y-range instead of the often employed $[0,1]$, leading to a y-symmetric field with four time-dependent vortices instead of two.

We compute all basic, ridge, aliasing, and region aggregation fields for $T \in [1,8]$ and $t_0 \in [0,8]$. Starting with the basic aggregation fields in Figure 4.1, we can see in $f_{\sigma_{\text{avg}}}$ (Figure 4.1b) very vague recurring features in t_0 and a strong dependency on T. This periodicity is captured much better by $f_{\sigma_{\text{max}}}$ (Figure 4.1a). A particular feature of this dataset is that when the main vertical LCS crosses from right to left and vice-versa, it is

less sharp, resulting in lower FTLE values. This is captured by $f_{\sigma_{\max}}$ and shown as a valley. $f_{\rho_{\text{len}}}$ shows an obvious proportionality with T (Figure 4.2a), and some extra periodic structure that shows variations in the total length of ridges. Figure 4.3 shows the actual FTLE field and ridges with two different advection times for this dataset. $f_{\hat{\rho}_{\text{len}(k)}}$ (Figure 4.2c) shows that the central values of T (between 3 s and 5 s) are the best candidates for high-quality ridges. An interactive exploration with the system reveals that at higher T values, the ridges break up, and the current resolution is not able to capture them well. The aliasing aggregation field supports this analysis. Figure 4.4a shows that the field resolution of 25×25 is unable to properly sample the field anywhere in the aggregation space, whereas Figure 4.4e shows that the resolution of 400×400 does not present problems when $T < 4$ s. In particular, looking at the aliasing field for the resolution of 200×200 (Figure 4.4d) for which $f_{\hat{\rho}_{\text{len}(k)}}$ has been computed (Figure 4.2c), we discover that areas with lower $f_{\hat{\rho}_{\text{len}(k)}}$ values (i.e., low-quality ridges) coincide with regions with high aliasing indication. Figure 4.5a shows both aggregation fields in the same image for comparison. We determine that we would need a higher resolution to effectively capture the ridges in that area. Finally, the maximum region aggregation field (Figure 4.7a) is able to separate the configurations which have disconnected areas from the rest and provides a clear picture of the degree of connectedness of the fields in the aggregation space.

4.4.2 Buoyant Flow

The Buoyant Flow dataset is a fluid dynamics simulation of air current moving around in a container with a heated bottom wall and a cooled top wall. Figure 4.12 shows basic and ridge aggregation fields for this dataset. We can see in $f_{\rho_{\text{len}}}$ (Figure 4.12b) that most of the structure is concentrated away from low t_0 and T values, but once it shows up, it exhibits little variation. We can infer that the Lagrangian coherent structures, represented by the ridges, move around with the field but the total number stays more or less constant, which indicates their high quality. Figure 4.11 contains a comparative analysis of the aliasing aggregation field with two different resolutions, 100×100 (Figure 4.11a) and 400×400 (Figure 4.11b). Observe how the aliasing problems diminish considerably with this factor of four increase in field resolution. We can be quite confident that the field is sampled correctly across the whole aggregation space, except for a very small problematic zone close to the bottom left. Figure 4.13 shows the Buoyant Flow FTLE with varying resolutions and identical τ_λ. Our system, as demonstrated in the accompanying video, supports interactive exploration of such parameters.



(a) (b)

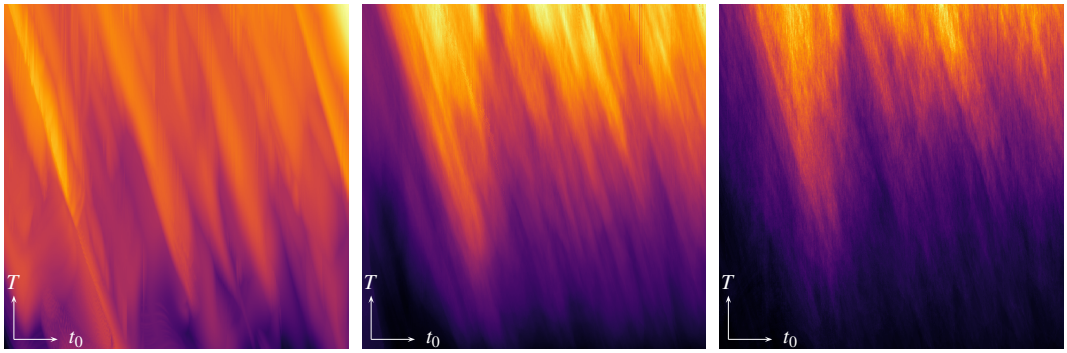
Figure 4.11: Aliasing aggregation fields ($G_{\text{agg}} = 500 \times 500$) for the Buoyant Flow dataset with $G = 100 \times 100$ (a) and $G = 400 \times 400$ (b). Observe how the aliasing values decrease significantly with the increased resolution.

4.4.3 Three Gaussian Vortices

The Three Gaussian Vortices dataset is a constructed vector field based on a blend of simpler fields generated using a mixture of programmatic rules and time-dependent analytical functions. It consists of three vortices, each spatially masked with a Gaussian function, that appear at different times and positions on top of a uniform velocity field to the right. The basic aggregation fields $f_{\sigma_{\text{max}}}$ (Figure 4.14a) and $f_{\sigma_{\text{avg}}}$ (Figure 4.14b) capture the general trend quite well. $f_{\sigma_{\text{avg}}}$ captures the moment when each vortex appears. Figure 4.14c shows that $f_{\rho_{\text{len}}}$ additionally captures the location of the vortices in the aggregation space in terms of ridges, providing a very good picture of interesting parameter combinations for this dataset. To that effect, the illuminated areas represent the subspaces of (t_0, T) where one, two, and three vortices are in the field and cause sufficient sharpness in the FTLE field so that the ridge extraction algorithm succeeds.

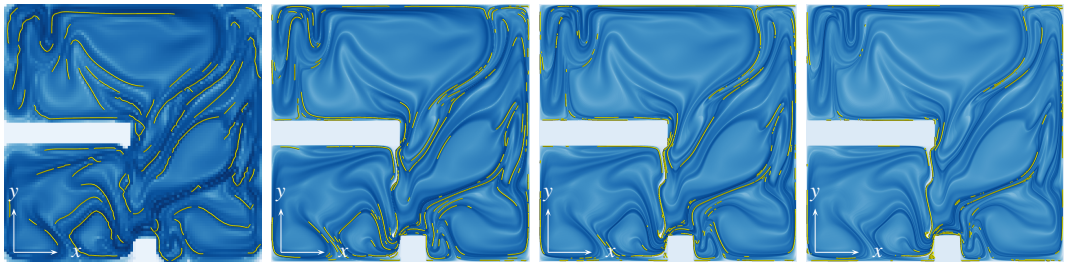
4.4.4 5-Body

We now demonstrate the utility of our approach to handle fields other than traditional FTLE. In this example, we use an inertial N-body system with five bodies. The simulated bodies start more or less evenly distributed in space and interact freely over time (Figure 4.17). We use the phase-space finite-time Lyapunov exponent field (PS-FTLE) introduced in Section 3.1.3, which extends the regular FTLE to the dynamics of inertial



(a) (b) (c)

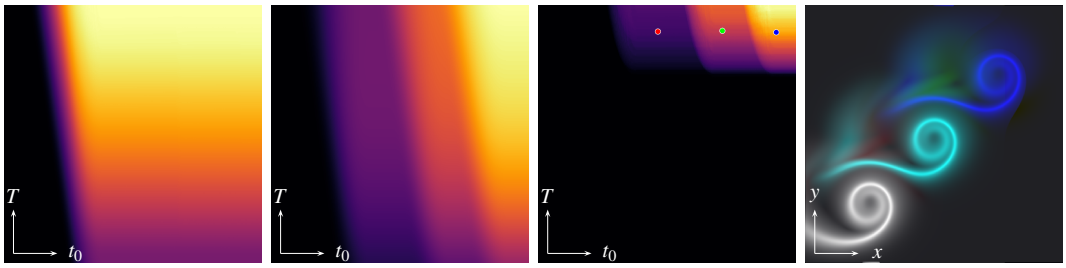
Figure 4.12: Buoyant Flow dataset with $G_{\text{agg}} = 500 \times 500$ and $G = 400 \times 400$, $t_0 \in [0, 0.3]$, and $T \in [0.01, 0.1]$. All fields $f_{\sigma_{\max}}$ (a), $f_{\rho_{\text{len}}}$ (b), and $f_{\rho_{\text{cnt}}}$ (c) show a similar structure with this dataset. The ridge aggregation fields, however, show that the LCS are only sharp enough to be detected as ridges at high T values.



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

Figure 4.13: FTLE fields extracted from the Buoyant Flow dataset with $G = 100 \times 100$ (a), $G = 400 \times 400$ (b), $G = 800 \times 800$ (c), and $G = 1600 \times 1600$ (d) with identical $t_0 = 0.09$ s, $T = 0.088$ s (red marker in Figure 4.11), and $\tau_\lambda = -0.04$ in all cases. Observe how the same τ_λ value leads to very different ridge configurations, depending on the resolution.

systems, with initial velocity constrained to $\mathbf{0}$ (PS-FTLE-P). The aggregation fields (Figure 4.16) exhibit more structure than most previous datasets. $f_{\sigma_{\max}}$ and $f_{\rho_{\text{len}}}$ clearly show the dispersion of the bodies toward higher t_0, i.e., the bodies exit the region of study toward the end of the time range. Figure 4.17 provides a PS-FTLE-P sequence for this dataset. The aliasing aggregation fields for $G = 100 \times 100$ (Figure 4.16c) and $G = 300 \times 300$ (Figure 4.16d) show that, even though the aliasing decreases considerably in some regions, a resolution of 300 is still insufficient. The PS-FTLE-P folds over and over close to the bodies, which the shown resolution is not able to capture appropriately. The region aggregation fields for this dataset (Figure 4.15) hint at configurations at which the ridges isolate disconnected regions (yellow zones in Figure 4.15b), and configurations where the ridges do not quite close at the given resolution and thus produce lower connectedness values.



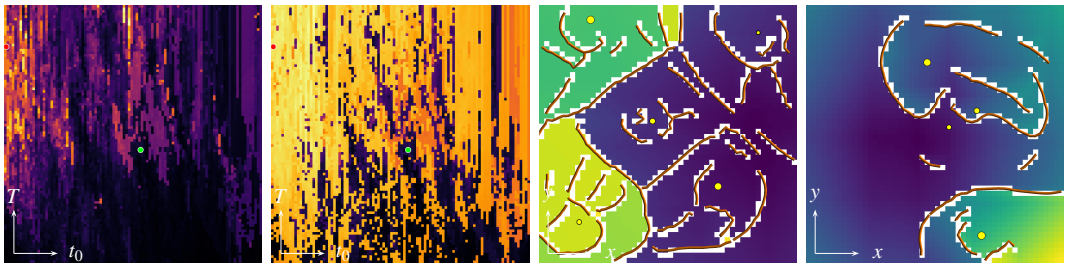
(a) (b) (c) (d)

Figure 4.14: Three Gaussian Vortices dataset with $G_{\text{agg}} = G = 400 \times 400$, $t_0 \in [40, 170]$, $T \in [1, 20]$, and $\tau_\lambda = -0.04$ $f_{\sigma_{\text{max}}}$ (a) captures the maximum value in the FTLE field, only showing the boundary where the first vortex appears, while $f_{\sigma_{\text{avg}}}$ (b) captures the average of all values, clearly displaying additional structure as each of the three vortices appears. $f_{\rho_{\text{len}}}$ (c) shows the contribution of each of the three vortices, as they appear, in the form of ridge length. The red, green, and blue markers correspond to the same $T = 18$ s and different t_0 of 100 s, 133 s, and 160 s, respectively. (d) shows the three FTLE fields corresponding to the three points in (c), mapped to the red, green, and blue channels. The bottom vortex appears at around $t_0 = 75$ s and stays there until the end, thus showing white (red + green + blue). The middle vortex is still present when the top-most one appears, thus showing in cyan (green + blue).

Stefan Jordan, one of our coauthors and an expert astrophysicist, has evaluated the tool with the 5-Body dataset: The system is able to guide to some interesting events like close-body interactions around $t_0 = 1.6$, showing up as a valley in the maximum and ridge length aggregation fields. Even though, to the best of our knowledge, FTLE has never been used in astronomy, the identification of relevant events and features with such a tool could aid in the interpretation and understanding of theoretical astronomical data like N-body and hydrodynamic simulations, the merging of galaxies, the dissolution of star clusters, star and planetary system formation and, convection in stars.

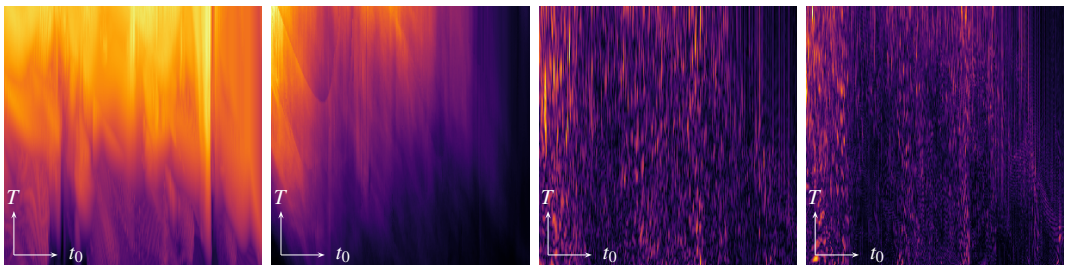
4.4.5 Atmospheric Wind

This dataset was generated by the Copernicus Climate Change Service (2020) and contains observations of wind velocity in the area of longitudes between 30° and 60° east and latitudes between −15° and 15°, corresponding to the region around the horn of Africa, for December 1 to 3, 2019. This is a rather complex dataset, but it is easy to identify the day–night cycles by investigating the aggregation fields in Figure 4.18. The maximum aggregation field Figure 4.18a proves to be the poorest at capturing it, but it shows some interesting additional structure. However, both $f_{\sigma_{\text{avg}}}$ (Figure 4.18b) and $f_{\hat{\rho}_{\text{len}(k)}}$ (Figure 4.18c) show that periodic trend, with the latter revealing the nights as times of higher activity with more pronounced ridges.



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

Figure 4.15: Average (a) and maximum (b) region aggregation fields for the 5-Body dataset with $G_{\text{agg}} = 100 \times 100$ and $G = 50 \times 50$, together with the actual connectedness fields at $t_0 = 0.057$ s, $T = 2.68$ s (c), and $t_0 = 3.44$ s, $T = 1.87$ s (d), corresponding to the red and green dots in the aggregation fields, respectively.

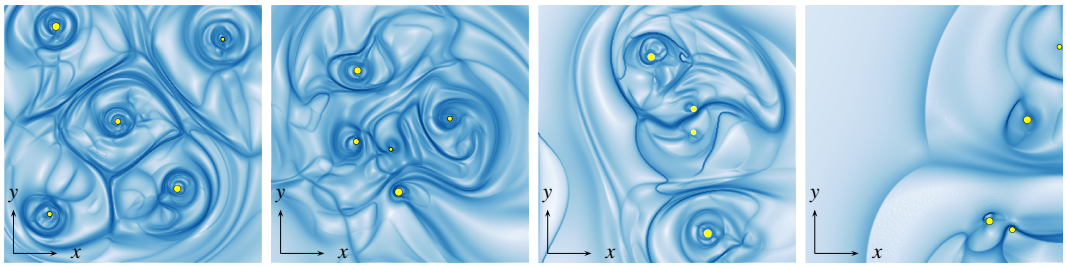


(a) (b) (c) (d)

Figure 4.16: Aggregation fields for the 5-Body dataset. $f_{\sigma_{\max}}$ (a), $f_{\rho_{\text{len}}}$ (b) with $G = 300 \times 300$, and aliasing aggregation fields with $G = 100$ (c) and $G = 300 \times 300$ (d). $G_{\text{agg}} = 300 \times 300$, $t_0 \in [0, 6.5]$, and $T \in [1, 3]$ in all cases. The aliasing aggregation fields suggest that the chosen resolution is too low for this dataset, irrespective of (reasonable) advection times.

4.5 Implementation and Performance

The reference application is implemented using Python and the Qt5 GUI library via PyQt5. Much of the computationally intensive operations ((PS-)FTLE fields, trajectories (or pathlines), connectedness, ridges, reverse integration, etc.) are implemented in highly parallel CUDA kernels and managed via PyCuda [Klö+13]. These are laid out and structured using a tailor-made templating system that is able to construct and compile the needed kernel on-demand from a large set of templates. The templating system produces 67 different single kernels, amounting to 20 thousand lines of CUDA code in total. The graphical representation in the panels is handled by the OpenGL abstraction layer, provided by the library VisPy [Vis], and extra care was put into implementing efficient and speedy updating and streaming solutions for all graphical components. The project has 11540 lines of code in total, 8421 of which are in Python, 2580 in CUDA, 179 in Bash scripts, 220 in GLSL, and 140 in HTML.



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

Figure 4.17: PS-FTLE-P fields for the 5-Body dataset with $G = 300 \times 300$, $T = 2.8$ s, and $t_0 = 0.25$ s (a), $t_0 = 2.25$ s (b), $t_0 = 4.25$ s (c), and $t_0 = 6.26$ s (d). The radius of the bodies (yellow dots) is proportional to the mass. The five bodies start evenly spread, with different initial velocities, then interact with each other and disappear to the right.

4.5.1 Performance Analysis

The computational complexity of this project resides mainly in the computation of the field matrix in the aggregation space. The grid resolution of both the aggregation field (G_{agg}) and the FTLE fields (G), plus the current advection duration T_i and integration step Δt_i, define the number of operations to compute to obtain the flow map as $N_{t_0} \cdot N_T \cdot N \cdot M \cdot (T_i/\Delta t_i)$. The interactivity performance analysis (Table 4.1) shows that the FTLE field frame time is proportional to the number of samples, but the sample time actually decreases slightly as the resolution increases. This may hint at a better GPU occupancy rate with higher resolutions, thus leading to lower sample times.

The precomputation performance analysis (Table 4.2) measures the computation of the aggregation fields. We run different aggregation field resolutions of the same dataset with three different cumulative aggregation function sets: the basic and aliasing aggre-

Table 4.1: Interactivity performance results with the Quad-Gyre dataset in average frame time, and sample time for different G values, and fixed $\Delta t = 0.001$ s and $T = 4.0$ s. The 'Steps' column refers to the integration steps per seed (i.e., $T/\Delta t$). Timings in seconds.

G	Steps	Samples	Frame time	Sample time
50×50		2 500	0.038	$0.152 \cdot 10^{-4}$
100×100		10 000	0.108	$0.108 \cdot 10^{-4}$
200×200	4 000	40 000	0.364	$0.091 \cdot 10^{-4}$
400×400		160 000	1.281	$0.080 \cdot 10^{-4}$
800×800		640 000	4.988	$0.078 \cdot 10^{-4}$

gation functions (B), the ridge aggregation functions (R), and the region aggregation functions (D). Each set contains the previous, as the ridge extraction is a prerequisite for computing the connectedness, and the basic aggregation fields are always computed. The ridge extraction and connectedness operations are done per field, without batching (see Section 4.5.2), and that reflects in their cost, with increments of a few hundreds percent for ridge extraction, and tens of thousands percent for region aggregation.

4.5.2 Optimization

Optimizations have been implemented in order to maximize parallelism on the GPU, improve core occupancy, and minimize CPU synchronization points. In order to do so, we batch the T dimension of the aggregation space into the main kernels, so that each computes N_T FTLE fields. We found that batching T is the most beneficial, because we can significantly shorten the computation time of the flow map by choosing an integration step Δt, N_T, and T extent $[T_0, T_1]$, so that $(T_1 - T_0)/N_T$ is divisible by Δt. This allows us to compute the flow map only once for the highest T setting (i.e., the top of the aggregation field) and sample it for the rest.

Other, more straightforward, optimization strategies include full GPU implementations of A* and ridge extraction. The resolution at which the connectedness fields can be computed is limited by per-thread memory constraints. We found that connectedness fields more than $G_s = 50 \times 50$ could not be computed on the GPU with the available hardware (Nvidia GTX 970 and 1070).

In order to speed up the processing of these very computationally expensive distance maps, we do not compute all possible paths from each seed point to every other seed point

Table 4.2: Pre-computation performance results with the Quad-Gyre dataset in total compute time for different aggregation field resolutions, and using only the basic and aliasing aggregation functions (B), additionally the ridge aggregation functions (R), and additionally the region aggregation functions (D). The fixed parameters for this analysis are $G = 50 \times 50$, $G_s = 10 \times 10$, $t_0 \in [0, 8]$, and $T \in [1, 8]$. All timings are in seconds.

G_{agg}	B	B+R	B+R+D
5×5	0.5	0.7 (+47%)	348.6 (+45651%)
50×50	11.9	40.1 (+236%)	29 990.2 (+74508%)
100×100	39.6	149.5 (+276%)	135 627.0 (+90610%)
200×200	142.3	590.3 (+314%)	480 310.5 (+81261%)

in the map, but we introduce an extra parameter, the seed grid size G_s. G_s is the size of a regular grid that we lay out on top of the field to define the end points of the paths. So, if we are computing the connectedness function of resolution $G = 50 \times 50$, with no walls, and a seed grid size of $G_s = 50 \times 50$, we need to compute $50^2 * 49^2 = 6\,002\,500$ A* paths. Using a seed grid size of $G_s = 5 \times 5$ we only need 62 500 A* computations, while the results are very close to the case where we use a full seed grid size (Figure 4.6).

4.6 Discussion

In this chapter, we presented a novel approach to the visual analysis of finite-time Lyapunov exponent based flow visualization, and we have demonstrated its usefulness with a variety of simulated and analytical datasets. We have introduced a set of aggregation functions that are able to capture different aspects of the underlying fields. We have found that the basic aggregation captures general trends very well, while the aliasing aggregation helps determine whether the discretization is sufficient, especially when the LCS are sharp. The ridge aggregation is able to identify areas with high-quality ridges and the region aggregation helps assess the topological "connectedness" of the dataset under study.

We leave as future work the determination of an approximate optimal resolution by means of heuristics that use information from the aliasing and ridge aggregation functions. Also, we plan to add more aggregation functions and extend the current ones, possibly with extra dimensions. A three-dimensional ridge aggregation field with τ_λ as an additional dimension could prove useful at analyzing the ridge extraction threshold subspace. Another obvious candidate for an additional dimension is the FTLE resolution G, but that would imply deep changes in the current precomputation engine.

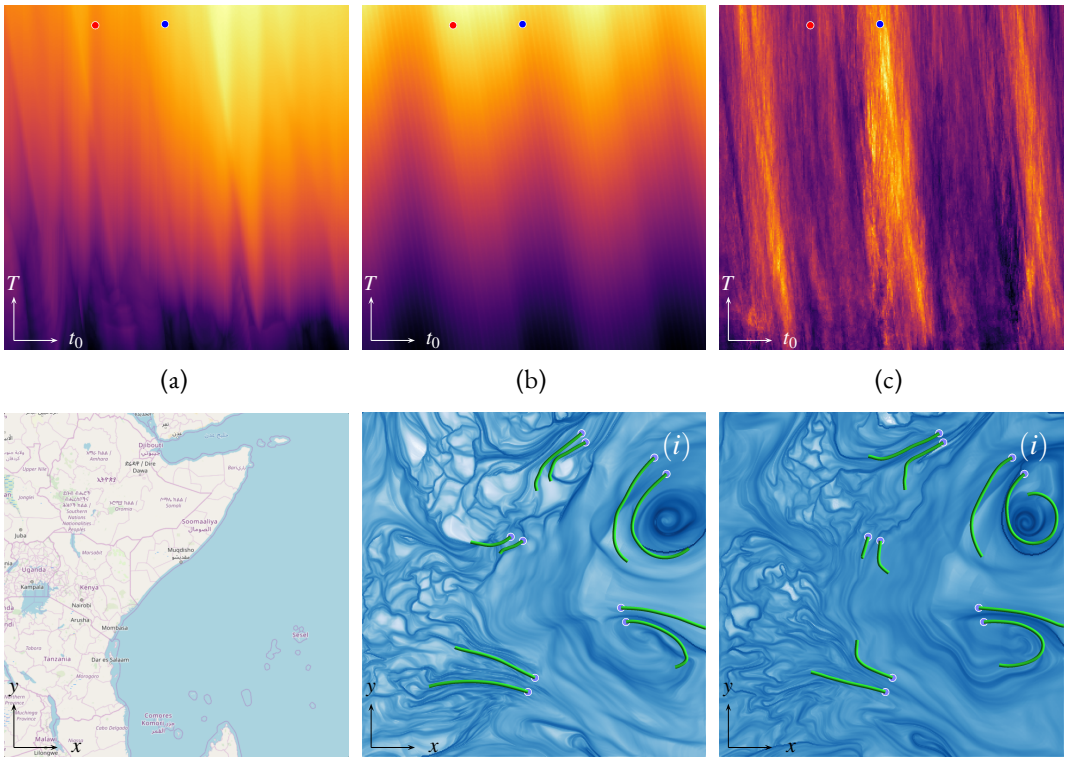


Figure 4.18: The Atmospheric Wind dataset represents wind data of a region close to the eastern coast of Africa (d). We apply our system with $G_{\text{agg}} = 300 \times 300$ and $G = 400 \times 400$, $t_0 \in [0, 60]$, $T \in [1, 10]$, and τ_λ =−0.08 $f_{\sigma_{\max}}$ (a) shows maximum FTLE values increasing with t_0 and T, with some additional substructure caused by the ever-changing wind patterns, while $f_{\sigma_{\text{avg}}}$ (b) reveals periodic structure. $f_{\hat{\rho}_{\text{len}(k)}}$ (c) exhibits three vertical regions with high quality ridges. The FTLE fields are captured at the same $T = 9.5$ h and different $t_0 = 15.8$ h (e), and $t_0$0 28 h (f), corresponding to the red and blue markers in the aggregation fields, respectively. We seed pairs of trajectories at close locations to aid exploration. For instance, the ridge (i), not present in (e), appears in (f) (see the lower seed spiraling into the vortex causing a larger separation). Other trajectories also reveal greater separation in (f). Interactive exploration reveals that this trend is maintained along the bright bands in (c).

The strongest affection and utmost zeal should promote the studies concerned with the most beautiful objects. This is the discipline that deals with the universe's divine revolutions, the stars' motions, sizes, distances, risings and settings.

— Nicolaus Coepernicus

5 Robust Extraction of Lagrangian Coherent Structures

In the previous chapter, we have developed a methodology to visually analyze time-dependent flow in the context of the FTLE. We have based one of the aggregation functions on the LCS, which are obtained as height ridges extracted from the field. However, this height ridge extraction process presents several issues, and is typically a quite challenging and involved task, usually necessitating of a considerable amount of trial and error work due to the many parameters involved. In this chapter, we present an approach to quantify the extraction of LCS as height ridges in the FTLE and introduce a metric-based method to select LCS visualization parameters.

Various approaches have been presented for extracting these separating structures in time-dependent vector fields, and a multitude of variants has been derived that enables the analysis of a wide range of applications. The major part of these approaches captures LCS by means of extracting locally maximizing manifolds (ridges) in derived scalar fields, such as the FTLE. These fields, in turn, are typically computed by means of the flow map, a mapping from seed points of trajectories to their respective endpoints. A common issue in feature extraction, however, is the difficulty of adjusting the parameters of the extraction technique. In the case of Lagrangian coherent structures by ridge extraction from derived scalar fields, there are two parameters that need to be determined for the flow map, i.e., the advection duration (i) for the trajectories and the spatial resolution (ii) of the seed points, followed by at least one parameter for suppressing weak and erroneous ridge parts by means of a threshold (iii).

Adjusting three parameters is already a difficult undertaking if the features are defined locally and have a clear interpretation, such as in the case of, e.g., edge detection in image processing. However, in the case of Lagrangian coherent structures, this task is incompa-

rably more demanding due to the non-local effect of these parameters, their intertwined interrelation, and mutual impact on properties such as:

(a) time scale, related to parameter (i),

(b) aliasing, related to parameters (i) and (ii), and

(c) cross-flux, related to parameters (i), (ii), and (iii).

An increase of the trajectory advection duration (i) provides a larger time scale (a) for analysis, and ridge regions that are sharper and exhibit lower cross-flux (c), and thus better represent LCS. However, at the same time, this causes more and longer ridges that tend to clutter, which leads, together with their increased sharpness, to aliasing (b), and thus requires higher resolution (ii) of the flow map. That is, one needs to balance advection duration vs. resolution, taking into account expressiveness in terms of sufficiently low cross-flux. An additional constraint for this balancing is computational cost, since each sample of the flow map requires a costly trajectory integration, and thus the total cost scales linearly with the number of samples and advection duration.

Given that a suitable advection time together with a respective resolution have been determined, a remaining problem is that not all parts of the resulting ridges have same expressiveness, i.e., one needs to filter (iii), i.e., reject, ridge parts that exhibit a too high cross-flux and thus do not act as separating LCS.

The determination of a suitable advection time together with a respective resolution, and the filtering of the ridges, have been accomplished until now in a visually-guided iterative process. That is, one typically determines the advection duration, the resolution, and the filtering threshold(s) in an intertwined trial-and-error manner, guided by the required computational cost, the desired time scope of investigation, and by visual assessment of the "quality" of the resulting ridges. The filtering has been mainly accomplished by employing a visually determined threshold that rejects ridge regions which exhibit a too low FTLE value or insufficient sharpness.

Obviously, this procedure is not clearly defined, requires substantial experience to apply and interpret, provides no quantitative guarantees, is hard to reproduce, and is—last but not least—tedious. This altogether might be responsible why LCS, although regarded a useful tool, did not yet reach wide application in science and engineering.

In this chapter, we present an approach that, among other measures, quantifies (a), (b), and (c), takes quantitative user inputs, and provides automatic selection of the parameters (ii) and (iii) in a quantitative, reproducible, effective, and easy-to-interpret manner.

The contributions in this chapter include:

- quantitative measures to asses the quality of LCS visualizations,
- metric-based selection of LCS visualization parameters,

- a quantitative ridge filtering approach based on cross-flux, and
- as a consequence, reliable and easy-to-use LCS visualization.

5.1 Basic Concepts

An introduction to LCS visualization by means of ridge extraction from derived fields is provided in Section 2.5. Their relation to cross-flux is discussed in detail in Section 5.1.1 below. The FTLE itself is introduced in Section 2.5.5, and its interconnection with resolution is explored in Section 5.1.2. Height ridges are discussed in depth in Section 2.5.7. This provides the basis for the technique developed in this chapter, detailed in Section 5.2.

5.1.1 Cross-Flux

Whereas the classical LE (Section 2.5.4) was primarily introduced as a measure for predictability, i.e., to quantify the butterfly effect in terms of error growth, the FTLE has become a valuable tool for topological analysis of time-dependent vector fields. Haller [Hal01] proposed and Shadden et al. [SLM05] defined ridges (Section 2.5.7) in the FTLE field to represent LCS. In an n-dimensional continuous time-dependent vector field $\mathbf{u}(\mathbf{x},t)$ in a domain $\Omega \subseteq \mathbb{R}^n$ with $\mathbf{x} \in \mathbb{R}^n$ and $t \in \mathbb{R}$, LCS represent $(n-1)$-dimensional manifolds that separate qualitatively different regions of the vector field. Shadden et al. have shown that a ridge in the FTLE field does only represent an LCS if its cross-flux, i.e., the flux of the underlying time-dependent vector field $\mathbf{u}(\mathbf{x},t)$ across the (moving) ridge, is sufficiently low. Shadden et al. derive a theoretical analysis for this cross-flux, and also measure it by extracting ridges for a given time t_0 from the FTLE field $\sigma^T_{t_0}(\mathbf{x})$, seeding points along these ridges, advecting these points to a later time $t_0 + T\gamma$, and measuring the distance of these advected points to the ridges extracted from $\sigma^T_{t_0+T\gamma}(\mathbf{x})$ (Figure 5.2a). In other words, they measure to what extent the ridges from consecutive (increasing t_0) FTLE fields move like material lines, i.e., advect as sets of massless particles in $\mathbf{u}(\mathbf{x},t)$.

We want to note here, however, that this approach can be affected by the repelling behavior of an LCS. Ridges in forward-time FTLE (obtained with forward integration, i.e., $T > 0$) represent repelling LCS, i.e., they repel particles in forward time. In other words, a perturbation perpendicular to the LCS will grow exponentially in forward time. The opposite holds for reverse-time FTLE (with $T < 0$), i.e., ridges in reverse FTLE repel particles in reverse time and attract particles in forward time, which is why they are denoted attracting LCS. Since these perturbations grow exponentially, and because exponential growth starts very slowly, it is the case that if the FTLE ridges used for the material line

test are extracted at sufficient accuracy, the cross-flux will dominate the repelling behavior. However, if the accuracy is insufficient, the repelling behavior can dominate the cross-flux. However, in either case the measure is useful, because both effects add up, and thus provide a conservative measure for the quality of LCS extraction. In consistency with Shadden et al.'s approach, we denote the discrepancy between the advected material line and the respective FTLE ridges "cross-flux", even though it might include LCS repulsion. Nevertheless, in Section 5.2.1, we present an extension of Shadden et al.'s approach which makes it more expressive and thus better suited to assess the quality of LCS visualization.

Besides this repulsion, there are mainly three reasons why an FTLE ridge can exhibit a high cross-flux and thus does not represent an LCS: First, the lifetime T_χ of the hyperbolic region (Figures 2.7a and 5.2b) that causes the ridge might be too short in relation to its hyperbolic strength

$$|\chi| = \min(0, -\det \nabla \mathbf{u}(\mathbf{x}, t)) \tag{5.1}$$

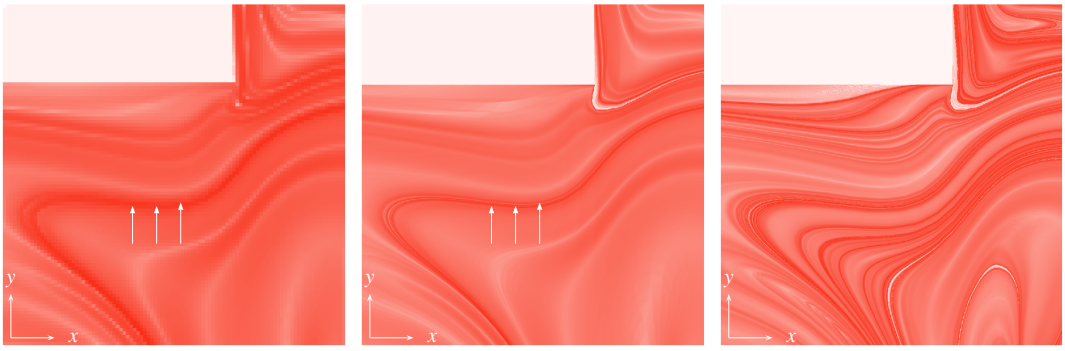
to cause sufficient particle separation. In such a case, the hyperbolic region does not contribute an LCS. Second, the advection time T might be chosen too short compared to the strength of the hyperbolic region. In this case, the lifetime of the hyperbolic region is longer than T, and increasing T will reduce the cross-flux and thus enable the extraction of the LCS. Third, there might be no hyperbolic region involved, and particle separation is caused by shear flow. FTLE ridges caused by shear do not separate different regions, they exhibit (except for degenerate configurations in unidirectional flow) high cross-flux, and thus do not represent LCS. Notice the related but algorithmically and numerically demanding concept of Haller [Hal11] to avoid extraction of shear-induced manifolds.

Until now, FTLE ridges have been filtered by means of thresholding of the FTLE value, or by their "sharpness" by thresholding of the respective eigenvalue of the Hessian [SP07], both on a qualitative "purely visual" basis. In Section 5.2.1, we present a cross-flux-based technique for FTLE ridge filtering that ensures that the resulting ridges represent LCS with respect to a predefined maximum cross-flux.

5.1.2 FTLE and Resolution

So far, we discussed both the LE and the FTLE in terms of infinitesimal sampling resolution ($\|\boldsymbol{\delta}_{t_0}(\mathbf{x})\| \to 0$, or continuous flow maps. Since the LE was motivated by predictability analysis in terms of the exponential growth of a perturbation due to an underlying linear vector field (thus the abovementioned linearization of the field), one needs to involve the limit $\|\boldsymbol{\delta}_{t_0}(\mathbf{x})\| \to 0$ when computing the LE for predictability.

Haller [Hal01] states that, in contrast to the LE, where a discretized computation would fail due to exponential growth of errors, the FTLE could be computed with sufficient fi-

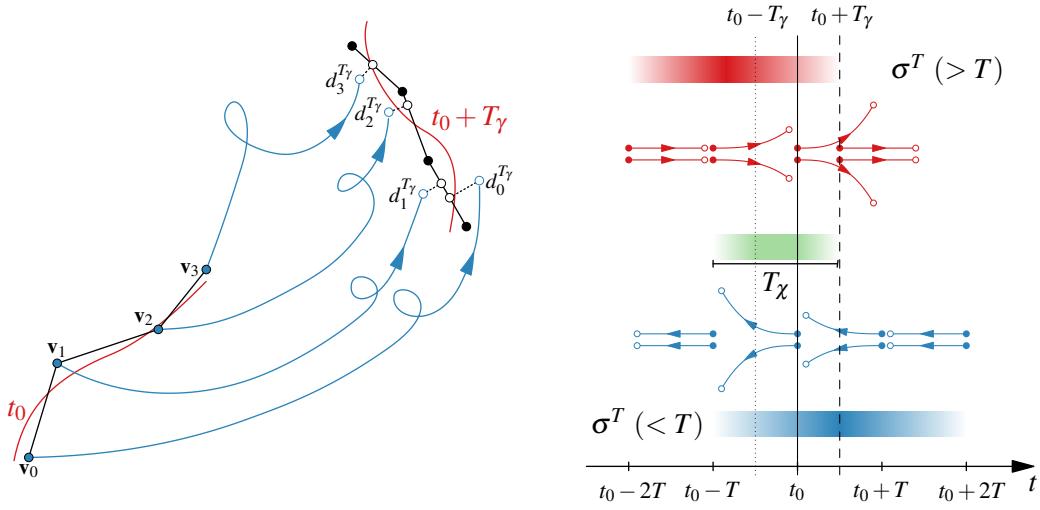


(a) (b) (c)

Figure 5.1: FTLE resolution and LCS in Buoyant Flow dataset. (a) Low resolution (100×100) FTLE ($T = 0.075\,\mathrm{s}$, $t_0 = 0.2\,\mathrm{s}$, higher by more saturation). (b) Same as (a), but at higher resolution (1000×1000) reveals closely adjacent LCS (arrows), which could not be distinguished in (a). (c) Same as (b), but with advection time $T = 0.2\,\mathrm{s}$. This increase of T by a factor less than three reveals finely folded LCS due to chaotic advection.

nite resolution, if T is limited accordingly. However, this would require that the trajectories stay close enough to allow for the linearization, which, however, cannot be achieved in most practical LCS visualizations, because for useful choices of T the flow map resolution would need to be extremely high. However, as common practice and the validations by Shadden et al. regarding FTLE ridge cross-flux have shown, such an incredibly high resolution is not needed when the FTLE is used for topological analysis, since we do not need to estimate a rate of perturbation growth, as would be needed for predictability analysis. For topological analysis, we only need to make sure that we seed trajectories along both sides of an LCS, i.e., that we do not miss regions that separate LCS.

Figures 5.1a and 5.1b show two different resolutions of a region with close-by LCS, as is often encountered. The underlying process responsible for LCS "convergence" is known as "thinning and folding", where thinning is caused by the stretching effect of the hyperbolic region, and folding takes place because the flow needs to return after some time since it is typically constrained to a finite range in space. This thinning and folding mechanism is responsible for so-called chaotic advection, and in our case, because LCS advect with the flow as material lines do, LCS are also folded and stretched, rapidly leading to very closely adjacent LCS (see Figure 5.1c for the same region as shown in Figure 5.1b but with only roughly three times higher advection time T). Thus, for reliable LCS visualization, a technique is needed that can detect such undersampling and possibly adapt the resolution with respect to a given advection time. In Section 5.2.2, we present such a technique that provides an LCS *consistency measure* for a given advection time T and resolution, and a refinement based on that in Section 5.2.3.

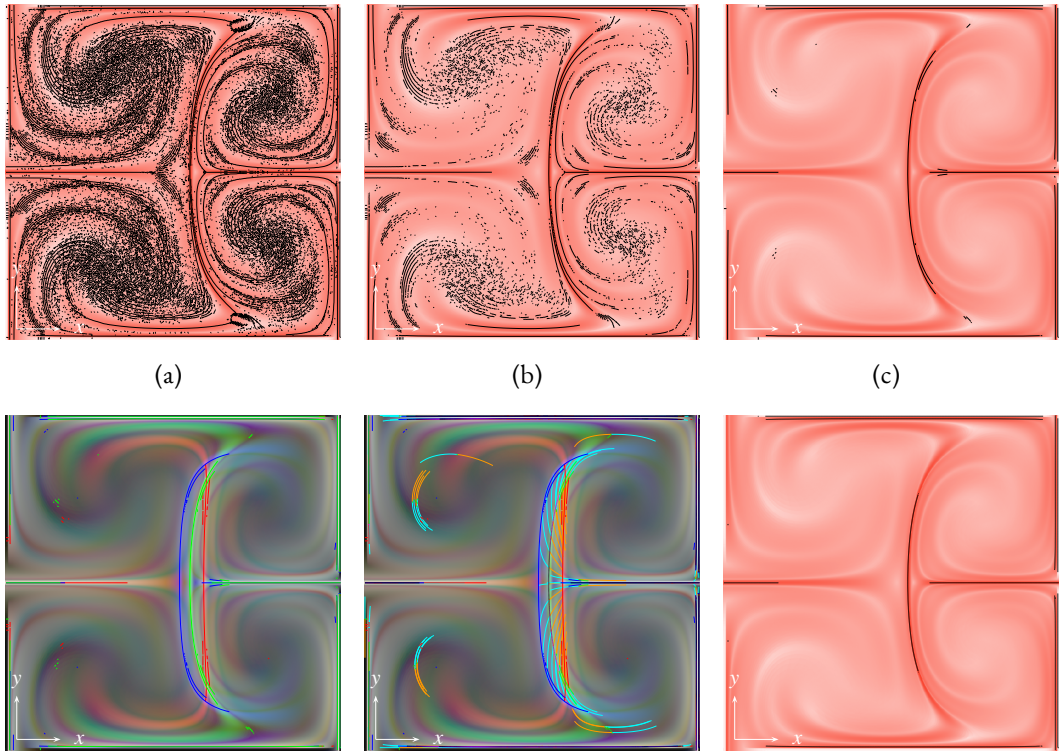


(a) (b)

Figure 5.2: Cross-flux computation. (a) Our technique measures, for each vertex $\mathbf{v}_i$ (blue dots) of FTLE ridge (black) at time t_0 (extracted from $\sigma^T_{t_0}(\mathbf{x})$), the closest distance $d_i{}^{T_\gamma}$ (dotted) from the respective pathline endpoint (blue circle) to the FTLE ridge (black) extracted at time $t_0 + T_\gamma$ from $\sigma^T_{t_0+T_\gamma}(\mathbf{x})$. The same analysis is conducted in reverse direction, i.e., $d_i{}^{-T_\gamma}$ is computed for $-T_\gamma$, and for each vertex $\mathbf{v}_i$, the minimum of $d_i{}^{T_\gamma}$ and $d_i{}^{-T_\gamma}$ is taken. The reason for going in both directions is to avoid "truncation" of LCS at the ends of their lifetime (as opposed to Shadden et al. [SLM05]). Such a temporal truncation would, e.g., happen for FTLE ridges extracted from $\sigma^T_{t_0}(\mathbf{x})$ in (b) because the FTLE $\sigma^T_{t_0+T_\gamma}(\mathbf{x})$ (computed at time $t_0 + T_\gamma$, dashed) would not exhibit a ridge anymore since the lifetime T_χ of the hyperbolic region does not extend into its time scope. Nevertheless, doing the cross-flux test at $t_0 - T_\gamma$ (dotted) would work because forward FTLE $\sigma^T_{t_0-T_\gamma}(\mathbf{x})$ would capture the hyperbolic region at that time. Hyperbolic strength $|\chi|$ by green, forward FTLE by red, and reverse by blue.

5.2 Method

Our method consists of two main components: an approach to assess the quality of FTLE ridges (and filter them) with respect to their cross-flux (Section 5.2.1), and an approach that quantifies their sampling issues with respect to "LCS consistency" (Section 5.2.2). These two components are integrated, i.e., the cross-flux measure is used to filter the result from height ridge extraction, and this filtered result is then tested with the ridge consistency measure. If this consistency measure is insufficient, the sampling resolution is automatically increased until a consistent result is obtained (Section 5.2.3).



(a) (b) (c)

(d) (e) (f)

Figure 5.3: Cross-flux-based filtering of FTLE ridges in Quad-Gyre dataset (FTLE resolution 800×800, $T = 5.0\,\mathrm{s}$, $t_0 = 0\,\mathrm{s}$). (Depending on the viewer, you might need to zoom into the electronic version of this thesis for the vector-graphics representation.) (a) Raw ridges, i.e., Hessian eigenvalue threshold $\tau_H = 0$. (b) Some spurious ridges suppressed (where minor Hessian eigenvalue $> \tau_H = -10^{-6}/{\mu_\phi}^{1.3}$, with μ_ϕ being flow map cell size). This threshold is used for all our results, except for the remaining images in this figure. (c) For illustration purposes (to avoid visual clutter), we set here $\tau_H = -6 \cdot 10^{-6}/{\mu_\phi}^{1.3}$ for (c)–(f). (d) $\sigma^T_{t_0}(\mathbf{x})$ mapped to green channel, $\sigma^T_{t_0+T\gamma}(\mathbf{x})$ to red, and $\sigma^T_{t_0-T\gamma}(\mathbf{x})$ to blue, together with respective ridge extractions (green, red, and blue lines, respectively), with $T\gamma = 1\,\mathrm{s}$. (e) Same as (d), but with particle paths for $T\gamma$ (orange), and $-T\gamma$ (cyan), and with ridge from $\sigma^T_{t_0}(\mathbf{x})$ (green in (d)) now colored by resulting cross-flux discrepancy (black: $\Delta_\gamma = 0$, magenta: $\Delta_\gamma = \mu_\phi$, green: $\Delta_\gamma > \mu_\phi$). (f) Resulting LCS visualization.

5.2.1 FTLE Ridge Cross-Flux

As discussed by Shadden et al. [SLM05] and in Section 5.1.1, LCS advect as material lines do. That is, if one puts particles on an LCS at time t_0 and advects these particles to time $t_0 + T\gamma$ along pathlines, they will be located on the corresponding LCS at that later time (Figure 5.2a). Notice, however, that this assumes that the lifetime $T\chi$ of the hyperbolic region that causes the LCS is sufficiently long with respect to $T\gamma$, i.e., that it extends into the time interval $[t_0 + T\gamma, t_0 + T\gamma + T]$ (Figure 5.2b). If this is not the case, as for the

illustrated t_0 where $t_0 + T\gamma$ (dashed) is located after the hyperbolic region has disappeared, the respective LCS might not be present at time $t_0 + T\gamma$ anymore.

Whereas Shadden et al.'s focus was on the theoretical derivation of the cross-flux across LCS caused by permanently present hyperbolic regions, they evaluated this flux numerically only for an isolated FTLE ridge. In this chapter, we present a technique that quantifies the cross-flux of ridges extracted from practical FTLE fields (containing many and adjacent ridges, including spurious ones), and conducts this measurement forward and reverse, to avoid truncation (Figure 5.2b). More important, instead of relating the measured cross-flux to a theoretical limit which is assuming hyperbolic activity over the entire FTLE integration time T, we present an approach to filter FTLE height ridges with respect to their physical quality by means of a single threshold, i.e., by means of the cross-flux time $T\gamma$. We achieve this by relating this time to the sampling properties of height ridge extraction and the cross-flux across the respective LCS.

The basis of our approach follows the idea by Shadden et al., i.e., we seed particles at the ridges of the respective FTLE field $\sigma^T_{t_0}(\mathbf{x})$ at time t_0, in our case at each vertex $\mathbf{v}_i$ of the height ridges extracted according to Eberly's [Ebe96] formulation. We then advect these particles (along pathlines) for the cross-flux time $T\gamma$, and measure the shortest distance ${d_i}^{T\gamma}$ of these advected particles to the height ridges extracted from the FTLE field $\sigma^T_{t_0+T\gamma}(\mathbf{x})$ at time $t_0 + T\gamma$ (Figure 5.2a). This provides us with a distance ${d_i}^{T\gamma}$ for each vertex $\mathbf{v}_i$ of the original ridge, i.e., its "advection discrepancy". To avoid the temporal truncation due to missing LCS at time $t_0 + T\gamma$, we additionally advect a particle from each vertex $\mathbf{v}_i$ in reverse direction, i.e., for cross flux time $-T\gamma$, measure its respective closest distance ${d_i}^{-T\gamma}$ to the ridges extracted from $\sigma^T_{t_0-T\gamma}(\mathbf{x})$ (dotted in Figure 5.2b), and determine cross-flux discrepancy

$$\Delta_\gamma = \min({d_i}^{T\gamma}, {d_i}^{-T\gamma}) \; . \tag{5.2}$$

A straightforward application of this concept would be to choose a user-defined cross-flux time $T\gamma$ together with a user-defined threshold for the resulting cross-flux discrepancy Δ_γ, and reject those parts of the ridges where Δ_γ exceeds this threshold. This approach would, however, exhibit two major shortcomings: First, it would be unclear how these two choices should be made because they are closely but nonlinearly related, i.e., increasing $T\gamma$ tends to increase Δ_γ because the cross-flux is "integrated" for a longer time. Second, and more important, we experienced problems with this approach if the raw ridge extraction (before our filtering) contains closely adjacent ridges, which typically happens due to noise (Figure 5.3a, (b)) or due to the abovementioned folding of LCS (Section 5.1.2 and Figure 5.1c). In such cases of high ridge density, ${d_i}^{T\gamma}$ and ${d_i}^{-T\gamma}$ would be limited by

the distance to the next (possibly unrelated) FTLE ridge. This means, that if the corresponding LCS would be farther away from an advected particle than an other (erroneous ridge or close-by LCS), the shortest distance would be to this unrelated ridge, leading to too small Δ_γ, and thus leading to false-positive LCS extraction after cross-flux filtering.

As shown by Shadden et al., LCS exhibit negligible cross-flux. This means that for flow maps that are not sampled at extremely high resolution and if sufficient advection time T is used (which we want to assess indirectly via our measure), the cross-flux discrepancy Δ_γ should primarily reflect the inaccuracies due to ridge extraction (see the deviation of the black polylines in Figure 5.2a from the red LCS), which are in the order of the size of a cell of the flow map. Notice that these inaccuracies would even grow during advection of the particles for cross-flux time $T\gamma$, because the respective LCS are repelling. Nevertheless, consistent with the discussion above, the repelling behavior was small in our investigations based on pathlines, i.e., the true cross-flux was clearly dominating. Together with the issue of possibly too low Δ_γ due to erroneous ridges or close-by LCS discussed above, this motivates us to limit Δ_γ to the cell size $\mu_{\boldsymbol{\phi}}$ of the flow map:

$$\Delta_\gamma < \mu_{\boldsymbol{\phi}} \ . \tag{5.3}$$

As a result, we have only one parameter left to control the quality of our LCS extraction with respect to cross-flux: the cross-flux time $T\gamma$. The second (implicit) parameter is the resolution of the flow map. In Section 5.2.3, we will present an approach to automatically increase this resolution of the flow map to obtain results that consistently sample LCS, which brings both concepts into relation. But for this, we first need a respective LCS consistency criterion, which is introduced in Section 5.2.2.

Our technique can be applied to the raw ridge extraction result, as shown in Figure 5.3a. This would, however, put particles on spurious ridges that are generated due to (numerical) noise, and thus cause a high computational cost because each of these particles needs to be advected by $T\gamma$ along a pathline. As proposed by Sadlo and Peikert [SP07], we suppress erroneous (and thus "unsharp") ridge parts by testing for each ridge vertex if the minor Hessian eigenvalue is below a threshold τ_H. Figure 5.3b shows a respective result with $\tau_H = -10^{-6}/{\mu_{\boldsymbol{\phi}}}^{1.3}$. Note that the purpose of this threshold is simply to speed up computation by removing noise, one can set $\tau_H = 0$ if this is not desired. We made this threshold a function of the flow map cell size $\mu_{\boldsymbol{\phi}}$ to compensate for the change of ridge sharpness as the resolution of the flow map is increased (used for the automatic resolution adaptation in Section 5.2.3). We determined the exponent of 1.3 by testing our technique with different datasets, resolutions, and FTLE advection times T—and use this threshold function for all our results. Nonetheless, this way of automatically choosing a threshold

for suppressing noise can, of course, only be a heuristics. We therefore recommend, as we do in our implementation and is shown in our results below, to provide and check visual feedback, or—more safe—to set $\tau_H = 0$.

For illustration purposes (to avoid visual clutter), we set $\tau_H = -6 \cdot 10^{-6}/\mu_\phi{}^{1.3}$ in Figure 5.3c–(f). This filtering (Figure 5.3c) also represents a traditional LCS visualization by means of filtered FTLE ridges, i.e., τ_H was chosen on a "visual qualitative basis". Comparing this visualization, however, with the result of our technique (Figure 5.3f) and examining its quality with respect to cross-flux (as seen by forward and reverse particle advection pathlines in Figure 5.3e and the consistency of their endpoints with the red and green ridges) clearly shows the advantage of our quantitative approach. It is apparent that our quantitative LCS visualization technique removed the (green) parts in that image that should not be considered LCS due to substantially higher cross-flux.

5.2.2 FTLE Ridge Consistency

So far, the cross-flux-based filtering of FTLE ridges from Section 5.2.1 enables us to filter FTLE ridges with respect to the advection property in a quantitative manner, i.e., according to the extent they advect as material lines. According to Shadden et al., this is a sufficient requirement for FTLE ridges to represent LCS. A remaining question is, however, the quality of their discretization. This question is of very high importance, because LCS tend to be closely adjacent, exhibit folds, and the FTLE tends to suffer from very strong aliasing at LCS because the sharpness of its ridges increases with advection time T. This makes FTLE fields very hard to sample in practice. Beyond that, it is a typical situation that one extracts an FTLE ridge at a certain resolution, but when the resolution of the FTLE is increased, it turns out that what has been considered a single LCS is composed of a large number of LCS instead (cf. Figures 5.1a and 5.1b).

Existing approaches that employ adaptive sampling of the FTLE field [Gar+07; SP07] guide their refinement by the local variation of the FTLE field at the current resolution. A common problem with these approaches is, however, that they only have a "phenomenological" view on the FTLE field, and that one typically needs to increase the sampling resolution very substantially before the FTLE field reveals that an apparent LCS actually consists of several ones.

Coming back to the assessment of the quality of LCS visualizations, a possible but unfeasible approach would be to increase the sampling resolution or zoom in interactively, compute the respective FTLE field, extract the ridges therefrom, and visually examine the differences. However, this approach would be tedious, and—more important—would

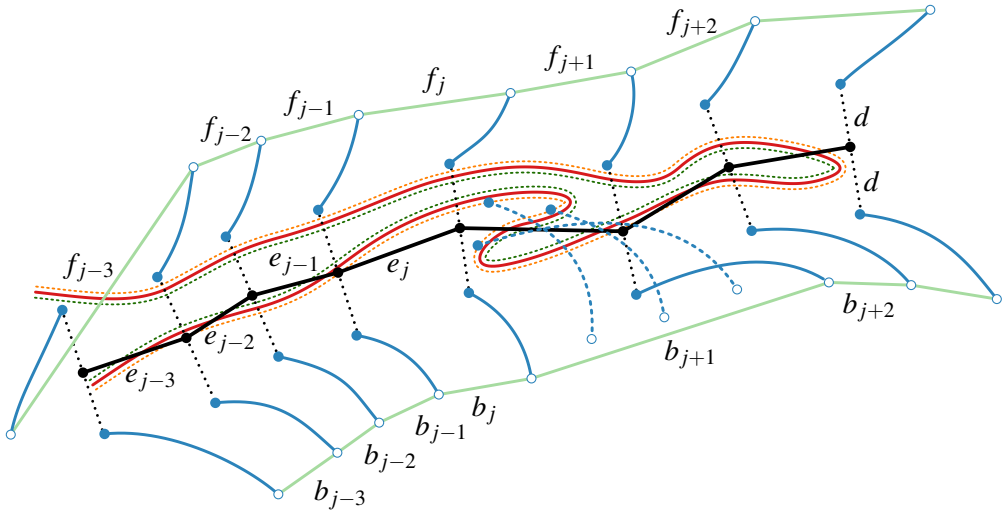
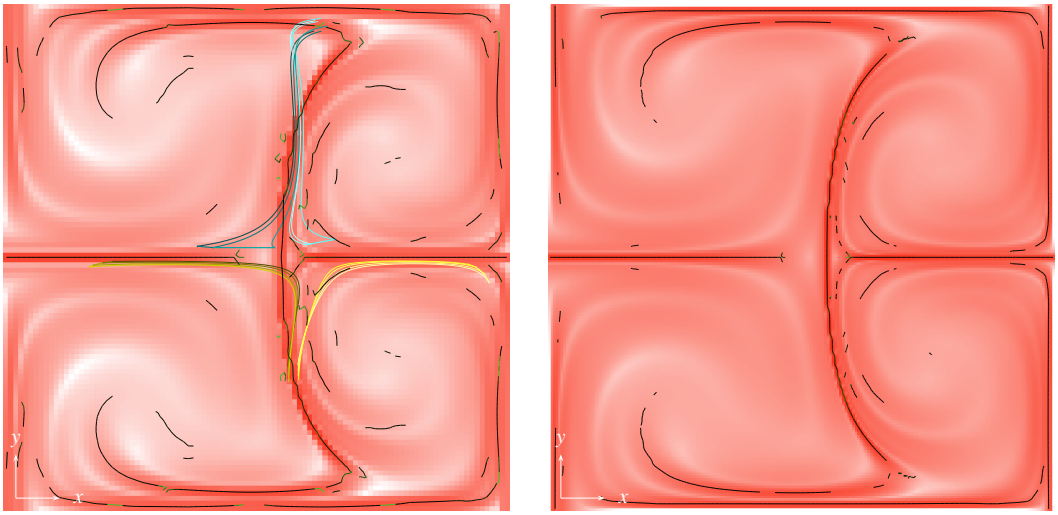


Figure 5.4: Ridge consistency computation. Finely folded LCS (red), together with discretized ridge extraction (black) thereof, and "front" and "back" seed manifolds (blue points above and below, respectively). The endpoints of trajectories seeded at blue points exhibit irregularities when seed manifold crosses an LCS (left), or when folds are "shortcut" because the discretized ridge did not capture them (bottom right). Ridge (LCS) consistency is determined for each ridge edge e_j from the maximum of the standard deviation of the lengths of f_{j-1}, f_j, and f_{j+1}, and the standard deviation of the lengths of b_{j-1}, b_j, and b_{j+1}.

likely require an enormous increase in resolution to be able to identify that there are, e.g., two LCS instead of one.

We follow a different approach here, motivated by the properties of LCS. An LCS separates regions of qualitatively different behavior of a vector field. Thus, there is qualitatively similar pathline behavior on either side of an LCS. As illustrated in the "upper half" of Figure 5.4, this means that if we regularly seed pathlines at the same time and of equal integration time along one side of an LCS, the endpoints of these pathlines will form a manifold that is "regularly" sampled (regular meaning here that this sampling will in general not be uniform but that the distances between the endpoints will vary continuously). The reason for this is that if there would be large gaps, i.e., "jumps" in the placement of the endpoints, this would have been captured by an LCS that would have been located between the respective seeds of our sampling along the original LCS, thus leading to a contradiction.

Without loss of generality, let $\mathscr{L}^T_{t_0}(\mathbf{s}) \in \mathbb{R}^n$ be an LCS in an n-dimensional vector field, i.e., due to the continuity assumption on the vector field, be a $(n-1)$-manifold obtained



(a) (b)

Figure 5.5: Ridge consistency computation at the example of the Quad-Gyre dataset (FTLE by red, $T = 5\,\mathrm{s}, t_0 = 0\,\mathrm{s}$). (a) Resolution 100×100, with FTLE ridges filtered with cross-flux ($T\gamma =$ 1 second), and seed manifold trajectories (blue in Figure 5.4) started at ridge segments with low LCS consistency (top, cyanish) and high LCS consistency (bottom, yellowish). Ridge segment consistency by good (black) to low (green). (b) For comparison, ridge segments for same case but computed from 200×200 FTLE field. One can observe better LCS consistency due to the higher resolution used.

at time t_0 for advection time T, with parametrization $\mathbf{s} \in \mathbb{R}^{n-1}$, and let $\mathcal{N}_{t_0}^T(\mathbf{s}) \in \mathbb{R}^n$ be its normal. Then, we can generate a seed manifold

$$\mathcal{F}_{t_0}^T(\mathbf{s}) = \mathcal{L}_{t_0}^T(\mathbf{s}) + d \cdot \mathcal{N}_{t_0}^T(\mathbf{s}) \tag{5.4}$$

at its "front", and a seed manifold

$$\mathcal{B}_{t_0}^T(\mathbf{s}) = \mathcal{L}_{t_0}^T(\mathbf{s}) - d \cdot \mathcal{N}_{t_0}^T(\mathbf{s}) \tag{5.5}$$

at its "back", which are both offset from the LCS in normal direction by distance d.

Using the flow map $\boldsymbol{\phi}_{t_0}^T$, we can obtain the advected front seed manifold and advected back seed manifold

$$\overline{\mathcal{F}}_{t_0}^T(\mathbf{s}) = \boldsymbol{\phi}_{t_0}^T\big(\mathcal{F}_{t_0}^T(\mathbf{s})\big)\,, \quad \overline{\mathcal{B}}_{t_0}^T(\mathbf{s}) = \boldsymbol{\phi}_{t_0}^T\big(\mathcal{B}_{t_0}^T(\mathbf{s})\big)\,, \tag{5.6}$$

respectively. Notice that since both the vector field and the LCS are continuous by assumption, the advected manifolds are continuous too. The manifolds will typically un-

dergo stretching and squeezing as they are advected by the flow, leading to a continuous variation of their parametrization, which is can be captured by their gradient

$$\nabla_{\mathbf{s}} \overline{\mathscr{F}}^{T}_{t_0}(\mathbf{s}) \quad \text{and} \quad \nabla_{\mathbf{s}} \overline{\mathscr{B}}^{T}_{t_0}(\mathbf{s}) \tag{5.7}$$

with respect to $\mathbf{s}$.

Let us now assume that the seed manifolds are generated from an FTLE ridge $\mathscr{R}^{T}_{t_0}(\mathbf{s})$ instead of the LCS $\mathscr{L}^{T}_{t_0}(\mathbf{s})$, and that because $\mathscr{R}^{T}_{t_0}(\mathbf{s})$ does not represent $\mathscr{L}^{T}_{t_0}(\mathbf{s})$ sufficiently well, at least one of the seed manifolds crosses the true LCS. Assume the black polyline in Figure 5.4 to represent our discretized $\mathscr{R}^{T}_{t_0}(\mathbf{s})$, the upper sequence of blue dots (one generated from each vertex of the ridge polyline and offset to the "front") the discretized seed manifold, and the leftmost blue dot of this sequence to be on the other side of the LCS (red). One can see that, since the LCS separates different trajectory behavior and therefore this trajectory has different shape, the segment f_{j-3} is very long compared to its neighbors, e.g., segment f_{j-2}. In other words, the fact that the seed manifold crossed the LCS caused a strong spread in endpoints of the trajectories. Such seed manifold crossings happen, for example, when multiple LCS are captured by a single ridge polyline, and the LCS are diverging, as illustrated at the left of the figure. Because LCS originate at hyperbolic regions and are folded therefrom, such openings in the folding need to take place, and thus provide a point for detecting insufficient sampling of an LCS.

An other problem that can arise with LCS sampling are local folds, i.e., situations as illustrated in the bottom half of Figure 5.4. The behavior of trajectories (and thus their endpoints) seeded along the lower side of the LCS (illustrated by orange dots) is by definition coherent (following the same argumentation as above). This includes, however, the trajectories started "within" the fold, i.e., the concave region containing the orange dots. The dotted blue trajectories seeded within that region illustrate the behavior of the trajectories, and one can observe that the endpoints of the dotted trajectories form a regular sampling together with the blue trajectories seeded outside the fold. However, since our discretized ridge (polyline) did not capture the fold, its discrete seed manifolds (the lower sequence of non-dashed trajectories) do not capture it neither—leading to a strong spread of the advected manifold, i.e., the segment b_{j+1} is much longer than its neighbors b_j and b_{j+2}.

These two mechanisms enable us to measure the LCS consistency of a discretized FTLE ridge by seeding trajectories along it and evaluating the behavior of their endpoints. We describe our approach here for 2D vector fields, although its extension for 3D vector fields is straightforward.

For each segment e_j of the ridge polyline, we determine its segment neighbors e_{j-1} and e_{j+1}, collect their four vertices they contain (at their ends respectively less), and seed a trajectory at each vertex, offset in forward direction of the tangent normal at distance d (blue "front", upper dots in Figure 5.4). Each of these trajectories is integrated for time T, leading to the respective advected segments f_{j-1}, f_j, and f_{j+1}. We then compute the standard deviation of the lengths of these three segments, and divide this standard deviation by the size of the physical domain of the dataset to make it dimensionless, leading to the front ridge coherency measure $\rho_j^{\gamma_f}$. We do the same for each vertex in reverse direction of the tangent normal, leading to the back ridge coherency measure $\rho_j^{\gamma_b}$. The LCS consistency measure for a ridge segment e_j is then given as

$$\rho_j^{\gamma} = -\max(\rho_j^{\gamma_b}, \rho_j^{\gamma_f}) \,. \tag{5.8}$$

Note that this is a negative value, i.e., that $\rho_j^{\gamma} = 0$ indicates full consistency.

For a given LCS visualization (i.e., a set of FTLE ridges), we quantify its total LCS consistency

$$\rho^{\gamma} = \min_{j \in \mathscr{R}_{i_0}^T(\mathbf{s})} \rho_j^{\gamma} \tag{5.9}$$

as the minimum over all its segments, i.e., ρ^{γ} provides a conservative measure with respect to LCS consistency.

5.2.3 Automatic Adjustment of Resolution

Now we are at the point to integrate the two presented techniques, i.e., the cross-flux-based filtering of FTLE ridges (Section 5.2.1) and the LCS consistency measure (Section 5.2.2), into their intended from.

The cross-flux-based ridge filtering enforces LCS visualization quality by filtering, i.e., rejecting, ridge regions with a too high cross-flux discrepancy $\Delta\gamma$. The ridge consistency measure ρ^{γ}, on the other hand, only *quantifies* the quality of LCS visualizations with respect to LCS sampling. These properties make them an ideal bundle for LCS visualization, i.e., the former filters an LCS visualization to reach a certain cross-flux quality, whereas the latter takes this filtered LCS visualization and determines if its resolution is sufficient with respect to the present LCS, or if its resolution needs to be increased. Remember that our cross-flux measure takes the cross-flux time $T\gamma$ as user input and relates it to FTLE resolution. Thus, if the LCS consistency measure suggests to increase resolution, the cross-flux measure will automatically steer and adjust the filtering with respect to this new resolution.

Our integrated approach works as follows: The user prescribes the desired quality of the LCS visualization with respect to cross-flux by providing a cross-flux time T_γ, and with respect to LCS sampling consistency by providing a threshold τ_ρ. He also defines a minimum (initial) FTLE resolution, the FTLE advection time T, and the desired instant of time t_0. Our approach then computes the FTLE for this resolution, advection time, and t_0, extracts the ridges therefrom, filters these ridges with respect to cross-flux using T_γ, and subsequently evaluates that result with the LCS consistency ρ^γ. If

$$\rho^\gamma < -\tau_\rho \ , \tag{5.10}$$

the approach increases the resolution (in our implementation, we double the resolution in each dimension) and reiterates the procedure with this increased resolution. Otherwise, the process stops, i.e., has converged, and the last filtered FTLE ridges represent the result. To avoid very long computation times (which are rather readily reached because accurate LCS visualization necessitates surprisingly high resolutions, as shown below in our analysis), we stop the process if a maximum resolution is reached.

Since increasing resolution constrains the tolerable cross-flux during cross-flux-based filtering, our procedure will always converge at least trivially because if the prescribed ridge consistency τ_ρ cannot be reached, the cross-flux filtering will eventually eliminate all ridge segments. Nonetheless, as our analyses below show, our technique converges in practical applications non-trivially, i.e., the ridge consistency is reached before cross-flux error hinders further cross-flux consistent extraction, however, given that the compute resources are available.

5.2.4 Implementation Details

We compute the FTLE (the flow map) on *Cartesian sampling grids*, and compute derivatives with central differences (and forward/backward differences at the boundaries). We compute the Hessian, needed for height ridge extraction, as the gradient of the gradient. All FTLE fields, flow maps, trajectories, gradients and Hessians, the adapted marching squares algorithm for ridge extraction, and the minimum distance from a point to a set of polylines used in cross-flux computation are vectorized operations implemented in CUDA kernels and run on the GPU in parallel. The backbone of the program gathering, transforming, and streaming the data to the kernels is implemented in Python.

As discussed previously, the bottleneck in terms of computing resources is typically the graphics memory, which may be exhausted as the resolution increases, depending on the amount available in the graphics card. In order to mitigate this, our implementation includes an automatic adaptive domain subdivision process that identifies and gen-

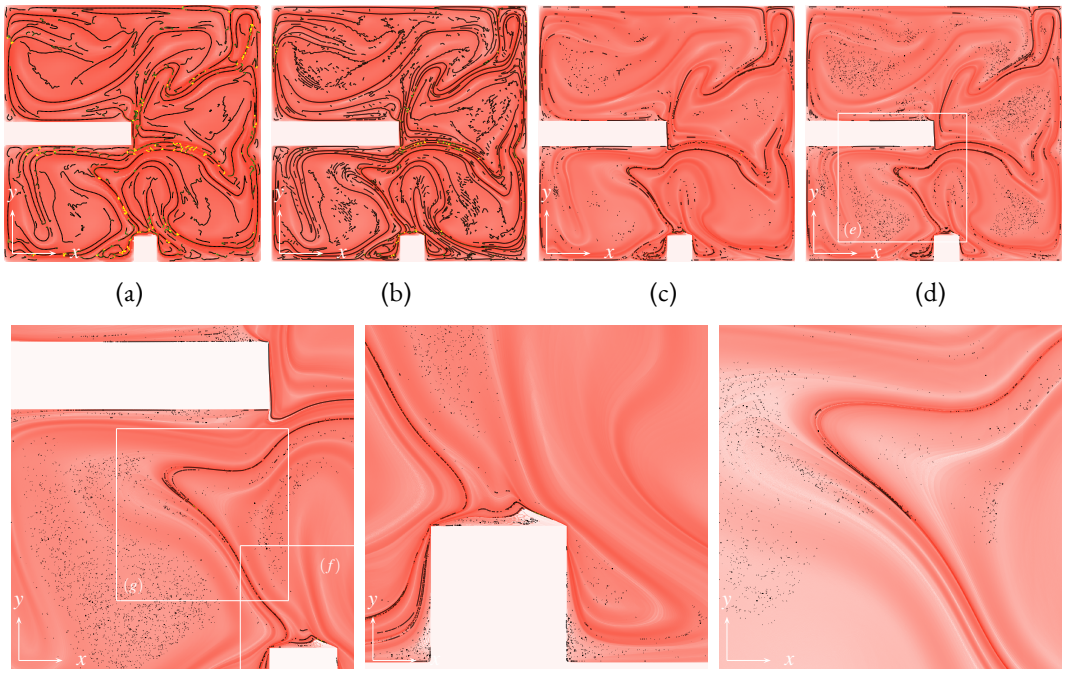


Figure 5.6: Automatic adaptive resolution at work. The top row contains the regular runs with the whole domain for the Buoyant Flow dataset with $T = 0.07$ and the resolutions 200×200 (a), 400×400 (b), 800×800 (c) and 1600×1600 (d). The latter did not converge, with four segments with ρ^γ over the threshold $\tau_\rho = 0.0625$, laid out in a single region, around the two walls. The bottom row contains this automatically-generated region with zoom $z = \times 2$ (e). This one did not converge, with 21 segments over the threshold laid out in two separate regions, so the system spawned two additional regions with zoom $z = \times 4$, shown in (f) and (g). Of these, the first one (f) did not converge due to 4 segments with ρ^γ over the threshold, but the maximum target resolution was reached, so the system did not subdivide further. The second one (g) converged successfully at this depth.

erates sub-regions recursively (Figure 5.6). If the maximum resolution is reached without achieving convergence, the algorithm first detects the areas which contain non-converged features within the current setting. Then, it delimitates sub-regions within the domain around these features by setting the appropriate parameters (i.e. zoom and pan), and finally it processes them in sequence, in a recursive manner (i.e. if the sub-regions themselves do not converge, additional levels are created automatically until convergence is achieved). The end result is a tree of sub-regions, with the full domain as the root node. Any node can have as many children as necessary.

This process works well enough in most cases, but it has some shortcomings:

1. The sub-domain extent may become too small with respect to the $T\gamma$, so that there is not enough room for the features to show up.
2. At high T values, as the effective resolution increases, the features may become too sharp, leading to a trivial convergence due to a lack of LCS, as discussed above in Section 5.2.3.

5.3 Results

Our approach is based on the cross-flux quality measure proposed and both analytically and numerically evaluated by Shadden et al. [SLM05]. Because this analysis and evaluation has been conducted at high detail, we concentrate here on the evaluation of our second main component, the quantification of LCS consistency, and the refinement (adaptation of resolution) it steers in a intertwined manner together with our formulation of the cross-flux discrepancy (since both concepts are related via the cell size of the flow map).

Because this interplay of these two techniques cannot be analyzed individually, we conducted several analyses of the convergence properties of our approach. To that end, we investigated the combination of different advection times T, ridge consistency thresholds τ_ρ, and cross-flux times $T\gamma$, resulting in several matrices of images together with data. In each field of these matrices, our technique adapts the resolution of the flow map (by increasing it) until no ridge line segment violates the condition in Equation 5.10, or until the maximum resolution has been reached.

Figure 5.7 provides such a series of tests within a region of interest in the Quad-Gyre dataset [SLM05], with different advection times and different thresholds to explore the convergence of our method in relation to the threshold value τ_ρ. These tests used $T\gamma = 0.5\,\mathrm{s}$. We observe that in this region of the Quad-Gyre dataset, convergence was met in all but the most restrictive quality setting and the highest advection time ($T = 11.0\,\mathrm{s}$). In this most restrictive case, where the threshold $\tau_\rho = 0.03126$, i.e., where the standard deviation of advected seed manifold segments exceeds 3.126% of the domain size, convergence was not met for the highest advection time even at a *grid resolution* of 3200×3200. We stopped our investigation here due to increased computation times (Figure 5.8).

Figure 5.10 shows the same matrix but using different $T\gamma$ values and a fixed τ_ρ set to 0.0625. In this case, all runs converged at different resolutions depending on the advection time and $T\gamma$. We observe that the largest $T\gamma$ setting we tested (rightmost column) converged much earlier than the others. This is due to a comparably large $T\gamma$, leading to early convergence with respect to cross-flux discrepancy.

The top row of Figure 5.9 displays a region of the Quad-Gyre dataset with a resolution of 100×100 and different τ_ρ values, from most restrictive (left) to most permissive

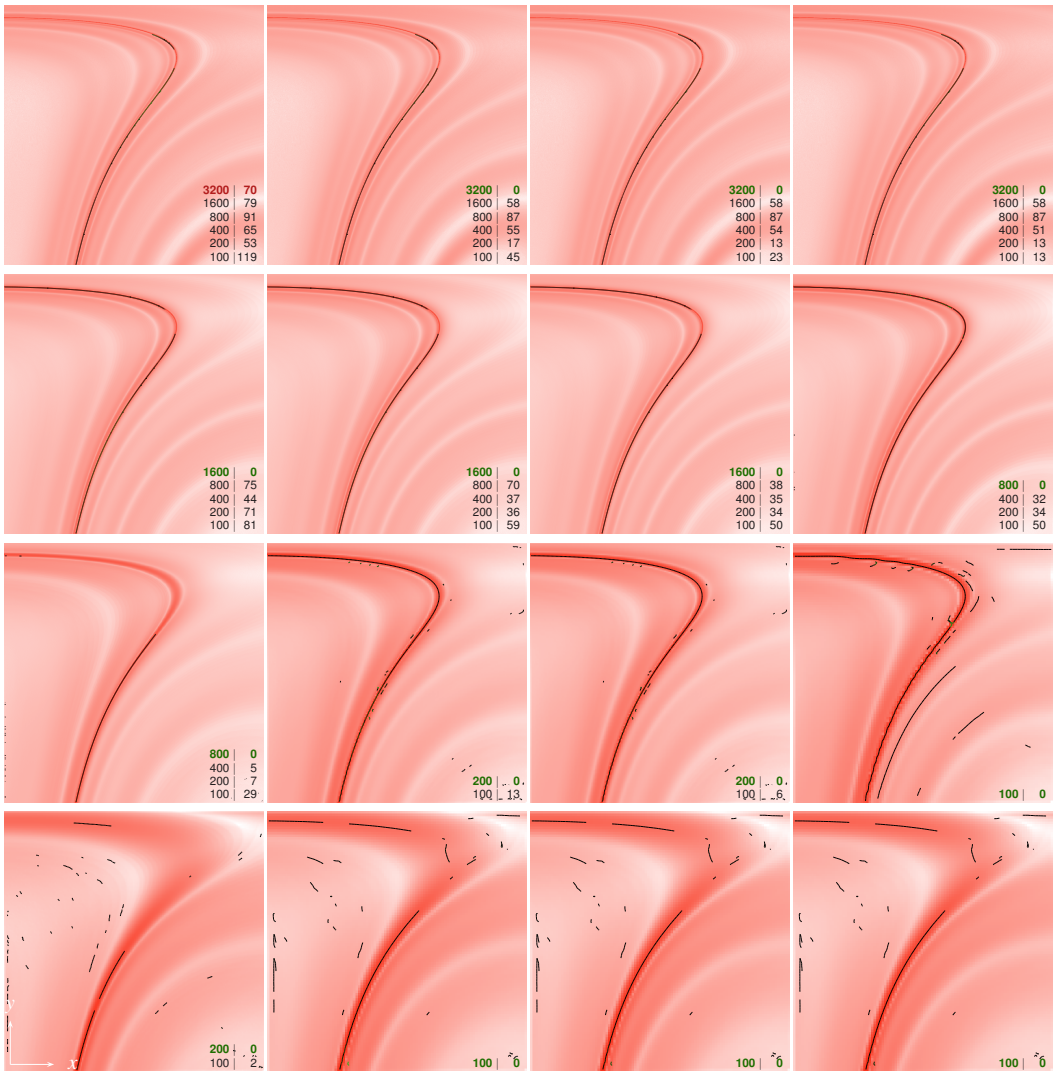


Figure 5.7: Ridge consistency test for the Quad-Gyre for different advection times $T = \{11, 9, 7, 5\}$ (top-to-bottom) and thresholds $\tau_\rho = \{0.03126, 0.0625, 0.125, 0.25\}$ (left-to-right). The tabulated values represent grid resolution vs. number of segments with ridge consistency value above the threshold. Converged values are highlighted in green.

(right). Note how the coloring of the ridges changes according to the threshold value. The bottom row shows the convergence series for the same run with $\tau_\rho = 0.03125$. As resolution increases the number of segments with ρ^γ value above the threshold decreases until convergence is met at resolution 800×800, where all segments are below the threshold. This series corresponds to the first image in the third row in Figure 5.9.

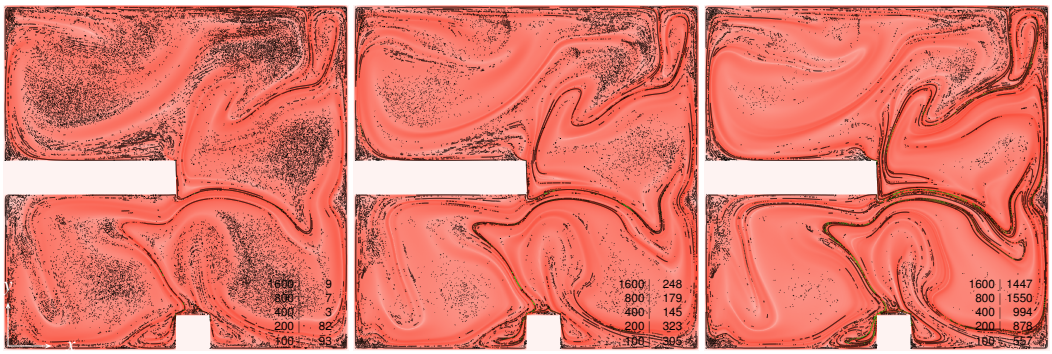


Figure 5.8: Convergence runs for Buoyant Flow dataset with different advection times. $T = \{0.05, 0.075, 0.1\}$ from left-to-right. None of these runs converged, even though $T = 0.05$ at 1600×1600 was very close to doing so with only 9 segments above τ_ρ (0.03126 in these runs).

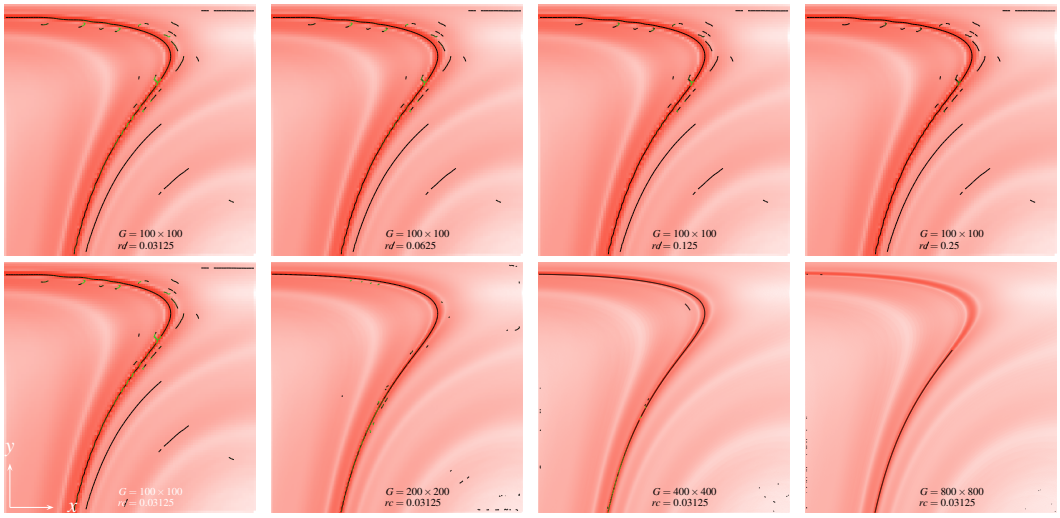


Figure 5.9: Ridge consistency test of the Quad-Gyre for different τ_ρ values at fixed grid resolution of 100×100 (top row) and for fixed $\tau_\rho = 0.03125$ and varying grid resolutions (bottom row).

As a second example, we applied our technique to a time-dependent 2D flow simulation of heat-driven air flow in a closed container with two barriers. We tested different settings for advection time T, but none of them converged. As is provided in the images, one can see that leftmost run almost converged with only nine segments violating the threshold τ_ρ. It can be seen, that while the case for $T = 0.05$ almost converged and is expected to converge with only a few more resolution increases, we do not expect convergence with the higher advection times, even if the resolution would be considerably increased. We account this fact to the prominently present chaotic advection dynamics in this dataset, which on the one hand required rather high advection times to exhibit sufficiently small cross-flux, but at the same time led to a topological complexity that re-

quires a very high sampling for LCS consistent sampling. This result shows, that LCS visualization without large compute resources has to be subject to substantial error due to LCS inconsistent sampling.

5.4 Performance Analysis

The following tables show some execution times for the convergence runs with the Quad-Gyre dataset. The run times are separated by cross-flux and ridge consistency time. The cross-flux values include the computation of the field and the ridge extraction at t_0, $t_0 + T\gamma$ and $t_0 - T\gamma$, and also the cross-flux processing itself. Computing the fields and extracting the ridges is by far the more expensive part in terms of run time. The ridge consistency values include only the ridge consistency processing.

Table 5.1 shows the run times for different resolutions and a fixed advection time $T = 11.0\,\mathrm{s}$. Table 5.2 shows the run times for different advection times at a resolution of 1600×1600. The performance measurements for the results shown in Figure 5.8 are displayed in Table 5.3 and provide a more detailed view on each of the single runs performed with this dataset. Here the compute time is not divided into cross-flux and ridge consistency, but rather the whole run time from start to end is shown for each resolution.

The RAM memory consumed in the worst-case run, the top-left 3200×3200 image shown in Figure 5.9, was 563.03125 MB. The GPU memory usage never went above 400 MB in any of the runs. All tests have been run on a machine with CentOS Linux release

Table 5.1: Performance results for different resolutions and fixed advection time $T = 11.0\,\mathrm{s}$.

	100^2	200^2	400^2	800^2	1600^2	3200^2
cross-flux	1 s	3 s	12 s	44 s	173 s	679 s
ridge consistency	1 s	2 s	2 s	4 s	5 s	7 s

Table 5.2: Performance results for different advection times and fixed resolution grid 1600×1600.

	$T = 5.0\,\mathrm{s}$	$T = 7.0\,\mathrm{s}$	$T = 9.0\,\mathrm{s}$	$T = 11.0\,\mathrm{s}$
cross-flux	418 s	222 s	191 s	182 s
ridge consistency	19 s	19 s	25 s	38 s

7.2.1511, kernel 3.10.0-327.el7.x86_64, 528 MB of RAM, an Intel(R) Xeon(R) CPU E5-2630 v3 at 2.40GHz and a pair of GTX Titan X 12 GB graphics cards.

5.5 Discussion

In this chapter, we presented, to the best of our knowledge, the first technique for visualization of Lagrangian coherent structures with quantitative error control, taking into account the intrinsic properties of LCS. We developed a technique for cross-flux-based filtering of FTLE ridges, and an approach to refine the FTLE resolution to ensure LCS consistent sampling. Whereas our experiments converged for various settings for the analytic Quad-Gyre example, obtaining accurate results with respect to cross-flux and LCS consistency turned out to be a challenge for the simulated dataset representing buoyant flow. This finding is, after all, not surprising, given the fact that many—if not most—FTLE visualizations underresolve LCS, leading to the fact that if the FTLE is recomputed at higher resolution, one typically observes additional LCS that could not be expected at the lower resolution. While we obtained good results with τ_ρ set to 0.0625 of the domain size, also with respect to quality of the obtained results in our convergence tests, i.e., the investigated examples perfectly matched our expectations, a future direction of research could include researching other measures for the quantification of LCS consistency.

As future work, we would also like to investigate the quality of (and respective extraction of) LCS defined by valleys in the finite-size Lyapunov exponent (FSLE) [Aur+97]. This quantity suffers less from aliasing, but has not yet been covered thoroughly both in applications as well as in analysis. Last but not least, we would like to extend our approach to 3D vector fields, which would, however, increase computation times even further, and require compute clusters for obtaining real-world results at high accuracy regarding cross-flux and LCS consistency.

Table 5.3: Detailed performance results for each run of the Buoyant Flow dataset (shown in Figure 5.8).

	$T = 0.05\,\mathrm{s}$	$T = 0.075\,\mathrm{s}$	$T = 0.1\,\mathrm{s}$
100×100	7 s	9 s	9 s
200×200	15 s	21 s	27 s
400×400	43 s	56 s	75 s
800×800	299 s	211 s	223 s
1600×1600	4825 s	1706 s	1064 s

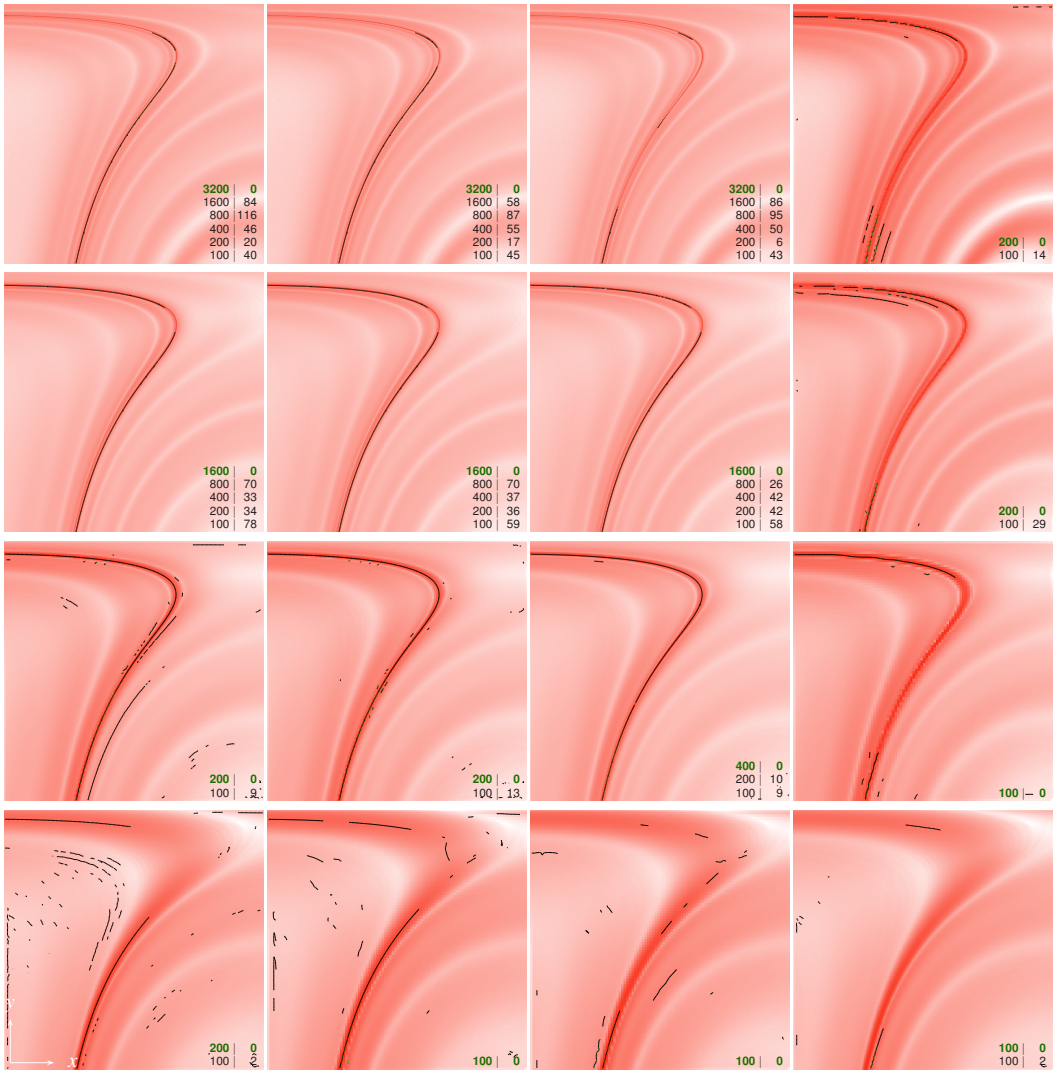


Figure 5.10: Look-ahead test for the Quad-Gyre for different advection times $T = \{11, 9, 7, 5\}$ (top-to-bottom) and look-ahead times $T\gamma = \{0.25, 0.5, 1.0, 2.0\}$ (left-to-right). The tabulated values represent grid resolution vs. number of segments with ridge consistency value above the threshold. Converged values are highlighted in green.

Part II

Large-Scale Visualization of Astrometric Data

Do not look at stars as bright spots only. Try to take in the vastness of the universe.

— Maria Mitchell

6 Computer Graphics and Rendering Fundamentals

In the first part of this work, we have developed novel methods and techniques in order to aid in the analysis and extraction of concealed information in astrophysical datasets, even though these methods are generic enough that they can be used and applied elsewhere in the greater flow visualization domain. We have so far based our approaches on concepts from vector field topology, and extended them to time-dependent inertial systems and phase space. We have also explored the time domain, while providing a comprehensive visual analysis framework to do so, and attempted to assist and quantify the extraction of structures that separate regions with distinct dynamics.

In contrast, this second part deals with the interactive representation and rendering of large-scale astrophysical systems in general, and astrometric data in particular. An example of such data are the extremely large star catalogs acquired by the *Gaia mission*. In this introductory chapter, we start with a comprehensive overview on computer graphics and rendering (Section 6.1) and later move on to methods and techniques typically used to achieve interactivity and improve performance with large scenes and datasets (Section 6.2). Finally, we provide a quick introduction to floating-point numbers, the most common and efficient decimal number representation used nowadays in graphics hardware, and an assessment of the precision of floating-point arithmetic (Section 6.3). These topics are very relevant to the work that follows in Chapter 7.

6.1 Rendering and Hardware

Rendering is one of the major topics in *computer graphics* (CG) and visual computing, and it is an essential part of the visualization process. It is usually the final step in the *visualization pipeline*, where models and other elements are given their final appearance.

Typically, rendering involves acquiring and processing information on geometry, viewpoint, lighting, texture and shading in order to produce a final image, also called render, which represents an impression of a (virtual) scene. The process by which a graphics system produces this final image is called rendering pipeline.

6.1.1 Computer Graphics Overview

Nowadays, a large part of the rendering process is carried out in specific hardware devices called *graphics processors* (GP or GPU) that execute operations independently from the *central processing unit* (CPU). While the whole process can be implemented entirely in software, and that is how it was initially, GPUs make it possible for the CPU to send away rendering commands and continue the execution of other tasks in an asynchronous manner. Apart from this off-loading of graphics tasks, they also accelerate the rendering process by implementing hardware architectures that are extremely fast at dealing with the kinds of operations involved in rendering. GPUs typically contain several types of electronic circuits in charge of different operations in the rendering pipeline, like texture mapping units (TMU), stream processors (pixel and vertex shading units) or geometry processors. All of these form the greater component of the graphics and compute array (GCA). The interaction with such hardware devices is abstracted to users by graphics application programming interfaces (APIs), which help them write graphics code that runs in a large variety of different GPU devices. OpenGL and Vulkan, both developed and maintained by the Khronos Group, are examples of cross-language, cross-platform and standard graphics APIs. The implementation of these standard APIs is usually left to the hardware manufacturers, which ship them in the form of graphics drivers. The communication is carried out via a rendering library (usually provided by the operating system), which in turn sends commands to the driver, which translates them to specific low-level instructions that the hardware can understand.

Modern 3D graphics boards usually (but not always) contain their own memory chip, called the video random access memory (VRAM). VRAM is directly wired to the GCA. It is much faster for the GCA to fetch and modify data from VRAM than from the main system random access memory (RAM). The VRAM usually holds the information that makes up the scene, like vertex buffer objects (VBO) and vertex array objects (VAO), together with the render buffers, textures and auxiliary information. In a simple, double-buffering scheme, the *front color buffer* contains the pixel data that is currently visible in the viewport, which is the area of the display that contains the rendered image. The *back color buffer* is the location of VRAM to which the data for the next frame are rendered. In triple-buffering and other more complex schemes, the back color buffer is actually com-

posed by more than one single buffer in an attempt to maximize performance and minimize artifacts derived from buffer swapping. The back buffer is not seen and exists so that the scene can be rendered in its entirety before being presented to the display. Typically, the front and back buffers are exchanged in the *buffer swap* operation once an image has finished rendering. This can be done by changing their memory addresses, or by actually copying the contents of the back buffer to the front buffer. The buffer swap is usually synchronized with the refresh rate of the display device to avoid an artifact known as *screen tearing*. VRAM also holds the *depth buffer*, also referred to as *z-buffer* (the name stemming from the convention that the z-axis in *camera coordinates* points directly out of the display screen). For each pixel in the image, the depth buffer stores a value that represents how far away it is, or how deep it lies in the image. The depth buffer is essential to perform hidden surface elimination by only allowing a pixel to be drawn to the color buffer if its *depth value* is less than the currently stored depth for that pixel. The depth values are measured as a distance from the virtual camera through which the scene is observed. The distance function most commonly implemented by default is a nonlinear function that improves the resolution of areas very close to the virtual camera and degrades rapidly when moving away from it. Most graphics APIs allow for the depth buffer to be programmatically accessed and written from user code in order to enable different distance functions to be implemented.

6.1.2 Rendering Equation

The rendering equation, also referred to as the reflectance equation, was introduced simultaneously by Kajiya [Kaj86] and Immel [ICG86]. It is an integral approximate description of the spectral radiance of a particular location in space based on the emitted spectral radiance and the reflectance distribution function. The radiance is all the radiant flux emitted, reflected, received and transmitted by a surface, per unit solid angle per unit projected area. The form in which the equation is presented is very well suited for computer graphics and includes contributions from different factors. The radiance from a position $\mathbf{x}$ and directed outward along direction ω_o is described as

$$L_\mathrm{o}(\mathbf{x}, \omega_\mathrm{o}) = L_e(\mathbf{x}, \omega_\mathrm{o}) + \int_\Omega f_\mathrm{r}(\mathbf{x}, \omega_\mathrm{i} \rightarrow \omega_\mathrm{o}) L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i})(\omega_\mathrm{i} \cdot \mathbf{n})\, \mathrm{d}\omega_\mathrm{i}. \tag{6.1}$$

The spectral radiance L_o is the addition of two terms. First, the emitted spectral radiance L_e along the outward direction ω_o, which may be due to an emissive property of the material itself, or to its temperature as described by the black-body radiation. Second, the integral over Ω (shown in Figure 6.1 as the unit hemisphere centered around

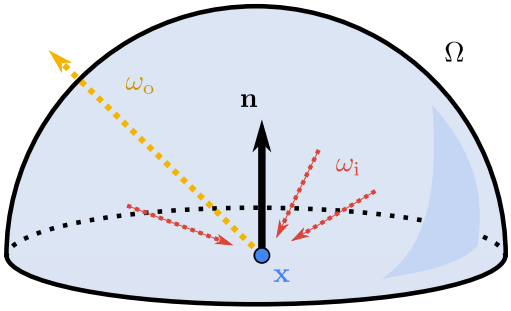


Figure 6.1: Ω is the unit hemisphere centered at position **x** around the surface normal **n**.

the surface normal **n** containing all possible values of ω_i) of the bidirectional reflectance distribution function (BDRF) f_r, which is the proportion of light reflected from ω_i to ω_o at position **x**, the spectral radiance L_i coming towards **x** from ω_i and the weakening factor of outward irradiance due to incident angle. In other words, the equation simply describes the transport intensity of light along a direction as the sum of the light emitted along that direction and the total light intensity which is scattered towards that direction coming from every other possible direction. The rendering equation is spatially homogeneous and linear. Even though the equation is quite general, it does not cover every single aspect of light transport and scattering in full. For instance, it includes neither refraction phenomena nor subsurface scattering effects. Respective extensions to model these phenomena resulted in the bidirectional scattering distribution function (BSDF), comprising the bidirectional reflectance distribution function (BRDF) and the bidirectional transmittance distribution function (BTDF), and others. Note, however, that it is recursive, since in order to evaluate the observed radiance L_o, one must know the incoming radiance L_i toward position **x**. This makes it actually astronomically expensive to evaluate reliably and exhaustively without using approximations like Monte Carlo methods. Hence, in practical applications such approximations are always used.

6.1.3 The Rendering Pipeline

The *rendering pipeline*, also known as graphics pipeline, describes the conceptual steps that must be followed in order to render a 3D scene to a 2D display or monitor. The steps required in the rendering process depend on both the software and the hardware available, and there is no universal rendering pipeline that fits all cases. Tracing every single photon in a scene and computing its effects is impractical and completely out of the scope of even the most powerful computing systems on Earth, so many rendering algorithms based on different approximation techniques have been researched extensively. The pro-

cesses by which the final image is produced may vary significantly from one to the other. We can distinguish between the following families of algorithms:

Rasterization. *Rasterization* is the process of geometrically projecting objects in a scene to an image plane. It uses a primitive-by-primitive approach, based on the determination of the affected pixels for each primitive. It typically uses very coarse, non-physically based techniques to approximate illumination, even though the boundary between rasterization-based graphics and higher-fidelity methods has become fuzzier lately. Nowadays, rasterization is sometimes combined with localized ray-marching and ray-tracing techniques aided by hardware in order to ramp up the graphical fidelity of scenes. Rasterization is still the most common method used to produce interactive visualization.

Ray casting. *Ray casting* algorithms parse the geometry of the scene pixel-by-pixel and line-by-line from a specific point of view. Rays are cast outwards from this point of view and, if objects are intersected, the respective color value is evaluated using several methods. The optical laws used in these methods are usually quite basic, but the rendering speed can be orders of magnitude faster than ray tracing.

Ray tracing. *Ray tracing* is quite similar to ray casting, but it aims at a greater realism by more accurately simulating the flow of light. While ray casting computes pixel colors on collision, ray tracing recursively traces additional light rays that sample the radiance incident on the point that the ray hit. Ray tracing is typically used to approximate the solution to the rendering equation by applying Monte Carlo methods.

In this section, we describe the rasterization rendering pipeline only, since it is the most common method to produce interactive graphics on common hardware. The typical scene that is to be rendered as 3D graphics is composed of many separate objects that fully define it. The geometry of these objects is represented by a list of vertices usually containing additional structural and geometrical information like normal vectors, indices or texture coordinates. The primitive type also needs to be specified. It indicates how the vertices are connected to each other in order to form shapes (points, lines, triangles, etc.). These vertex data are prepared in the CPU and conform, together with associated supporting and texture data, one of the most important input data and starting point of the rendering pipeline.

The rendering pipeline has three main stages: the application stage, the vertex transformation stage and the rasterization stage.

Application Stage

This step is concerned with the preprocessing and manipulation of the rendering data into formats suitable to be rendered. This stage is carried out by the application in the CPU. During this stage, changes are made to the scene by means of input devices in the form of camera adjustments or input data modifications. The new scene is then passed to the next step.

Vertex Transformation Stage

Geometrical data in three-dimensional coordinates are passed to the graphics hardware. This stage transforms these data into geometry that can be drawn into a two-dimensional viewport via a set of transformation matrices that must be prepared beforehand, as illustrated in Figure 6.2. Specifically, the coordinates are usually sent in in model space (local coordinates). These are then transformed to world space by means of the *model transformation* $\mathbf{M}_{\text{model}}$. Then, world coordinates are converted into the camera space by means of the *view transformation* $\mathbf{M}_{\text{view}}$. Then, these are projected into the homogeneous *clip space* using the *projection transformation* $\mathbf{M}_{\text{proj}}$ to be finally transformed to screen space using the *viewport transformation*. The transformation from object coordinates $\mathbf{p}_{\text{obj}}$ to clip coordinates $\mathbf{p}_{\text{clip}}$ is described with the given equation:

$$\mathbf{p}_{\text{clip}} = \mathbf{M}_{\text{proj}} \cdot \mathbf{M}_{\text{view}} \cdot \mathbf{M}_{\text{model}} \cdot \mathbf{p}_{\text{obj}}. \tag{6.2}$$

After that, only the viewport transformation needs to be applied to get the coordinates in screen space. All of these matrices are typically set up in the application stage and sent to the GPU, where user shader code is in charge of applying them to the vertices. The shader code that works on vertices and transforms them is known as *vertex shader*.

Rasterization Stage

Once the model has been clipped and transformed into screen or window space, the GPU needs to determine what pixels in the viewport are 'hit' by which graphics primitives. Rasterization is the process of filling the horizontal spans of pixels belonging to a primitive. All data items associated with a vertex (position in screen space, normal vector, texture coordinates, color, etc.) are interpolated for each pixel, conforming, together with that pixel's calculated depth value, what is called a *fragment*.

The steps involved in the rasterization stage are outlined in Figure 6.3. First, an application may decide whether to apply *face culling* or not. In the face culling process, applied to polygonal primitives, the faces facing away from or towards the camera are eliminated.

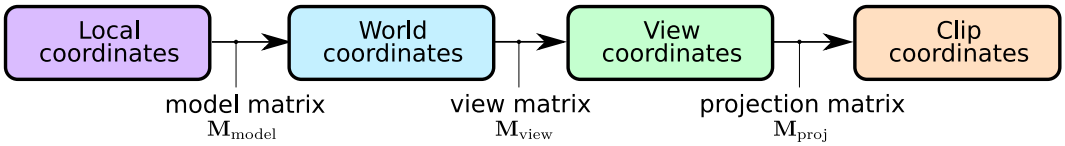


Figure 6.2: Local object coordinates are converted to world coordinates via the model matrix. The view matrix is applied to world coordinates to get view coordinates, which combined with the projection matrix yield clip space coordinates.

This step is usually configurable via the graphics API. Then, rasterization is applied on the remaining primitives, producing the fragments. Another user program known as the *fragment shader* is in charge of computing color, depth and possibly other values for each fragment. Typically, in a per-vertex-lighting scenario, the fragment color is simply the interpolated value from each of the vertices of the primitive, and the depth value is their interpolated depth at the fragment's position. However, this step can be as involved and complicated as necessary, with different levels of lighting approximation being traded off to performance. Some models, like the Phong shading model, are very simple and physically inaccurate. Others, like physically-based rendering (PBR) attempt to model the flow of light realistically at the expense of computing power.

In a final stage, additional configurable fragment operations are applied to the remaining fragments. Most of these operations determine whether a fragment is to be drawn to the viewport or discarded altogether. Examples are the pixel ownership test, which checks whether a fragment lies in the region of the viewport that is currently visible, or the depth test, which compares the depth value of the current fragment with the depth value already stored in the depth buffer at the fragment's location. Some of these operations can also be moved into the fragment shading stage. The very final fragment operation combines the current fragment values with the values of the fragment at the same position in the color buffer, and is known as *blending*. We cover it in detail in Section 6.1.7.

6.1.4 Coordinate Systems

As we have seen in Section 6.1.3, there are several reference systems involved in the series of transformations that bring an object from its local coordinate system to *normalized device coordinates* (NDC), where the whole geometry is between -1.0 and 1.0 in all three dimensions, to the final screen or window coordinates. Transforming coordinates to NDC and screen coordinates is typically accomplished in a gradual fashion, step-by-step, where we transform the vertex coordinates to several distinct coordinate systems. The reasoning behind this many coordinate systems is that some operations make more sense, are easier to process or simpler to code in certain coordinate systems. We can dis-

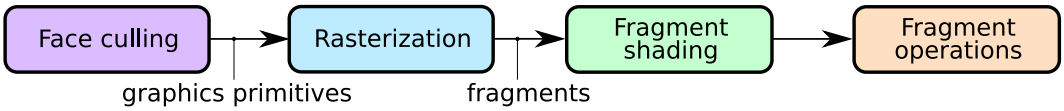


Figure 6.3: First, face culling optionally eliminates forward or backward-facing faces. Then, the remaining graphics primitives undergo rasterization and are converted to fragments. These fragments are shaded in the fragment shader and finally undergo some configurable fragment operations.

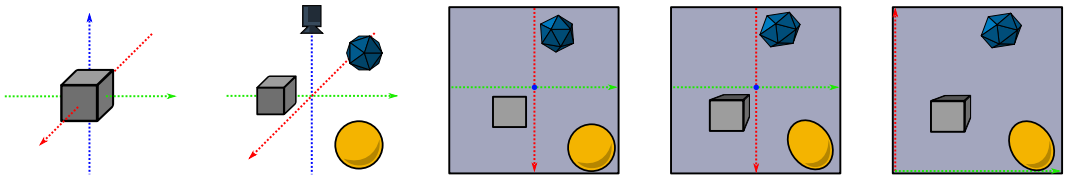
tinguish between five different coordinate systems (Figure 6.4): Local (or object) space, world space, view space, clip space and screen space. As we have seen in the geometry stage section, we employ different matrices to transform from one coordinate system to another. These matrices are typically produced and prepared in the CPU, and sent to the GPU as accessory data items to the rendering data themselves.

Local space. The coordinate system that is local to every object is known as local or object space. 3D objects and models are defined, when they are being created, in this space. Typically, the model is centered around the coordinate system origin $[0, 0, 0]$.

World space. The global coordinate system that all objects in a scene share is known as world space. It is application-dependent and is used to position the objects and elements of a scene relative to each other. Each object has its own transform matrix, called the model matrix $\mathbf{M}_{\text{model}}$, which transforms the object's local coordinates to world coordinates. This matrix is used to place the object in the world with a given position, orientation and size. The application can, for instance, make the object move simply by manipulating the translation of this transformation matrix.

View space. The coordinate system centered on the camera and aligned with its direction and up vectors is known as view space. The view space contains the coordinates as seen from the camera's point of view. This is typically accomplished using a combination of translations and rotations stored inside the view matrix $\mathbf{M}_{\text{view}}$.

Clip space. This space contains all the *visible* coordinates normalized in the $[-1, 1]$ range. Vertices that are not visible due to falling outside of the *viewing volume* have coordinates outside this range. To convert view coordinates to clip coordinates the projection matrix $\mathbf{M}_{\text{proj}}$ is used. This matrix specifies the range of coordinates in each dimension. The process by which this is determined depends on the type of projection used. The two most common projections are the perspective projection, typically used in scenes representing a three-dimensional space, and orthographic projection, commonly used in two-



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

Figure 6.4: Detailed look at the most common reference systems involved in the geometry transformation of a scene. Every vertex of every object starts in local coordinates (a), which are then translated to world coordinates (b) using the model matrix $\mathbf{M}_{\text{model}}$. World coordinates are transformed to view coordinates (c), which corresponds to the scene viewed from the perspective of the camera (at the top looking down), with the camera's view matrix $\mathbf{M}_{\text{view}}$. These coordinates are then normalized and projected to clip space (d) using the camera projection matrix $\mathbf{M}_{\text{proj}}$. Finally, the viewport transformation converts from clip coordinates to screen space (e) so that the coordinates in $[-1, 1]$ are mapped to the range defined by the viewport.

dimensional scenes. The view volume contains all objects and entities that are visible, and is known as *view frustum*. Once all vertices are transformed to clip space, a perspective division operation is carried out. In this operation, the position components are divided by the homogeneous component in order to transform 4D clip space coordinates to 3D NDC. This is automatically performed at the end of the vertex transformation step.

Screen space. Also known as window space, screen space coordinates are simply clip coordinates mapped to the range defined by the viewport. This is usually a window in a windowing system, but it can also span the full display, or even multiple displays, depending on the setup. Screen coordinates are typically in the range $[0, 1]$.

6.1.5 Projections and Frustum

The projection matrix $\mathbf{M}_{\text{proj}}$ is applied to transform from view coordinates to clip coordinates. It defines the frustum, which is the view volume containing everything that will be rendered. Objects and geometry outside the frustum are culled and discarded. The projection matrix usually takes two froms, where each defines its own unique frustum: orthographic or perspective.

Orthographic projection. This projection defines a cube-like frustum defined by its height, width and the near and far distances determining the near and far clipping planes respectively. This projection maps coordinates to the 2D plane of the display directly, and it is used mainly for 2D applications. Once projected, all objects have the same size in the clipping plane regardless of their position in the z-space.

Perspective projection. The perspective projection takes into account the phenomena where far away objects appear smaller and close-by objects are larger. This effect is called perspective. The frustum in a perspective matrix (Figure 6.5) is defined by a field of view angle α and a near and far distances, z_n and z_f, defining the near and far clipping planes. The perspective projection matrix maps a given frustum range to clip space and also manipulates the w value of each vertex coordinate in such a way that the further away a vertex coordinate is from the viewer, the higher the w becomes. Coordinates in clip space are in the $[-w, w]$ range, and anything outside is clipped. Coordinates in clip space are applied the perspective division by w, mapping them to $[-1, 1]$. Each component the vertex coordinate is divided by its w, giving smaller vertex coordinates the further away the vertex is from the camera.

6.1.6 Depth Testing

Depth testing is an essential fragment operation in the graphics pipeline. It prevents geometry to be rendered out-of-order in the z-space, and it does so without sorting it beforehand. The depth test uses the depth buffer, which has the same size as the color buffer, and stores depth information per fragment. Typically, the precision of the depth buffer is 16-, 24- or 32-bit *floating-point* numbers.

The depth test is carried out as one of the fragment operations in the final stage after the fragment shading. In this test, the depth value of each fragment is compared against the depth value stored at the fragment's position in the depth buffer. Typically, the test passes (i.e., the fragment is not discarded) when the depth value of the fragment is less than the one found in the depth buffer. This condition, however, can be manipulated in most current graphics APIs. When the test passes, the depth value at the fragment's location in the depth buffer is updated with the fragment's depth value, and the fragment continues along the pipeline.

The values in the depth buffer are normalized to the z range in clip coordinates. However, the z-values in view space can be any value between the frustum's near and far planes. The depth test function transforms from view-space z-values to clip-space normalized coordinates. The obvious way of doing so is using the linear function $f(z)$, defined as

$$f(z) = \frac{z - z_n}{z_f - z_n}, \tag{6.3}$$

where z is the z-value in view-space and z_n and z_f are the distances to the near and far clipping planes of the frustum in view-space. In practice a linear depth buffer like this

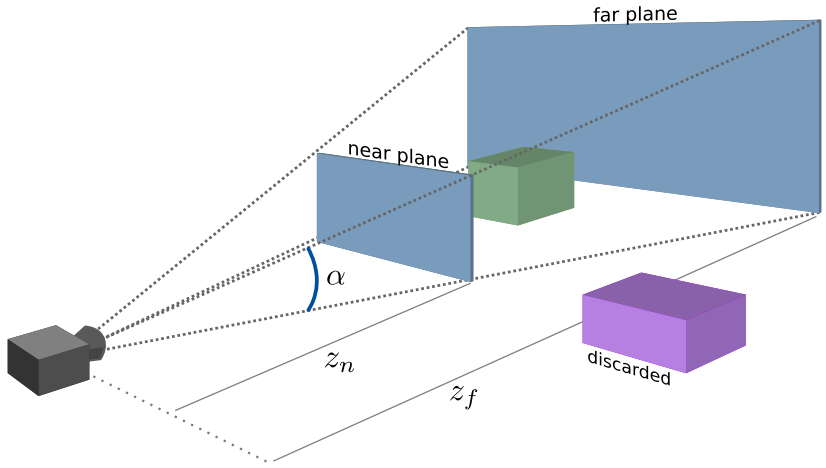


Figure 6.5: The view volume, or frustum, is the volume between the near and far plane as determined from the camera. In the image, the perspective camera defines the frustum using the field of view angle α, the near distance z_n and the far distance z_f. The inside of the volume is mapped to $[0, 1]$. The yellow object has all vertices in this range, so it is located inside the frustum and its coordinates get transformed to view and clip spaces. The magenta object's coordinates are out of this range, and are discarded in the *frustum culling* stage.

is almost never used because of perspective projection properties. Usually, a non-linear function proportional to $1/z$ is a more sensible choice,

$$f(z) = \frac{1/z - 1/z_n}{1/z_f - 1/z_n}. \tag{6.4}$$

This provides higher depth precision closer to the near plane at the expense of lower precision at higher distances. This function performs well in small scenes. In them, depth values are greatly dominated by the small z-values, providing a large depth precision for objects close by. Note that this function yields non-linear values in clip-space. Practically, the function is embedded within the projection matrix $\mathbf{M}_{\text{proj}}$, so when coordinates are transformed from view space to clip space, and then to screen space, the non-linear equation is applied.

6.1.7 Blending

The very final fragment operation is known as *blending*. This operation calculates a new color from the fragment's final color and the color already stored in the color buffer at the fragment's location. The blending equation is quite powerful, and most graphics APIs

allow it to be completely configured. It describes how to weight and combine the source and destination colors to obtain the final resulting color:

$$\bar{C}_{result} = \bar{C}_{source} \cdot \bar{F}_{source} + \bar{C}_{dest} \cdot \bar{F}_{dest}. \tag{6.5}$$

Here, $\bar{C}_{source}$ is the source color, the output of the fragment shader. $\bar{C}_{dest}$ is the destination color, which is the color currently stored at the fragment's location in the *color buffer*. $\bar{F}_{source}$ and $\bar{F}_{dest}$ are the source and destination color factor values. These are used to weight the impact of the source and destination colors in the result, respectively. These factors are what makes this equation so powerful. Setting them to the source alpha (transparency value) and one minus the source alpha we get the standard alpha blending, which simulates transparency and takes into account the order of the fragments. If we set them both to one, we get additive blending. Most graphics APIs allow for a full customization of both the shape of the blending equation and the factor values.

6.1.8 Frame Buffers, Textures, and Post-processing

Frame buffer objects are a combination of a color buffer, a depth buffer and a stencil buffer. These buffers reside in GPU memory and can be created and destroyed at will. They are typically used as render buffers as holders of the output of the render process. Frame buffers are necessary for off-screen rendering, where a scene is rendered but not presented on screen. For example, the result of the shadow map pass is a scene with the depth information from the point of view of each light. This is rendered to an off-screen frame buffer and used as input in later render stages. The color and depth buffers of a frame buffer can be sent into shaders in the form of texture attachments. Rendering a scene to a frame buffer with a texture attachment allows us to use that texture as input in our shaders to perform post-processing. Post-processing is a group of operations that are performed usually at the end of the render process and act on the final image or render. There are countless effects and operations that can be implemented via post-processing, such as bloom, lens flare, anti-aliasing, cell-shading, screen-space reflections and much more. Any effect whose name starts with 'screen-space' is implemented as a post-processing operation.

6.2 Interactivity and Performance

There are several techniques to improve the performance and interactivity of graphics applications by making them produce frames faster. These techniques can be categorized

into two big groups: culling strategies and spatial structures. Even though there may be some overlap between them, we approach them separately in the present work.

6.2.1 Culling strategies

In three-dimensional rendering of complex scenes typically only a tiny subset of the scene is visible from a given location at a given time. The determination of this subset is what is known as the *visibility problem*, and mainly has two solutions: visible surface determination and hidden surface determination, also known as culling. While the former determines the visible set without checking all the geometry, the latter works from the full scene and removes all invisible parts to approximate the visible set.

Culling strategies attempt to eliminate as many elements as possible from a scene in the early stages of the rendering pipeline in order to render only the absolutely essential ones. The most important culling algorithms are discussed below.

View-frustum culling. As noted in Section 6.1.5, a virtual camera is defined by a projection matrix, which encodes the information of the viewing frustum. The viewing frustum is the volume that contains all visible objects in a scene, and is defined by the six planes front, back, top, bottom, left and right, which together form a cut pyramid. View-frustum culling refers to the process of discarding the geometry elements that fall outside this volume. The naive version of this process tests every object against the six planes, leading to an asymptotic complexity of $O(n)$. However, we have seen that in the clip space the coordinates are normalized device coordinates after the perspective division, in $[-1,1]$, so this check becomes a trivial comparison and is quite fast and straightforward.

Back-face culling. A model object is composed of vertices and faces. In average, roughly half of the faces are visible from the camera at a particular moment in time. This means that half of the faces are not visible at all, so there is no need to process them. Back-face culling applies only to polygonal graphics primitives and is typically performed before the rasterization process begins. It looks for faces which are facing away from the camera and discards them. These faces correspond to the unseen far side of the model, so they can (most times) be safely discarded. In order to detect the back-facing faces the *winding order* of the vertex data is used. The winding order is the order in which the vertices of a triangle are defined when the model is created. This order can be clockwise or counter-clockwise. By default, triangles defined with counter-clockwise vertices are processed as front-facing triangles. Since the actual winding order is calculated at the rasterization stage after the vertex transformations have already completed, the vertices are

seen from the viewer's point of view. In this case, all we have to do to cull back-facing triangles is check their winding order from our perspective. If it is reversed, the primitive can be safely discarded.

Occlusion culling. Occlusion culling is the process of removing objects that are hidden by other objects in a scene from the viewpoint. Frustum culling and back-face culling are usually not enough, especially in dense, complex scenes. The third culling strategy, occlusion culling, attempts to identify the visible parts of the scene, thus reducing the number of primitives rendered. Occlusion culling is often cell-based. The most prominent technique to is based on binary space partitioning (BSP), where space is split with a plane to two half-spaces, which are again recursively split. This can be used to force a strict back-to-front ordering [Fol+90]. Occlusion culling is typically based on the potentially visible set (PVS) [Tel92], which evaluates for each cell a set of visible cells, which are then used at rendering time to discard non-visible cells beforehand. PVS-based culling is conservative, as it always overestimates the set of visible cells in order to avoid artifacts. Additionally, since the PVS determination can be quite expensive, in some cases the sets are precomputed. Occlusion culling algorithms can be typically classified depending on the following many criteria:

Online vs. offline. A major distinction is whether the technique precomputes and stores visibility data beforehand or whether it dynamically evaluates and computes it during the course of its run time.

Image vs. object space. Object-based methods use the geometry of the whole object to determine visibility. Image-based methods, in contrast, operate on the visual representation of an object when broken into fragments in the rasterization stage.

Conservative vs. approximate. Conservative techniques overestimate the visible set, guaranteeing the fidelity of the rendered scene at the expense of speed. Approximate techniques may underestimate the visible set, resulting in rendering artifacts and popping.

Continuous vs. discrete. Continuous methods determine the visibility in all view directions in the image. In contrast, discrete methods determine the visibility only for a subset of view directions, e.g., a single direction for each pixel in the image.

Individual vs. global. Individual methods determine whether objects are fully occluded by other objects individually. Global methods can also determine whether an object is fully occluded by the combined geometries of two or more other objects from a viewpoint.

Coherence use. Algorithms can exploit different types of coherence, like spatial (the visible parts of a scene tend to consist of a group of compact sets or regions), ray-space

(similar sets of rays tend to have the same visibility) or temporal (visibility at two successive moments tends to be coherent despite small changes in the scene).

PVS-based arbitrary geometry occlusion culling. Recently, PVS-based occlusion culling algorithms for more general environments have been proposed. These algorithms follow the PVS principle in that they try to find a tight overestimation of the true PVS for a given cell. The algorithm by Schaufler et al. [Sch+00] first discretizes the scene by building an octree containing empty, boundary and opaque nodes. Then, they find opaque (solid) octree nodes, and, for all view nodes, check what portion of space the node definitely occludes with respect to any position in the view node. Only the nodes not hidden are inserted into the PVS for the cell node. The approach is also well suited for finding the definitely hidden parts of terrain in terrain rendering. An alternative algorithm to the problem by Durand et al. [Dur+00] introduces extended projections to find the PVS for viewing cells. Both presented algorithms can handle occluder fusion. This means that two or more occluders positioned in the same screen region from the point of view of the observer can hide an object even if none of them could do it alone.

Contribution culling. *Contribution culling* is an example of a non-conservative method, as it discards objects if their screen projection is small, introducing artifacts such as popping. The term "aggressive culling" is sometimes used to refer to this family of non-conservative methods [AM04]. In other words, contribution culling is the process of removing objects that do not contribute significantly to the final image due to their small size or large distance. This form of culling is still very often used in some applications. To smooth out the transitions and avoid popping, distance fog is sometimes added. Another common technique is to replace objects with their lower-fidelity versions with a lower *level-of-detail* (LOD) when they are sufficiently far from the viewpoint.

6.2.2 Spatial Data Structures

A solution to the visibility problem can be conservatively approximated by using spatial structures that exploit spatial and temporal coherence, like the algorithm by Coorg et al. [CT97]. There are various ways to store and represent the objects of a scene in a computer system, depending on the final aim: point clouds, volume data, edge lists or mathematical analytical curves and surfaces. Here, as well as in the rest of this introduction, we focus on the point-based and triangular geometry often used to approximate surfaces. The spatial structures we discuss are well suited to store such geometry, even though in some cases may also accommodate other representations. Storing scene data in a linear,

naive fashion is unsuited for all but the most basic use cases, as there is no efficient way to access the needed spatial information to perform the culling operations.

Since there are many possible structures, their unique properties must be compared and balanced with the needs of a particular application. There is the spatial complexity (in memory and disk), the generation and update complexity, and the code complexity necessary to support its use. Typically, due to its nature, only hierarchical spatial structures are well suited for most uses; if an object is not visible, all of its children are also not visible. Most of the structures discussed here are complimentary. This means that they can be combined to approach different problems within a single application.system.

Non-Hierarchical Grids

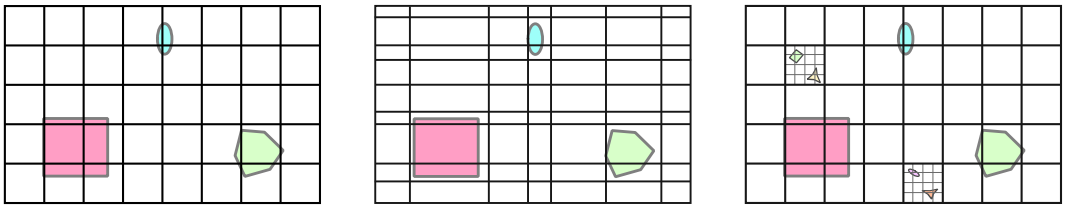
In *non-hierarchical grids*, the space is divided only once using a unique structure for the entire scene. There are no further recursive sub-divisions, so their use, especially in scenes with large distance ranges, is quite limited. The most clear candidates are uniform and non-uniform grids.

Uniform grids. Uniform (regular) grids represent the simplest conceivable way to divide space. In these, the space is divided into cells of equal size, covering the whole world. Such partitioning suffers from both performance and scalability problems when used for complex scenes with large environments, or when the object density is far from uniform across the world. Advantages of this method are the simplicity of set-up and use. The use of a uniform grid for dynamic scene occlusion culling is explored by Batagelo et al. [BS02].

Non-uniform grids. Non-uniform, *irregular grids* trade higher a generating cost for a higher efficiency at accommodating the geometry and objects in a scene. Non-uniform grids have splitting planes which are still axis-aligned, but they can be position arbitrarily along the axes. These grids are typically made up of non-equally-sized cells, and their configuration is closely coupled with the scene at a specific time. In this case, whenever there are variations in the scene the grid may need to be re-generated from scratch. Irregular grids are better at dealing with anisotropic scenes where the object density varies from region to region.

Hierarchical Grids

Hierarchical grids are able to subdivide space recursively at each cell when needed, solving the problematic large scenes which span over very large distances.



(a) (b) (c)

Figure 6.6: An example of a uniform grid (a), a non-uniform grid (b), and a recursive grid (c).

Recursive grids. *Recursive grids* are still quite simple. They simply bring the concept of recursivity into uniform grids. Their construction process is as follows: start by setting up a uniform grid on the scene. Then for each cell whose object count is higher than a defined threshold, recursively create a new grid with a smaller scale, and repeat. The concept is similar to the BSP or the octree, but in this case a cell is subdivided into an arbitrary number of cells instead of two or eight.

Quadtrees and Octrees

Quadtrees and *octrees* are actually the same data structure, but while quadtrees are two-dimensional, octrees are three-dimensional. An octree (Figure 6.7) is a data structure that subdivides space hierarchically and recursively in which each node, also called octant, has exactly eight (four for quadtrees) children nodes. They have been shown to be very effective at managing large 3D point clouds [EBN11]. The root node is set up so that it encloses all the objects in the set, and is subdivided further in accordance to defined density rules. Typically, octrees are used as a supporting structure to a level-of-detail method, and store coarser version of objects and entities in higher-level nodes, and finer, more detailed ones in the bottom nodes. Since the octree organizes space partitions hierarchically, its traversal can be easily implemented using a simple visibility condition based on the solid angle of the octant, or on the distance from the viewpoint to the octant and its size. Depth-first traversal and access operations on octrees have an asymptotic complexity of $O(\log n)$.

However, octrees are not particularly well suited to dynamic scenes with fast-moving objects. While local re-generations and re-balancing operations are possible, they are not perfect and usually require of a full rebuilding after a certain time. There are several variants of the octree, like the octree-R, where the splitting planes inside subdivided octants are arbitrarily positioned, or the loose octree, which is a regular octree where all the octants are subdivided down to a certain depth.

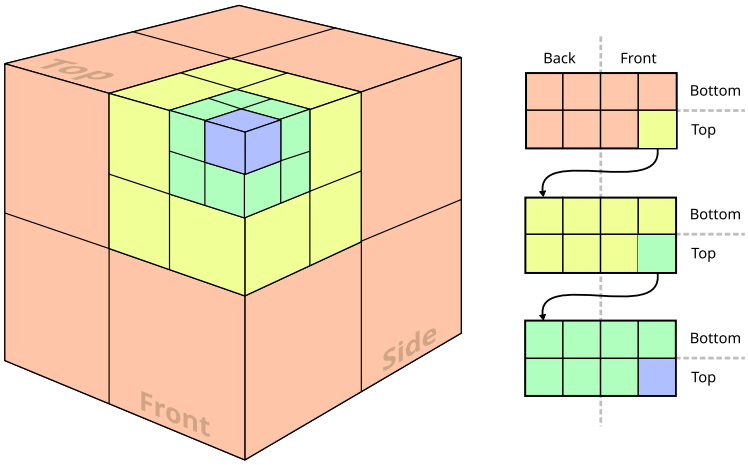


Figure 6.7: Example of an octree with four levels, 0 to 3. At the root level (salmon) only the front-top-right node is subdivided (yellow), belonging to the first level. The same front-top-right node is further subdivided into the second level (green). This, in its turn, is subdivided into the third level, where the blue node belongs.

BSP Trees

Binary search partitioning (BSP) is a technique developed by Schumacker et al. [Sch69], based on recursively subdividing space into two convex sets by using codimension-one manifolds. In the 3D case, these manifolds are two-dimensional planes. The subdivision continues until one or more requirements are met. This partitioning scheme is typically represented by a data structure known as a BSP tree, which can be seen as a generalization of other spatial data structures like the k-d tree Section 6.2.2.

This technique was initially used in the context of 3D computer graphics because it provides effortless back-to-front sorting of primitives and polygons, enabling the efficient integration and usage of the painter's algorithm [FKN80]. In this algorithm, polygons are rendered in a descending order of distance to the viewpoint, back to front, leading to far away polygons being rendered first and closer ones being rendered last. This algorithm requires a time-consuming sorting of the polygons, and is unable to render correctly parts where two or more polygons overlap. Even though nowadays the usage of the z-buffer renders the painter's algorithm unnecessary, BSP trees are still widely used.

BSP trees are typically quite expensive to set up, construct and update because of the expensive search of the best splitting plane at each division, and are not particularly well-suited to dynamic scenes. The time complexity for creating a BSP tree is $O(n^2 \log n)$, where n is the number of objects in the scene.

k-d trees

A *k*-dimensional tree (*k-d tree*) is a special version of the BSP tree that only uses axis-aligned splitting planes. In contrast, the splitting planes in BSP trees used to speed up the painter's algorithm are polygon-aligned. This data structure organizes points in a k-dimensional space, and is used in many applications, like multidimensional space searches or creating point clouds.

In each node of a *k*-d tree is a *k*-dimensional point. Each non-leaf node generates a splitting codimension-one plane that divides the space into two parts known as half-spaces. Points to each side of this plane are represented by either children subtrees.

There are very many variations of the *k*-d tree. For instance, the *implicit k*-d tree is defined by an implicit splitting function, the *min/max k*-d tree associates a minimum and maximum value to each node, and the *relaxed k*-d tree uses arbitrary discriminants for each node.

6.2.3 Level-of-Detail

Most of the material we have seen in the previous sections can be applied to implement level-of-detail (LOD) into an application. LOD typically refers to the complexity of a 3D model representation of an object, but it can also be generalized to culling strategies and visibility determination. All in all, we can distinguish between two separate categories: object LOD and view-dependent LOD.

Object LOD. Object LOD has entire objects as the principal subject of the levels of detail. For example, several versions of the same 3D model for an object with increasing fidelity and complexity can be used depending on a heuristic function. This can be as simple as a set of conditions based on particular physical quantities like distance or solid angle. This scenario is an example of *discrete level-of-detail* (DLOD), where several separate versions of an asset are created and exchanged in full when needed. This discrete nature of the levels usually results in visible artifacts like popping when the exchange happens. These artifacts can be mitigated by smoothly morphing between levels. In contrast, *continuous level-of-detail* (CLOD) uses a structure that contains a variable spectrum of detail. This structure can be queried to provide the appropriate level-of-detail in each situation.

View-dependent LOD. In *view-dependent LOD* the principal subject of the levels of detail is not an entire object, but the geometry and polygons themselves. These methods allow detail to be dynamically added and subtracted from parts of a polygon mesh based on control parameters which are typically view-dependent. This is especially well-suited

for large, complex objects that span over large areas. In such cases, it may make sense to use the highest-detail level for areas close to the viewpoint, while keeping distant parts coarser. The most obvious candidate to view-dependent LOD is *tessellation*, which is already supported by all the major graphics APIs. Tessellation is the process of dynamically subdividing a polygonal mesh based on some rules.

6.3 Floating-Point Precision

There are several mechanisms by which numbers can be represented in computer systems. Nowadays, most GPUs are optimized for *floating-point representation*, which enables the representation of real numbers as an approximation that trades off range and precision. In this part we focus on the IEEE 754 [IEE19] standard introduced in 1985, which is widely adopted and used.

In floating-point arithmetic, numbers have a base β, which is always assumed to be even, and a precision p. Numbers are represented with a fixed number of bits, typically 32 or 64. These bits are distributed between the *significand* (also known as *mantissa*) and the exponent. Each of these is represented with a fix number of bits, and together they conform the floating-point representation

$$\pm d.dd\ldots d \times \beta^e, \tag{6.6}$$

where $d.dd\ldots d$ is the significand, β is the base and e is the exponent. More precisely, $\pm d_0.d_1d_2\ldots d_{p-1} \times \beta^e$ represents the number

$$\pm\left(d_0 + d_1\beta^{-1} + \cdots + d_{p-1}\beta^{-(p-1)}\right)\beta^e, 0 \leq d_i < \beta. \tag{6.7}$$

Two additional parameters, e_{min} and e_{max}, represent the minimum and maximum values for the exponent, and depend on the number of bits assigned to it. These, together with p and β, determine the representable range.

The term floating point indicates that the decimal point (also known as radix point) can "float", or move around and be placed anywhere with respect to the significant digits of the number by means of the exponent, in a representation similar to scientific notation. The problematic stems from the fact that Equation 6.6 represents a finite set. The distribution of this finite set in $\mathbb{R}$ is what typically produces loss of precision when representing large enough numbers.

If $\beta = 10$ and $p = 3$, then the number 0.1 can be represented exactly as 1.00×10^{-1}. However, if $\beta = 2$ and $p = 24$, it cannot be represented exactly. The approximate representation $1.10011001100110011001101 \times 2^{-4}$ is then used.

A real number may not be exactly representable as a floating-point number due to two main reasons. The first one is outlined in the 0.1 example above, where the real number lies strictly between two floating-point numbers and neither of them represents it exactly. The second one happens when the number is out of range, i.e., its absolute value is either larger than $\beta\beta^{e_{max}}$ or smaller than $\beta\beta^{e_{min}}$. When this happens, IEEE 754 defines two special "infinity" values $\pm\infty$. Precision issues in graphical representation systems have their grounds on the first case.

Intuitively, one can think of this floating-point representation as a set of buckets laid out in the real number space containing each the same number of items. The number of items that each bucket holds is the same, and depends on p, or the number of bits used for the mantissa. Each of this items is a single real number that can be represented exactly. The size of the buckets, i.e., the range of real numbers it covers, varies depending on how far away from the origin, zero, the bucket is. Close to the origin, the buckets cover a very short range, so lots of very close-by real numbers can be represented. When the absolute value of the exponent increases, i.e., we move away from the origin in either direction, the buckets gradually cover larger and larger ranges, resulting in the *representable* numbers being further apart from each other. Since only a small subset of exact real numbers can be represented with this system, a rounding mechanism is in place. In the case of IEEE 754, the rule is to round to the nearest value, and in the case of a tie, round to whichever value is even.

6.3.1 Relative Errors

Since rounding is an essential part in the floating-point representation process, measuring the errors is important. In general, if the floating-point number $d.dd\ldots d \times \beta^e$ represents z, then it is in error by

$$|d.dd\ldots d - (z/\beta^e)|\beta^{p-1} \tag{6.8}$$

units in the last place. As an example, if the real number 0.0314159 is represented with 3.14×10^{-2}, then it is in error by 0.159 units in the last place. Note that the errors increase as the represented numbers get larger.

6.3.2 Catastrophic Cancellation

As a result of the formalization of floating-point representation, the relative error committed using floating-point arithmetic when subtracting two large and nearby quantities can be very large. Therefore, the evaluation of an expression containing an effective subtraction (explicit or implicit in the signs of the operators) may result in a large enough relative error so that all the digits are meaningless. When subtracting nearby operands, the most significant digits match and cancel each other, leaving only the less significant digits as a result. This phenomena is known as *cancellation*. There are two types: *catastrophic* and *benign*.

Catastrophic cancellation occurs when the operands are subject to rounding errors. In this case, when they are subtracted cancellation may cause many significant digits to disappear, leaving behind only digits polluted by rounding error. This happens in a major degree the larger and more similar the operands are. In contrast, benign cancellation occurs when subtracting exactly known quantities, i.e., quantities that have no rounding error. In this case, the difference has a small relative error and no harm is done.

We find, therefore, under this orderly arrangement, a wonderful symmetry in the universe, and a definite relation of harmony in the motion and magnitude of the orbs, of a kind that is not possible to obtain in any other way.

— Johannes Kepler

7 Gaia Sky

These are exciting times for astronomy and *astrometry* research. The Gaia satellite, launched back in December 2013, has been measuring since July 2014 the star positions and proper motions of roughly one percent of the stellar content of our Milky Way, a spiral galaxy populated with 100–300 billion stars. It has two fields of view, separated by an angle of 106.5°, which are projected on the same focal plane by a complex system of mirrors. With 106 CCD sensors and about a billion pixels, it is the largest camera to ever be flown into space. DPAC, the Data Processing and Analysis Consortium, is the multinational endeavor with institutions from more than 20 countries responsible for data processing and construction of the final Gaia catalog. However, the release of this catalog is rolled out in an incremental fashion. The Gaia Data Release 1 (DR1), published in September 2016, contains a catalog of over 1.1 billion 2D star positions. Additionally, more than two million parallaxes and proper motions (angular velocities of stars in the sky, as observed from Gaia) could be derived using cross-matching techniques with earlier catalogs, such as Tycho-2 [Høg+00] and Hipparcos [Per+97], enabling this part of the star map to raise into the third dimension, as well as 250,000 radial velocities by cross-matching with RAVE [RAV+17] data. The second data release, DR2, was released April 25, 2018. Based on more than 22 months of data collection, it represents the largest leap the Gaia catalog will ever make. It contains the positions, parallaxes, proper motions, magnitudes, and colors of more than 1.3 billion stars. Additionally, it offers more than 7 million radial velocities, half a million variable stars, and about 14 thousand asteroid orbits. It can be considered the first real Gaia dataset, and it is enabling astronomers to study the history, composition, and structure of our galaxy in great detail. Since then, the early third Gaia data release (eDR3) was published December 3, 2020, and represents a big leap in accuracy and a moderate leap in raw numbers when compared to DR2. In this chapter, we will use 'DR2+' to refer to any Gaia catalog containing three dimensional positions for more than a billion stars, i.e., any catalog after

Gaia's first data release. The appeal of such a mission goes well beyond astronomy, and in order to spark an interest in the laymen, clear and simple messages and attention-catching media material are usually required. In this chapter, we present a free and open-source software package called Gaia Sky (Figure 7.1), created with the main objective of delivering the Gaia catalog to related research areas and the general public, and enabling anyone to gain insight into not only the mission and its catalog, but also supporting astrometric, astronomic, and cosmological research.

The contributions of this chapter include:

- A new magnitude–space level-of-detail octree structure,
- a floating camera approach to address single-precision floating-point number accuracy issues,
- a novel depth buffer function to address high-dynamic distance ranges,
- integrated visualization of relativistic effects, and
- the representation of proper motions of stars.

7.1 Other Astronomy Visualization Frameworks

There are, of course, several tools and frameworks that have different aims and propose distinct approaches to Gaia and astronomy visualization in general. We can distinguish between two groups: 3D universe software and Gaia-tailored visualization tools.

In the Gaia visualization domain, the Gaia Archive Visualization Service [Moi+17] offers a web-based interactive visual exploration tool, that features direct access to the Gaia catalog in a visual collaborative environment based on 2D linked views. It also features a three-dimensional representation of the star catalog, albeit quite limited in functionality and scope. Topcat [Tay05] is a desktop tool for operations on astronomical catalogs and tables. It offers 2D and 3D plotting and is focused on astronomy use cases. Since version 4.2-1, it supports GBIN files, the native Gaia data format. Topcat is widely used in the larger astrophysics community. Vaex [BV18], although not Gaia-specific, was initially developed within the context of DPAC, and is a visualization and exploration tool for big tabular data with its own volume rendering package. Gaia Sky focuses on a direct spatial representation of the catalog, but all these tools can be used in conjunction with Gaia Sky via SAMP [Tay+11], a messaging protocol that enables astronomy software tools to interoperate and communicate, in order to provide additional functionality.

Several similar systems fall in the category of planetarium systems that run on full dome setups (Section 7.1.1). Other, more closely related packages to Gaia Sky are comprised by open-source, freely available 3D universe software packages that run on laptops and desktop computers. In this category, we find Open Space (Section 7.1.2), Celestia (Sec-

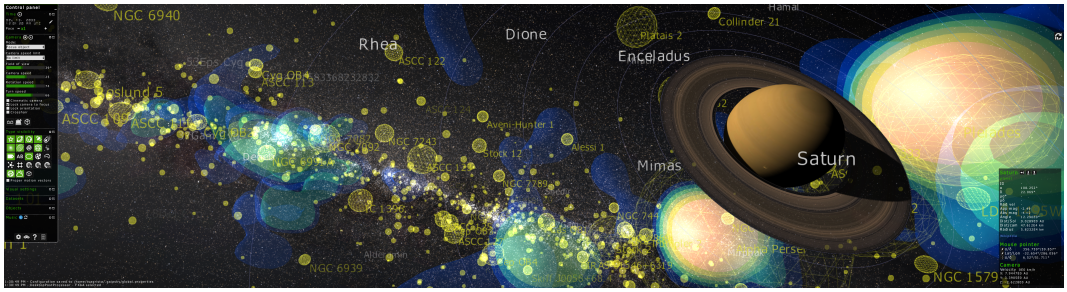


Figure 7.1: Screenshot of Gaia Sky with Saturn and some of its moons, the Milky Way Star Clusters catalog indicated by yellowish spherical meshes, semi-transparent isosurfaces of OB star densities, and stars from Gaia showing parts of the Milky Way galaxy. © 2019 IEEE.

tion 7.1.3), and Space Engine (Section 7.1.4), even though this last one is proprietary and neither open source nor free.

7.1.1 Planetarium Systems

In the realm of 3D universe software, Uniview [Kla+10], Digistar [ES], and Mitaka [4D2] are the main players in planetaria software worldwide, and offer advanced, visually appealing simulations that run on multi-projector dome systems. All of them enable a full three-dimensional representation and traversal of the known universe. Star Strider [FMJ] is a planetarium and virtual spaceship application that offers a 3D star chart and includes a representation of relativistic aberration and Doppler-shift.

7.1.2 Open Space

Open Space [Boc+17] is an open-source interactive data visualization framework designed to visualize the entire Universe. We have evaluated its latest available public version, 0.16.0, which is a beta release. The package is only available for Windows and macOS in executable form, even though it can be built from source for Linux. On the other hand, Gaia Sky is out of beta since version 1.0.0 and provides off-the-shelf builds for Linux (rpm, deb, aur, sh, flatpak), macOS (dmg), and Windows (64-bit installer).

The default dataset is downloaded automatically at startup of Open Space and contains a lot of data of the Solar System and its space missions. It implements virtual texturing to provide high resolution views of planetary surfaces, and it also makes use of height maps. Even though Gaia Sky can accommodate this kind of data, by default only the planets, a selection of moons, some minor planets, 14,000 asteroids, and the Oort cloud are provided. Also, Gaia Sky also does implement terrain level-of-detail and use height maps, but it lacks virtual textures as of today.

Open Space uses a scene graph for its internal model, like Gaia Sky. Open Space, however, makes use of the more complex Dynamic Scene Graph [Axe+17] approach to be able to represent a high dynamic range of distances. Gaia Sky, on the other hand, uses the floating camera approach (Section 7.2.2) to avoid floating-point precision issues.

7.1.3 Celestia

Celestia has been around for many years, even though its development was stopped from 2011 to 2017. The current build is version 1.6.2.2 on Windows and macOS, and 1.6.2.1 on Linux. It still uses Hipparcos as the main catalog (120 K stars), and it is mainly focused on the Solar System, including virtual texturing for handling on-demand high-resolution textures of planetary surfaces and SPICE kernels for Solar System objects. Gaia Sky, in contrast, can handle much larger catalogs, but does not implement virtual texturing. Celestia has built over the years a vibrant and active community, producing different data packs. It also offers mobile ports for iOS and Android.

7.1.4 Space Engine

Space Engine is proprietary closed-source software and Windows-only. It borrows heavily from the gaming industry, making use of impressive graphics, particle effects, and procedurally generated environments and worlds. As a star catalog, it uses Hipparcos. It contains only 130 K real objects, counting stars and other object types. Gaia Sky's focus, on the other hand, lays not so much on graphics but on the efficient traversal and representation of the star catalog and other astrophysical data.

7.2 Gaia Sky

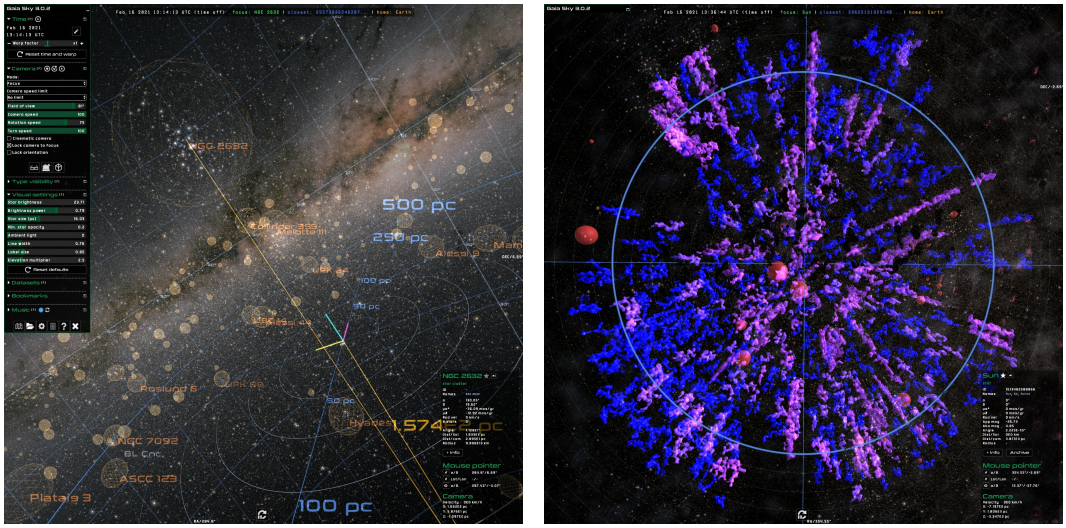
Our framework, named Gaia Sky [Saga] (Figure 7.1), is an open-source endeavor developed since 2014 in the framework of the Data Processing and Analysis Consortium of ESA's Gaia cornerstone astronomy mission. Gaia aims at creating a six-dimensional map of more than one billion stars with positions and velocities. In this context, the main mission of Gaia Sky is to deliver an off-the-shelf visualization of the Gaia catalog, and to aid in the production of outreach material. However, it has a wide range of other applications, from scientific to purely recreational. For instance, it was used within DPAC to help visualize and understand the stray light path, an optical issue with the spacecraft construction arisen in its first months of operation.

Gaia Sky contains a full simulation of our Solar System, with all its planets, dwarf planets, the main moons, and thousands of asteroids and asteroid orbits. In its current version, it allows for the exploration of Gaia eDR3. We offer different datasets based on DR2+ data, selected according to different parallax relative error criteria. They contain from 5 million stars to more than 600 million. Other included data are star clusters, extragalactic sources, and derived structures, like *isodensity surfaces* of stars, dust, and HII regions (see Figure 7.2). Beyond that, Gaia Sky also enables the visualization of Gaia's trajectory from Earth to its site of operation, 1.5×10^6 km behind the Earth away from the Sun, where it follows a Lissajous orbit around the Lagrangian point L_2 (see Figure 7.3), and the octree structure, the Gaia data is stored in (Figure 7.4).

The main strength of Gaia Sky, however, is that it can handle catalogs in the hundreds of millions of objects. The main challenges, that led to novel design solutions in Gaia Sky, include the efficient handling of these data to enable interactive exploration, and managing the very large range of scales with sufficient numerical precision. Gaia Sky also features, besides a large number of exploration tools, a scripting interface, which can be accessed by directly running the scripts, or via a REST API, a variety of stereoscopic modes, a panorama (360°) mode, a planetarium output mode together with multiple projection common data interchange (MPCDI) support for multi-projector systems, and VR support via the OpenVR API. Additionally, a camera recorder utility enables the recording and later playback of the full camera state at a user-defined frame rate. Before we provide a detailed description of the system (Section 7.3), we describe our data representation technique (Section 7.2.1), our approach to ensure sufficient numerical precision (Section 7.2.2), our solution to the depth buffer precision problem (Section 7.2.3) and a representation of relativistic effects (Section 7.2.4). A discussion on the performance of the system, together with a comparison to other similar solutions, is provided in Section 7.4.

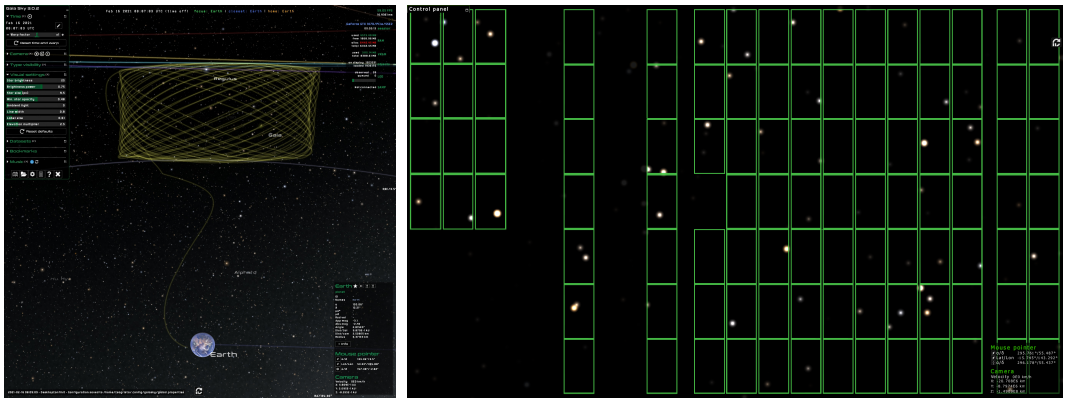
7.2.1 Data Representation and Access

One of the central challenges in the conception and development of Gaia Sky was the effective handling of the large star catalog. Compared to other star catalogs, DR2+ are large in several regards. First, they are large in the number of stars. DR2+ contain well over a billion stars with, however, varying quality of the line-of-sight distance (denoted parallax relative error). Here, we use a subset of 601 million stars for which the parallax relative error is smaller than 90.0%. This is orders of magnitude larger than previous 3D star catalogs. We offer a selection of eDR3-based star catalogs on our data repository. The largest of them contains 1.46 billion stars and uses the geometrical Bayesian distance



(a) (b)

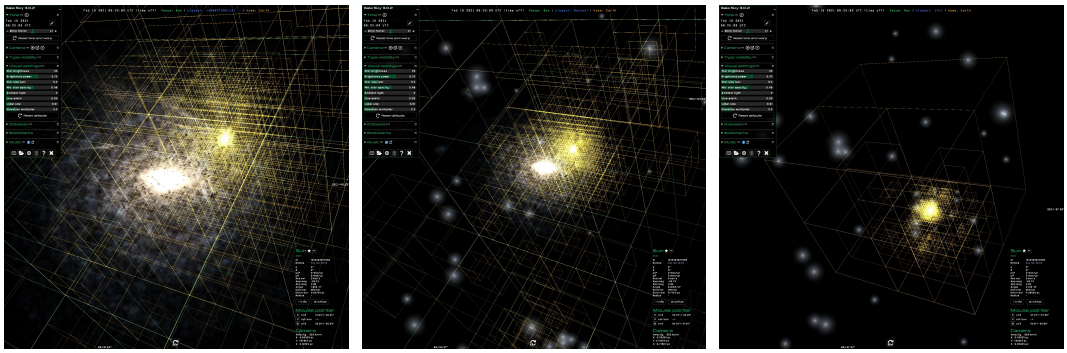
Figure 7.2: (a) Objects from the DR2 Open Clusters (OCDR2) catalog indicated by spherical meshes, with a logarithmic *galactic grid* in blue. (b) Semi-transparent isosurfaces of dust (black), HII regions (red), and hot OB stars (blue-purple) [Jar]. The blue circle sits at the 2.5 kpc mark from the Sun, located at the center of the picture.



(a) (b)

Figure 7.3: (a) Gaia trajectory from Earth to its site of operation, the Lagrangian point L_2, and its Lissajous orbit. The scales are adapted such that both parts can be shown together. In the background we can see the orbit trails of other Solar System objects and some stars of the eDR3 catalog. (b) The CCD array of Gaia in the FOV camera mode of Gaia Sky.

determination method by Bailer-Jones et al. [Bai+21]. Second, DR2+ are large in terms of spatial scales. Our Milky Way galaxy spans about 1×10^{21} m in its diameter, but on the other hand, the dimensions of the Gaia spacecraft are only a few meters. Third, it is large in the range of star brightnesses. In contrast to most previous catalogs, DR2+ in-



(a) (b) (c)

Figure 7.4: Octree structure of Gaia Sky using the eDR3 bayesian distances catalog [Bai+21] with 1.47 billion stars. (a) displays a close-up view of the Milky Way and the inner octree structure, while (b) and (c) show successively farther scenes that contain the full octree.

clude astrometric data, spanning roughly 18 orders of magnitude in brightness [Gai+18]. These different high dynamic ranges are intertwined, and an effective exploration and navigation system needs to take this information into account and exploit it. A tailored level-of-detail (LOD) structure with streaming capabilities is therefore comprising a central part of Gaia Sky's core system. In many applications, LOD is accomplished by an octree (Section 6.2.2), whose higher levels contain coarser approximations of the scene components, usually called impostors, while lower levels contain more detailed versions. During navigation, these octrees are traversed, and distance-based culling is employed to decide on the visibility of each of the octants of the octree. The idea underlying these approaches is to omit detail that the observer cannot perceive, i.e., to omit detail that would appear too small, because it is too far away compared to its size.

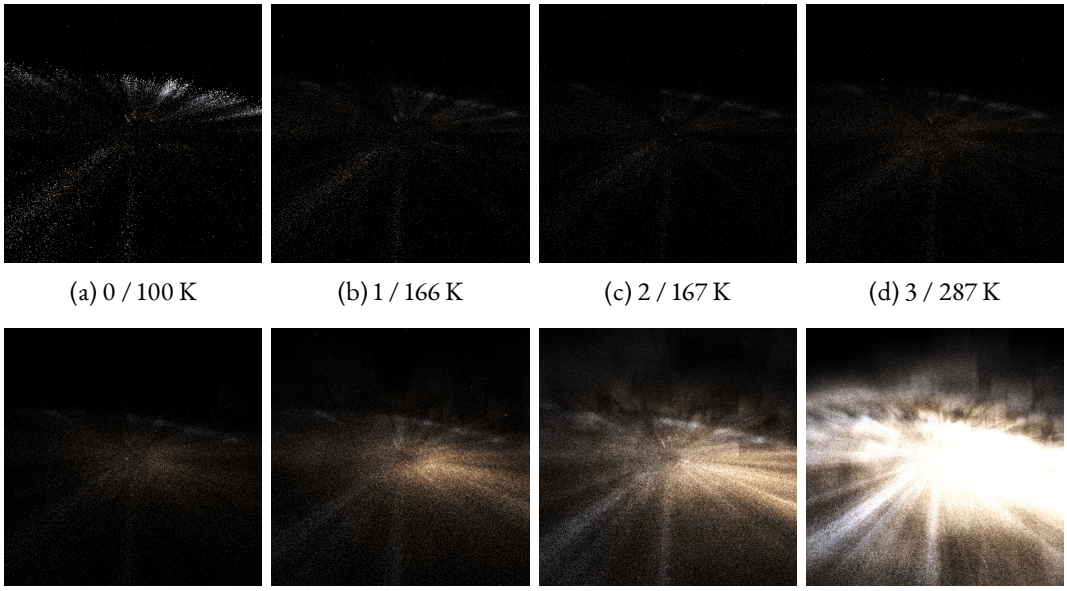
In Gaia Sky, however, we follow a different approach, because there are substantial differences between typical scenes and Gaia data. First, stars are, by nature, discrete. That is, although they are huge objects, they are clearly confined and, given the incredible distances between them, they have to be considered points on galactic scales. Therefore, using smoothed and coarsened representations as impostors for sets of stars would not be an optimal choice, as the impostors would need to remain consistent from all viewing directions. Additionally, stars in Gaia Sky can move during the simulation due to their proper motions, and their shading parameters can be manipulated in real-time, making it even harder to use impostors. At the same time, the visibility of stars is not determined by their distance alone—it is the combination of their absolute brightness (which is defined for a fixed distance of 10 parsec) together with their distance to the observer (and possibly extinction) that is responsible if a star can be perceived or not. These considera-

tions motivate our main contribution regarding star catalog representation in Gaia Sky: magnitude–space level-of-detail (MS-LOD).

We exploit the correlation between distance, absolute star brightness, and apparent star brightness by constructing an octree that contains the stars sorted by absolute magnitude in a descending manner. That is, the root level contains the brightest stars, and lower levels progressively contain fainter and fainter stars. Additionally, the mapping of magnitudes to octree levels is bijective, so that each level contains a well-defined range of absolute magnitudes, and a given absolute magnitude can only be assigned to one level. Figure 7.5 shows an example for the stars in the first eight levels of the octree, from depth zero, the root (a), to level seven (h). Due to the non-uniform distribution of stars and the limited number of stars in each octant, some irregularities can be seen because the stars of a level are inspected in an isolated manner here. From (a) to (d), the number of stars in each level is nearly the same, but the brightness of the stars decreases, as does the overall brightness of the resulting image. From (f) to (h), the brightness of the stars still decreases, but the number of octants as well as the number of stars increases. That is in particular the case because there are many more faint stars in the Universe than bright ones. Hence, because of the additive blending, the overall image brightness strongly increases as we move to deeper levels, from (a) to (h).

Octree construction. For the construction of the MS-LOD octree structure, the whole catalog needs to be loaded into memory, where the stars are sorted by absolute magnitude from brightest to faintest. The starting point is the Cartesian axis-aligned bounding box that contains all the stars, and that serves as the root of the octree. The recursive subdivision of the octree is controlled by the parameter N_σ which is the maximum number of stars in an octant.

In practice, we start with level $n = 0$ and fill the root node with the N_σ brightest stars. If there are more stars than N_σ, we subdivide the root into its 8 octants. Then, in the next level, we take stars from the top of the (brightest) list and assign them to the corresponding level octant, determined by the position of the star, until an octant reaches N_σ stars. Next, we proceed one level deeper and start over. This procedure is carried out recursively until no star is left. Thus, a small N_σ leads to larger octrees with more octants, whereas a larger value leads to smaller, more compact octrees which are faster to process but have more stars in each node. Nevertheless, this has only an effect on performance and not on visual appearance. Note that this octree construction method provides a mapping between absolute magnitude and depth level, which adapts automatically to the absolute magnitude distribution of the input catalog. This partitioning in magnitude space be-



(a) 0 / 100 K (b) 1 / 166 K (c) 2 / 167 K (d) 3 / 287 K

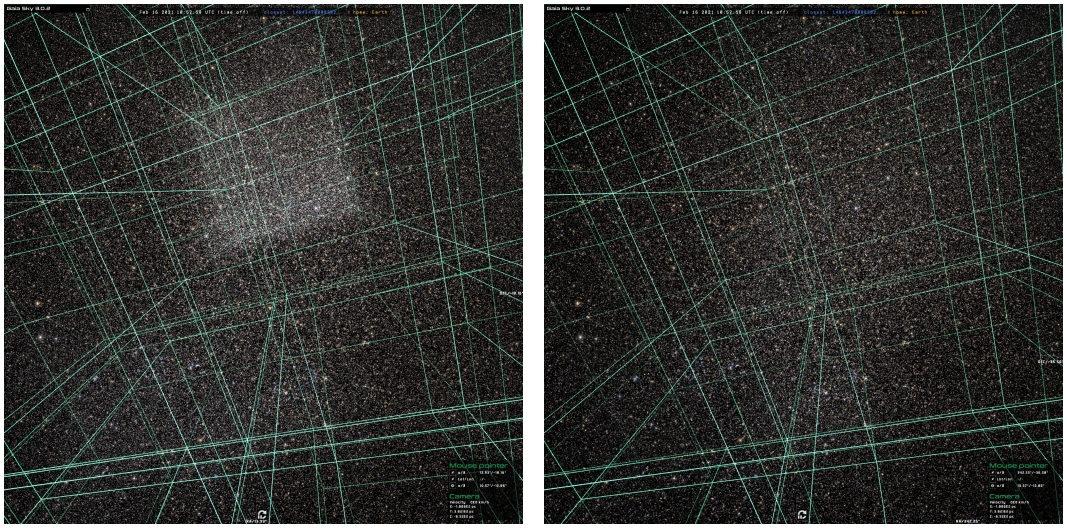
(e) 4 / 521 K (f) 5 / 1.54 M (g) 6 / 4.48 M (h) 7 / 14.4 M

Figure 7.5: View of all the stars in the first eight levels of the octree of the catalog with a parallax relative error up to 90% (601 million stars). Captions provide octree depth / number of stars in respective level. © 2019 IEEE.

tween depths guarantees that brighter stars will always be visible from farther away than fainter stars.

Octree traversal. For rendering, the octree is traversed and the visibility of each octant is determined using a user-defined view angle Θ, which is compared to the solid angle θ_i of each octant i with respect to the position of the camera. Hence, Θ represents a draw distance measure by defining a threshold below which the octants are not visible, and above which they become visible. Octant i is visible if $\theta_i \geq \Theta$ and it intersects the camera's view frustum. Figure 7.7 explores different scenarios with varying camera distances and octant sizes.

When an octant is visible, its stars are sent to the rendering system, and its children octants, if any, are processed the same way. If an octant is not visible, its stars are not rendered and its children octants are not processed. Without any further modifications, however, this would yield pop-ins of octants, as shown in Figure 7.6. Therefore, we employ a LOD smoothing mechanism to fade-in octants (and its stars) into view. This is accomplished by a linear mapping of the solid angle θ_i to $[0, 1]$, with $\theta_i = \Theta$ mapping to zero, and $\theta_i = \Theta + r$ mapping to one, where r is an offset to be used as a masking value for the star shader. Usually, the star shading (Section 7.3.6) automatically renders distant,



(a) (b)

Figure 7.6: (a) Direct octree traversal based on the view angle Θ alone may yield pop-ins of octants. (b) LOD smoothing to fade-in the octants and eliminate pop-ins.

faint stars invisible, but this entirely depends on Θ. Thus, large values for Θ make pop-ins more probable and the smoothing more useful.

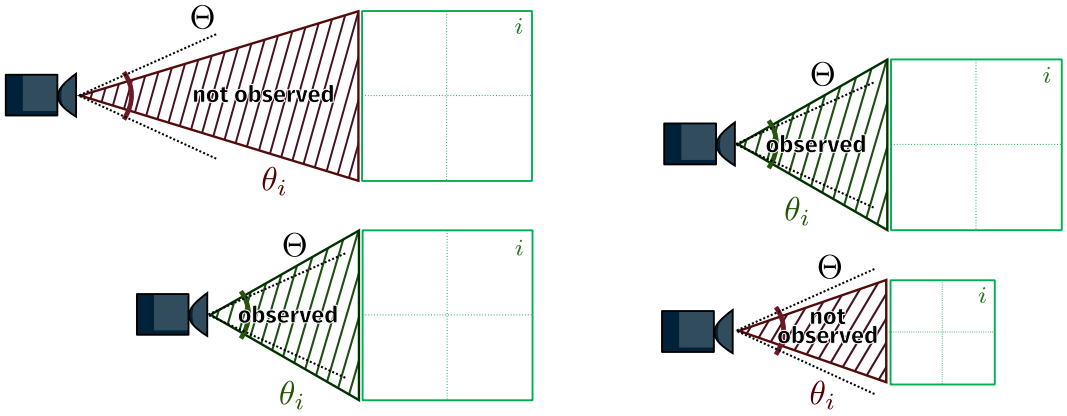
7.2.2 Floating Camera

Gaia Sky contains a scene which represents vast distance ranges, from centimeters to gigaparsecs, far beyond what can be represented using 32-bit floating-point arithmetic (see Section 6.3). To avoid such issues, we follow a new approach which is somewhat complementary to previous ones like, e.g., the dynamic scene graph proposed by Axelsson et al. [Axe+17] or the logarithmic landmarks introduced by Li et al. [LFH06].

The idea is to keep the camera at the origin of the global coordinate system at all times, and offsetting the whole scene graph so that the current camera is always located at position $(0\ 0\ 0)$, as shown in 7.8b. In practice, we add a new node at the top of the scene graph (Figure 7.15) with a translation by the inverse of the camera position vector, which is updated every frame. In detail, we split the standard vertex transformation pipeline from object space to the camera's clip space into two parts:

$$\mathbf{p}_{\text{clip}} = \mathbf{M}_{\text{proj}} \cdot \mathbf{M}_{\text{view}} \cdot \mathbf{M}_{\text{model}} \cdot \mathbf{p}_{\text{obj}} = \mathbf{M}_{\text{proj}} \cdot \mathbf{M}_{\text{mv}} \cdot \mathbf{p}_{\text{obj}}. \tag{7.1}$$

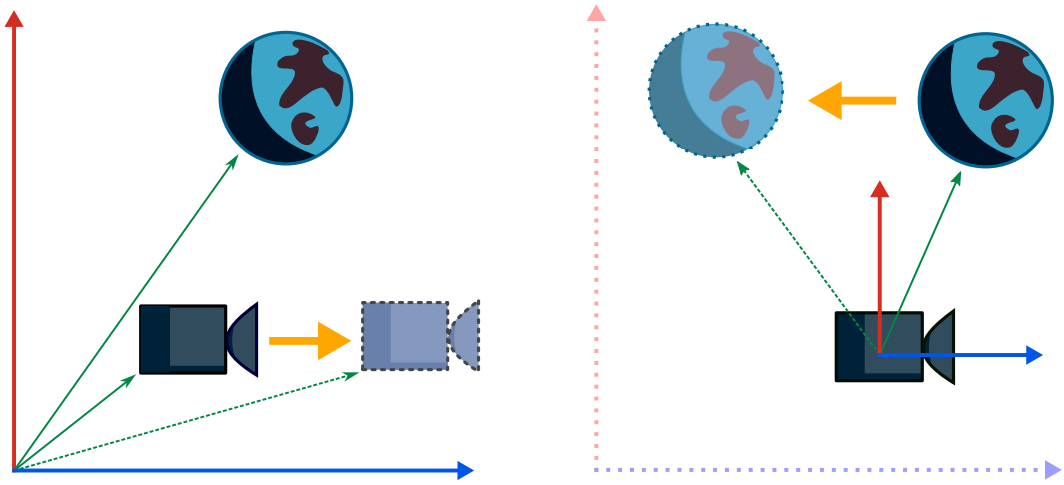
The crucial step is the transformation defined by the model-view matrix $\mathbf{M}_{\text{mv}}$ between object coordinates, $\mathbf{p}_{\text{obj}}$, and camera coordinates. For that, we use the high-performance



(a) (b)

Figure 7.7: Two different octree traversal scenarios. In (a), we have the camera at different distances from the same octant. At the top, the camera is far enough so that $\theta_i < \Theta$ holds and the octant is thus not observed. At the bottom, the camera is closer to the octant, allowing θ_i to become greater than Θ, making the octant observed. In (b) we have the camera at the same distance from two differently-sized octants. At the top, the octant's solid angle θ_i is large enoguh to be observed. At the bottom the octant is much smaller (i.e. belongs to a deeper level in the octree and contains fainter stars), to the point where $\theta_i < \Theta$, rendering it not observed.

arbitrary-precision floating-point operations from the `Apfloat` library on the CPU to handle every transformation within the scene graph. The resulting model-view matrix is then uploaded to the GPU, where we can still use single float precision shaders to convert from local camera coordinates to clip coordinates $\mathbf{p}_{\text{clip}}$, and finally determine screen coordinates. Figure 7.9 shows the Gaia telescope model which is located in the Lagrangian point L_2, about 1.5 million kilometers behind the Earth away from the Sun, while the camera is only 275.04 m away from the telescope. Hence, the model matrix $\mathbf{M}_{\text{model}}$ consists of a translation vector for Gaia with a length of 1.5×10^{11} m (distance Sun–Earth) plus 1.5×10^{9} m, while the view matrix $\mathbf{M}_{\text{view}}$ consists of a translation vector of a similar length. Using global coordinates and standard 32-bit floating-point shaders, this yields *jittering* of the vertices due to the lack of floating point precision in a phenomenon known as catastrophic cancellation (Section 6.3.2), and finally leads to incorrect per-pixel lighting (Figure 7.9a). Note that in Figure 7.9a, we even had to artificially reduce the distance to Gaia down to 1200 km because rendering broke down completely for larger distances. With our method (Figure 7.9b) based on the arbitrary-precision model-view matrix calculation on the CPU, there is no jittering problem because distances of a few centimeters compared to the relative distance between camera and the telescope can be handled by single float precision.



(a) (b)

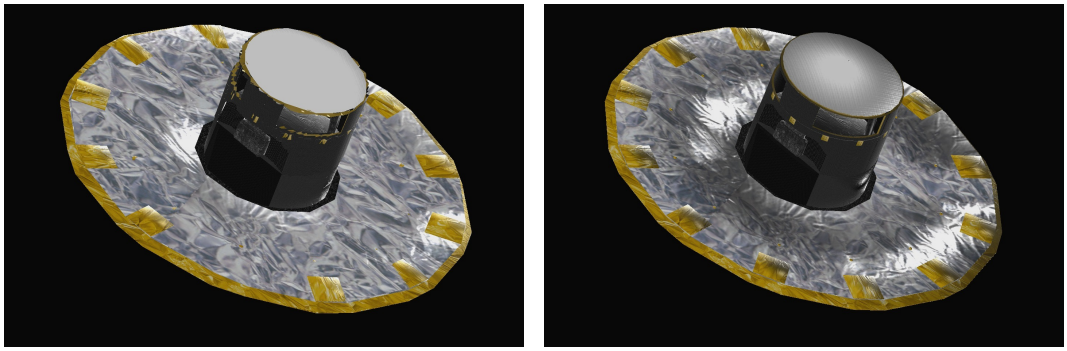
Figure 7.8: Classic camera setup (a), where a camera forward movement implies an update of the translation part of the camera view matrix (yellow vector) in the global world coordinate system (in red and blue). In contrast, our method (b) floats the reference system to the camera position and applies the inverse translation to all the objects in the scene. Note that, from the camera's point-of-view, the relative movement experienced by the planet is exactly the same.

The *floating camera* approach provides a simple and effective way to handle a high dynamic range of distances without precision issues. It achieves this by just adding a translation to the transformation matrices. In comparison, the Dynamic Scene Graph requires the camera to be moved around in the scene graph, and its position to be taken into account when computing the transformations applied to all nodes.

7.2.3 Logarithmic depth buffer

In the previous section we have addressed the numeric representation in the context of floating-point arithmetic of a large scene with vastly different distance ranges. This problem is also present when we calculate the z-value that goes into the depth buffer. The depth buffer, as we have seen in Section 6.1.6, contains the depth value of the fragment nearest to the camera at each pixel position. It avoids the need for sorting the geometry before the rendering stage, speeding up the whole process.

In most graphics hardware, the depth buffer has a precision of 16 to 24 bits. While this is sufficient for small, contained scenes, in larger worlds where occlusion elements can be seen at large distances this might not be enough. Gaia Sky represents very small objects of a few hundreds of centimeters, such as satellites or spacecraft, together in the same scene with very large occluding structures spanning over vast distances of parsecs and kilopar-



(a) (b)

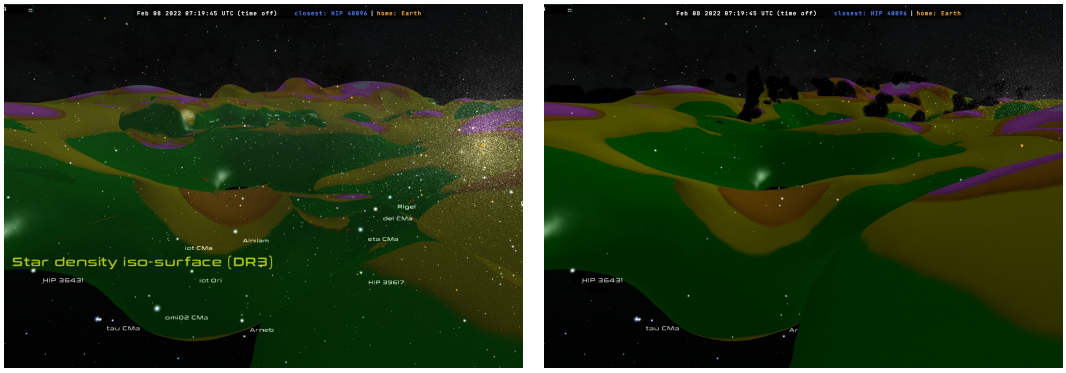
Figure 7.9: (a) Straightforward vertex transformation using 32-bit floating point precision leads to jitter. The resulting misaligned structures of the model lead to wrong per-pixel lighting. (b) Our floating camera approach does not exhibit these issues. © 2019 IEEE.

secs, like star density *isosurface meshes* or dust clouds. The classical depth buffer function shown in Equation 6.4 is a non-linear function proportional to $1/z$ overwhelmingly favoring precision close to the camera by sacrificing it at far away distances. For instance, in the classical function, the far half of the frustum volume in z is mapped to the last 2% of our available z values in the $[0,1]$ between the near and far planes of the frustum. Unfortunately, the classical configuration means that most of the depth buffer's precision is essentially wasted. For our purposes, this method gives excessive precision for object close to the near plane, and almost none for objects further away. In Gaia Sky, this results in large and distant objects to be assigned totally incorrect z-values. As seen in Figure 7.10, images are rendered incorrectly, where far away polygons are drawn over closer ones.

Common methods to address this issue involve moving the near plane further away or partitioning of the z space into two or more segments to address near and far geometry separately. Both these methods bring on their own sets of problems. Moving the near plane away helps with improving the precision at large distances, but closer objects are no longer visible, as they fall outside of the viewing frustum. On the other hand, partitioning the z space into different regions adds complexity and can get hard to manage, especially as the number of partitions increases.

Our approach uses a different depth function altogether. This function moves some of the wasted precision from nearby areas to distant ones. We choose as the basis of our z-value distribution a logarithmic function that depends on the fragment's z value, the far plane distance z_f and a constant C that determines the resolution near the camera:

$$f(z,C,z_f) = \frac{\log(Cz+1)}{\log(Cz_f+1)}. \tag{7.2}$$



(a) (b)

Figure 7.10: (a) A scene with star density iso-surface meshes and dust cloud structures is rendered incorrectly using the classical depth buffer function. The mesh label, in yellow, is several kiloparsecs behind the green surface that is rendered behind it. While the dust clouds are not visible when they should be, most of the bright stars are visible but they should be occluded. (b) Our logarithmic depth buffer approach addresses this issue by employing part of the precision wasted on nearby objects to improve depth determination over the whole distance range. The dust clouds are correctly rendered and visible in the distance.

The resolution of this function at a distance d, for a given C and n bits of z-buffer can be computed as

$$r(d, C, n, z_f) = \frac{\log(Cz_f + 1)}{2^n C/(Cd + 1)}, \tag{7.3}$$

which, along with a far better utilization of the available depth buffer precision almost gets us rid of the near clipping plane. In our tests we have seen that a C value of 1×10^7 gives us good results, but this is heavily linked to the scaling of internal distance units, so it varies from application to application.

Figure 7.11 contains a visualization of various depth buffer functions, including the classical and the logarithmic ones.

7.2.4 Relativistic Effects

Gaia Sky implements a couple of relativistic effects, namely relativistic aberration and the apparent visual distortion due to gravitational waves. Both effects are implemented in an integrated way, i.e. acting on the whole of the geometry in the scene, and are turned off at code level using the shader pre-processor, which minimizes the general overhead to zero when the effects are turned off.

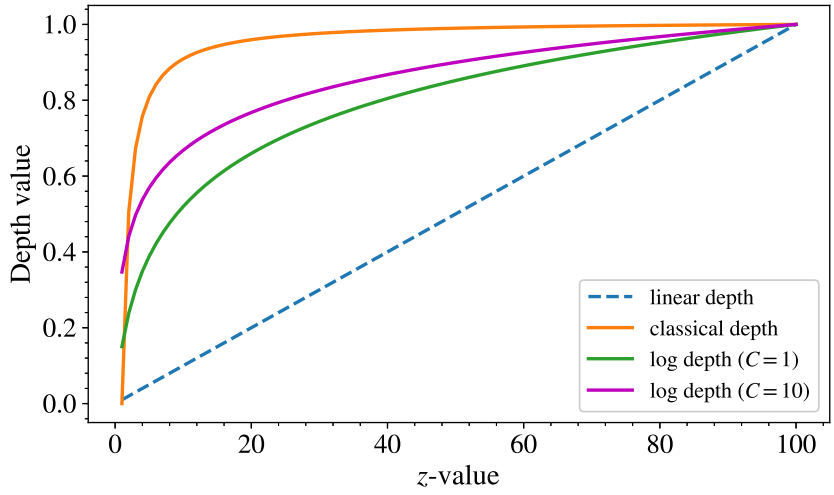


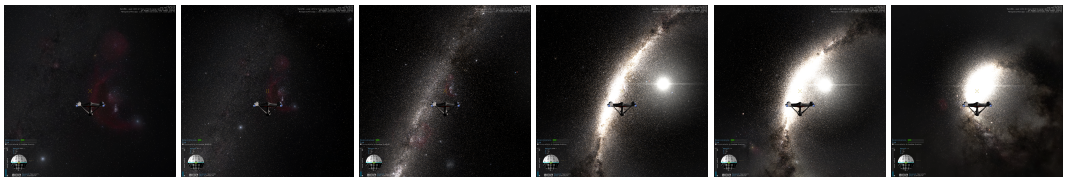
Figure 7.11: A comparison between the linear depth buffer (blue, dashed), the classical non-linear function (orange) and our logarithmic depth buffer function (green with $C = 1$, magenta with $C = 10$) for world z range in $[1, 100]$, mapped in the x-axis. Observe how the classical function is heavily biased towards lower z-values, while neglecting higher ones. In contrast, our approach provides a better depth distribution by using a logarithmic function.

Relativistic Aberration. The relativistic aberration of light is a special-relativity effect that arises when an observer moves at a velocity v close to the speed of light c. In that case, the rays of light from each source that reach the observer are tilted towards the observer's direction of motion, which yields an apparent reduction of the effective angle ϑ between the velocity direction and the light source to the apparent angle ϑ' according to the aberration formula

$$\cos\vartheta' = \frac{\cos\vartheta + \beta}{1 + \beta\cos\vartheta}, \quad \beta = \frac{v}{c}. \tag{7.4}$$

This has a strong apparent minification effect in the direction of motion and a strong magnification effect in the opposite direction. As a result, it is even possible that objects, which are actually behind the observer, appear to be in front of her.

Figure 7.12 shows Gaia Sky in relativistic spacecraft mode. While the spacecraft keeps its position relative to the camera, the velocity is increased from zero to nearly the absolute speed limit c. The relativistic aberration is non-linear, which means that even for high velocities, the effect is rather small, and only for velocities near the speed of light, the tilting toward the direction of motion increases dramatically. In that case, even objects which are actually behind the spacecraft, like the Sun in Figure 7.12d, appear in front of the observer. The increasing brightness of the stars is due to the additive blending of the



(a) 0 c (b) 0.5 c (c) 0.9 c (d) 0.99 c (e) 0.999 c (f) 0.9999 c

Figure 7.12: Gaia Sky in relativistic spacecraft mode, with increasing velocity. Even with half the speed of light, $v = 0.5\,\mathrm{c}$, the relativistic aberration is rather weak. The closer the spacecraft approaches the absolute speed limit c, the stronger the aberration becomes. © 2019 IEEE.

star shading. Although this does not show the correct behavior, it is quite similar to the searchlight effect, which causes an increasing brightness in the direction of motion and a strong decreasing brightness in the opposite direction. The searchlight effect, as well as the relativistic Doppler shift, are, however, not implemented yet. A detailed discussion of the relativistic effects is out of the scope of this work, and we refer the reader to, e.g., Müller et al. [MKA08].

Gravitational Waves. Gravitational waves are a general relativistic effect, which produces a disturbance in the curvature of spacetime, generated by accelerated masses. These waves propagate outward from their source at the speed of light. Gaia Sky implements the visual effects that an observer would perceive in such an event, based on the model by Klioner [Kli18]. There are a few parameters (4 amplitudes/polarization and the wave frequency) which can be adjusted according to the source. The current model used in Gaia Sky has a few caveats. First, it is only valid for slowly moving observers, with velocities much smaller than the speed of light. Second, the usual amplitude (strain) of the gravitational wave is very small, but can be artificially exaggerated for visualization purposes. And third, such sources of gravitational waves of the model—supermassive binary black holes in the centers of galaxies—are typically short-living (few thousands of years), so adjusting the parameters according to the simulation time might be in order. Figure 7.13 shows the influence of the apparent distortion due to a gravitational wave on the view of Mars and its moon Phobos.

7.3 System

This section provides a bird's eye view of the system parts, which are not novel, but nevertheless essential to the system as a whole. Gaia Sky is implemented using Java and OpenGL. The OpenGL bindings are provided by the Lightweight Java Game Library [LWJ], and an additional framework layer, libGDX [Lib], is used for its caching,

(a) (b) (c)

Figure 7.13: Apparent distortion due to a gravitational wave on the view of Mars and its largest moon Phobos. © 2019 IEEE.

batching, and loading utilities. The choice of Java as a main platform is due to the abstraction provided by the JVM, which makes it easy to port to and maintain several platforms, one of the main requirements of the project. Furthermore, the language of choice of DPAC for the data processing of the Gaia data is Java, and there exists a vast code base with interdependent libraries and utilities, to compute, for example, the attitude quaternions of the satellite, which would need to be all translated to another language otherwise. Additionally, the default use of the just-in-time compiler, which gives the compiler access to runtime information not available to native languages, plus the avoidance of garbage collection as much as possible by using object pools and custom collections, helps minimizing the impact of using such a platform for a real-time graphics application where high frame rates are needed. Performance hot-spot functions, such as matrix operations, are implemented in plain C and called using the Java Native Interface (JNI).

7.3.1 Main Loop

At its highest level, the system implements a very simple update–render loop, in charge of updating the model objects and rendering them each frame. The loop also computes the amount of time elapsed since the last iteration, and passes it on to the update and render stages. An effort is currently ongoing to migrate the object-oriented structure into an entity component system (ECS) and to parallelize some of the update processes.

7.3.2 Structure

Gaia Sky is structured into modules, each with a defined set of responsibilities. Only five main modules compose the backbone of Gaia Sky, as seen in Figure 7.14: the user

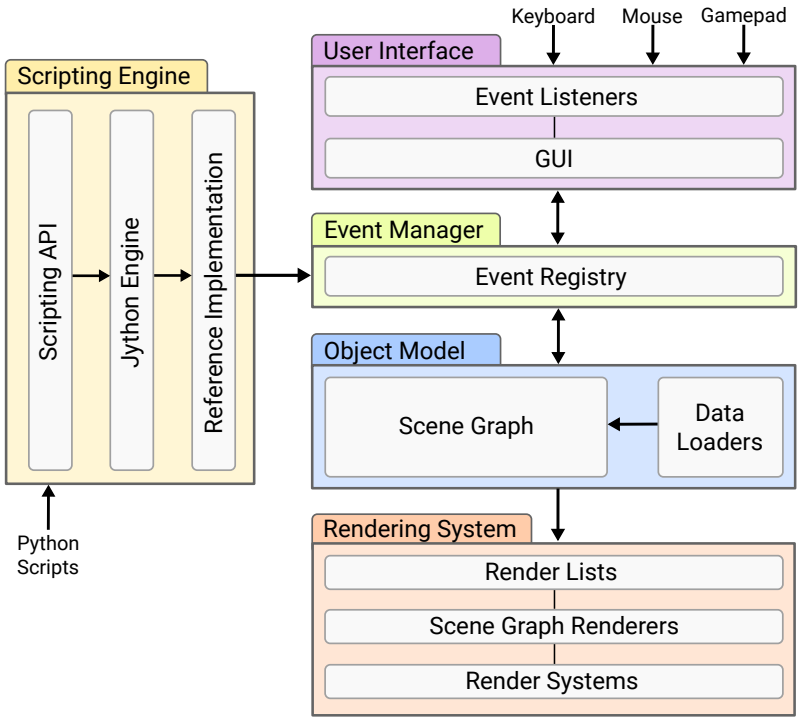


Figure 7.14: Gaia Sky component structure. The interactions captured by the UI are processed and the relevant events are posted to the event manager, which broadcasts them to the relevant parties. The object model is updated continuously, and its entities are added to the render lists, which are used by the rendering system to display the scene. © 2019 IEEE.

interface module, the event manager, the object model, the rendering system, and the scripting engine. Below, we give a brief overview of the individual modules, and provide detailed descriptions in the subsequent sections.

User interface. This module is in charge of managing all interactions with the user. Among its responsibilities are generating and rendering the graphical user interfaces, as well as listening and processing the user's input events through the input listeners. The user interface is skinnable and supports HiDPI themes. The communication of the user interface module with other modules is accomplished by means of the event manager.

Event manager. This component is a generic registry, where event producers and consumers are connected via a centralized hub. Any entity can publish and subscribe to events of interest. The event manager reacts to events synchronously and sequentially. Actions can be specified to be processed after the current loop cycle has finished, ensuring thread safety and consistent state of the object model.

Object model. The object model contains and manages all the scene objects, and is in charge of updating their state during every iteration of the main loop.

Rendering system. This module is in charge of rendering the scene. Objects are routed from the object model to the rendering system using a number of render lists. These lists are used by the renderer objects, producing different kinds of visuals for each element. Initially, the objects to be rendered are contained in a series of render lists, which are previously filled up in the update stage, where the objects themselves decide how they are to be represented visually. These render lists are used by the scene graph renderer objects, which produce different kinds of render outputs from the same scene graph.

Scripting engine. Finally, the scripting engine is a component that allows for the execution of Python scripts using an API to access and modify the internal state of Gaia Sky with ease and precision.

7.3.3 User Interface

The user interface contains the graphical user interface objects and the input event listeners. It controls and directs all the information flow between the user and the relevant components of Gaia Sky.

There are several GUI objects in Gaia Sky which can be registered and unregistered at will depending on certain conditions. Some GUI objects' life span is short-lived, like the loading GUI, which is only ever used when Gaia Sky starts up, and initially loads data and allocates resources, while others live as long as the program itself. When the screen mode changes from normal to stereoscopic, the regular GUI is unregistered and the stereo GUI is registered. The user interface is skinnable (four different skins are provided), can be arbitrarily scaled to accommodate HiDPI displays, and is internationalized. This internationalization system pulls text elements from a series of files corresponding to different languages. So far, it contains, additionally to the base English file, translations to Spanish, German, French, Catalan, Slovenian, Russian and Bulgarian.

The input event listeners define and implement the input interactions available via input interfaces, like keyboards, mice, game controllers, or VR controllers. The control scheme may change from mode to mode, and so do the input event listeners, which are registered and unregistered when needed, in the same fashion as the GUI objects.

7.3.4 Event Manager

The event manager is a general hub which routes and directs most of the information flowing through Gaia Sky. It contains a registry where listener objects can register to certain events, which, when posted, trigger notifications on the listeners. All listeners need to follow a contract similar to the observer pattern, in order to be notified when an event, they have registered to, is posted. Once an event is posted to the event manager, the notifications to the listeners are sent immediately in a synchronous and sequential way. It is the responsibility of the listeners to parse the event data, and handle any errors that may come up. Events can be posted from any thread, and the subsequent actions, implemented by the listeners, are run immediately in the same thread. However, these actions may make use of a post-runnable job queue, which allows for posting jobs that will be run right after the render stage in the main thread, ensuring thread safety and a consistent state of the object model.

7.3.5 Object Model

The object model holds all the objects in the scene. These objects are organized into a scene graph (Figure 7.15), a tree-like data structure in which geometrical transformations are inherited from parent to children. All objects in the object model are categorized into component types, which organize the objects according to their typology. Available component types are stars, planets, moons, satellites, asteroids, clusters, labels, grids, orbits, atmospheres [ONe05], constellations, constellation boundaries, galaxies, topological information, locations, arbitrary meshes, titles, and others. These component types are mainly used to implement the switch-on/off behavior, to prevent all data being displayed at once, and thus cluttering the viewport.

Gaia Sky uses a global equatorial Cartesian reference system with the origin at the Sun. All objects need to be translated to that reference system during the loading phase. As catalogs are usually given in heliocentric coordinates, and so is Gaia's, a translation is not necessary. The Milky Way node represents the Milky Way object, and needs a translation of about 8 kpc from the position of the Sun. Other datasets, such as the Nearby Galaxies (NBG) or the Sloan Digital Sky Survey (SDSS, distant galaxies and quasars), are also children of the Universe node. Finally, the "Sun" node represents both, the star and the barycenter of the Solar System. All objects in the Solar System are descendants of the Sun in the scene graph.

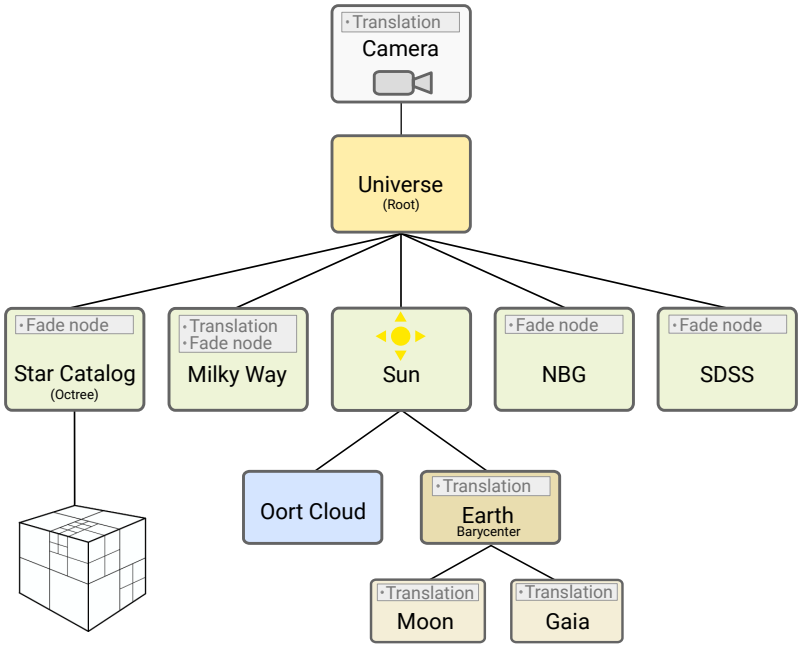


Figure 7.15: Part of the scene graph used in Gaia Sky, with floating camera at the root. The global reference system is centered around the "Sun". © 2019 IEEE.

Data Loaders

Gaia Sky handles many different kinds of data, and their loading mechanisms can vary substantially. In a broad sense, the types can be organized into two large groups: *particle data* and *non-particle data*.

Particle data. Stars, galaxies, and essentially all data which are point-based, and come from astronomical catalogs, are particle data. They usually have a single node representing them in the scene graph. The different catalog formats are supported through the STIL Java library [Tay05], including VOTable [Och+13], FITS, ASCII, CSV, and GBIN. The STIL loader relies on Unified Content Descriptors (UCD) [Der+04] defined by the International Virtual Observatory Association (IVOA) to assign units and semantics to each data column. Gaia Sky also uses an own binary format that is very compact and can be memory-mapped very easily for increased performance. As an example, Figure 7.16 shows the catalog descriptor file for one of the eDR3 particle datasets.

Non-particle data. Any non point-based data, such as planets, orbits, constellations, locations, satellites, grids, and others, have their own object in the data model, and their own node in the scene graph. These data are usually loaded from JSON files, using a

```
"name": "eDR3 - default",
"version": 2,
"type": "catalog-lod",
"mingsversion": 30002,
"description": "Gaia eDR3 default: 20%/1.5% bright/faint parallax relative error.",
"link": "http://gaia.ari.uni-heidelberg.de/gaiasky/files/autodownload/",
"size": 1199686648,
"nobjects": 14950954,
"data": [
{
   "loader": "gaiasky.data.group.OctreeGroupLoader",
   "files": [ "data/catalog/edr3-default/particles/", "data/catalog/edr3-default/metadata.bin" ],
   "epoch": 2016.0
}]
```

Figure 7.16: Example descriptor of the default eDR3 catalog. It contains metadata, and the octree loader, which provides the level-of-detail metadata file, containing the actual tree structure and all its nodes, and the folder where the actual star data files are.

specific set of loaders that match attributes by name via reflection. Each of the files defines a few one-to-many relationships which specify what files are to be loaded by which data loaders. The default loader (JSONLoader) is in charge of fetching the data in several JSON files, such as `meshes.json`, `planets.json`, or `locations.json`. Figure 7.17 shows, as an example, a snippet of `planets.json`, which defines the planet Earth. The definition contains some physical properties, such as the size, the apparent magnitude, the atmospheric scattering wavelength values, or the rotation elements. It also contains the name of the orbit object, the coordinates provider (in this case, we use VSOP87 [Bre82], a semi-analytic model for high-precision calculation of planetary ephemerides), and various rendering properties, such as the type of model or the base, specular, normal, metallic, roughness and emissive texture file paths.

Levels of Detail

Once the octree is constructed, Gaia Sky loads its structural metadata and traverses it at each frame, selecting the "visible" nodes and sending them to the relevant renderer.

If the catalog is large and does not fit into memory, as is the case of most DR2+ catalogs, Gaia Sky implements a streaming loader solution, depicted in Figure 7.18. This loader relies on the octree data pages (octants) to be distributed in different files. The loader defines two queues, a load queue, which is a priority queue weighted by the inverse of the depth of the octant in the tree—lower-depth octants containing brighter stars are loaded first—and an unload queue, which contains all loaded octants sorted by access date—octants that have not been visited for the longest time are in the head. In this case, each time an octant is visited, the culling algorithm is run to determine its visibility. If it is visible, we query its state (*not_loaded*, *loaded*, *queued*, or *loading*). If the octant is not yet loaded, it is added to the load priority queue. If it is loaded, its state in the unload

```
    "name": "Earth",
    "wikiname": "Earth",
    "color": [.13, .26, .89, 1.0],
    "size": 6371.1,
    "ct": "Planets",
    "absmag": -2.78,
    "parent": "Sun",
    "impl": "gaiasky.scenegraph.Planet",
    "refplane": "ecliptic",

    "coordinates": { "impl": "EarthVSOP87",
                     "orbitname": "Earth orbit" },

    "rotation": { "period": 23.93447117,
                  "axialtilt": -23.4392911,
                  "inclination": 0.0,
                  "meridianangle": 180.0 },

    "model": {
       "type": "sphere",
       "params": {
          "quality": 180,
          "diameter": 1.0 },
       "texture": {
          "diffuse": "data/tex/base/earth-day*.jpg",
          "specular":
                "data/tex/base/earth-specular*.jpg",
          "normal": "data/tex/base/earth-normal*.jpg",
          "emissive": "data/tex/base/earth-night*.jpg" }
    },

    "cloud": {
       "size": 6395.0,
       "cloud": "data/tex/base/earth-cloud*.jpg",
       "params": {
          "quality": 200,
          "diameter": 2.0 }
    },

    "atmosphere": {
       "size": 6450.0,
       "wavelengths": [0.650, 0.570, 0.475],
       "m_Kr": 0.0025,
       "m_Km": 0.001,
       "params": {
          "quality": 180,
          "diameter": 2.0 }
    }
```

Figure 7.17: JSON definition of the Earth object within the `planets.json` file. Images with an asterisk are given in different resolutions according to the quality defined in the configuration settings. All units are described in the documentation [Sagb].

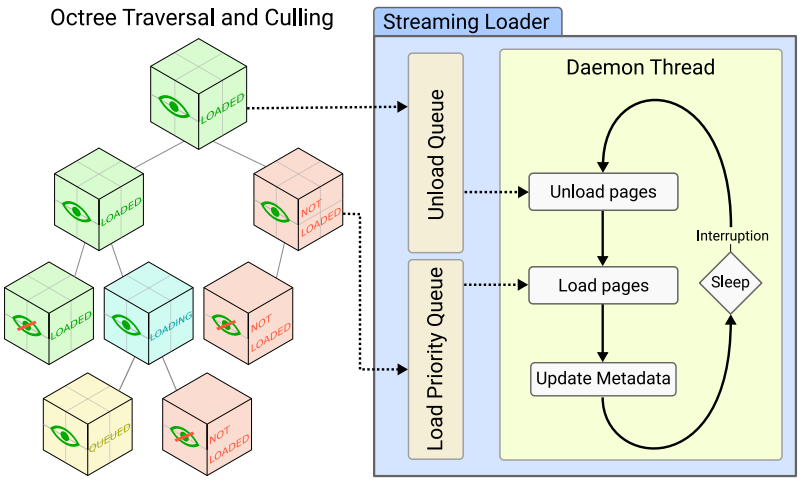


Figure 7.18: Streaming catalog loader. Adding objects to the load queue causes the system to send an interruption to the daemon thread, which then unloads the objects in the unload queue and loads the objects in the load queue. Finally, metadata like constellations are updated. © 2019 IEEE.

queue is updated (it is removed and added to the head). Depending on I/O bandwidth and latency of the file system, the streaming loader may cause pop-ins if the loading of a data page is slower than the camera. The actual loading and unloading of data is handled by a background daemon thread.

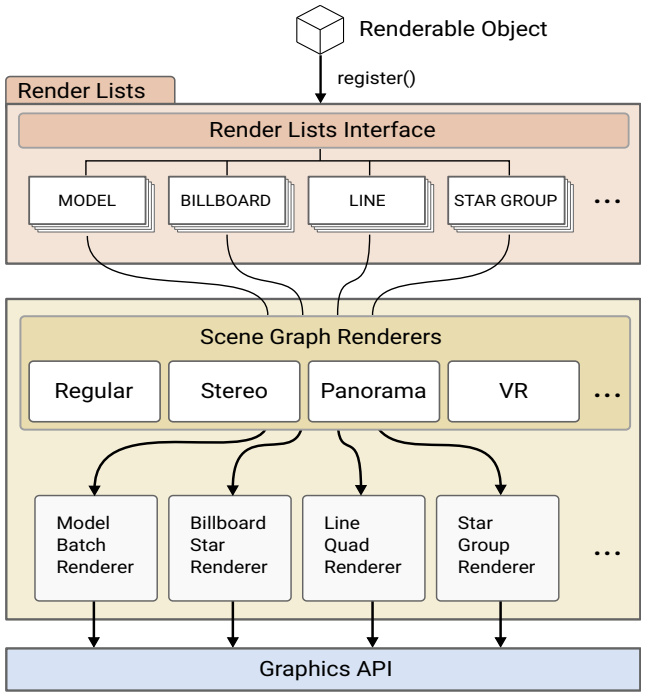


Figure 7.19: Detailed view of the rendering system. The incoming render lists are routed through the scene graph renderers and to the actual renderer objects, which communicate directly with the graphics API. The render lists are filled in the update stage, where objects compute their visibility, and decide how they are to be rendered. After initiating the render stage, the pre-processing stage is run, followed by the scene rendering. Finally, the post-processing stage yields the output image, which runs the pre-passes for different effects and store them in FBOs. After that, the scene is rendered with the currently active SGR. The actual rendering of the scene is done with a set of render objects called render systems. Each render system gets a render target list, prepares the rendering environment and renders the object. Finally, the post-processing stage is run and the result is copied to the screen buffer. © 2019 IEEE.

7.3.6 Rendering System

The rendering system, depicted in Figure 7.19, is in charge of rendering the scene to the active render targets. Possible render targets are the display, image files, or the VR API. The main rendering pipeline consists of four main stages executed in a sequential manner: the pre-passes, the scene rendering, and the post-processing stage. They are described in the following sections.

Pre-Pass Stage

The first stage is the pre-pass stage. In this stage, the scene is rendered to FBOs using different techniques, to be used in later stages.

Shadow map pass. The shadow map pass gathers the relevant model objects, positions the camera at a certain distance in the direction of the light source, and renders a depth map. This will later be used by the models to compute the shadows. Gaia Sky implements an adaptive shadow mapping solution instead of a global one because the depth map resolutions needed for such a sparse scene, where there are huge distances between the objects, which are themselves very small, are simply not feasible with current hardware. For each model in view we render a single depth map using an orthographic camera that contains the whole model and minimizes the space at the margins. This depth map is used in the rendering stage to render the shadows casted on the model by itself. This maximizes the shadow quality and resolution at the expense of inter-object shadows. A possible improvement to the technique is to determine what objects may cast shadows on what objects and render the shadow maps for groups of objects which are sufficiently close together, setting some limits on what is acceptable in terms of resolution to avoid jagged shadows.

Occlusion pass. The occlusion pass renders close-up stars and models using a solid black color into a single low-resolution FBO. This is later used in the light glow post-processing effect, which *estimates* the number of visible fragments of each star and overlays a light glow effect, which spills over the rest of the geometry.

Scene Rendering Stage

The next stage is the actual scene rendering. After the update phase, the render lists are prepared to be used by the relevant scene graph renderer (SGR) object, once the pre-passes have been rendered. The responsibility of the SGR objects is to prepare the render environment (viewport, post-processing FBOs, etc.), call each render system with the relevant render list, and run the post-processing stage. Since the rendering and the post-processing may need to happen more than once—for instance, the stereoscopic mode renders the scene twice— these two stages are handled by the SGR objects.

Gaia Sky has four SGR objects. The normal SGR, the 360° panorama SGR (Figure 7.20a), the Gaia FOV SGR (Figure 7.3b), which projects both fields of view of Gaia on a single viewport, and overlays the CCD array configuration, effectively producing a visualization of what the Gaia satellite 'sees', and the stereoscopic SGR. Additionally, the VR spinoff adds an OpenVR SGR.

The render systems are in charge of handling and rendering a render list each, corresponding to a render group. The render systems are constructed and managed automatically, and the routing of the correct render list to the relevant render system is also done in

a manner transparent to the rest of the program. Each render system, then, corresponds to a render group.

Post-Processing Stage

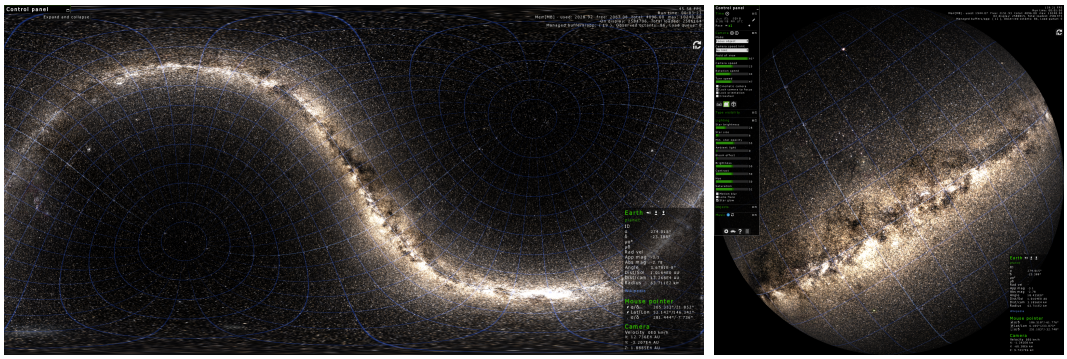
The final stage, which is actually handled by the SGR objects, is the post-processing stage, where a series of filters is defined and runs on the result of the rendering stage. Filters can be easily linked and composited by the use of ping-pong buffers. The lens flare, anti-aliasing (FXAA [Lot09] and NFAA [Sty]), light glow, camera motion blur (Figure 7.21c), bloom (Figure 7.21b) and unsharp mask effects are all run as post-processing filters.

Shaders

Gaia Sky features a wide variety of shaders, which are in charge of producing the final picture. The models are rendered using per-pixel lighting. Lines can be rendered either as primitives or using polyline quadstrips. In order to render stars as realistically as possible, we determine their visual appearance and RGB color using their measured apparent magnitude and photometric intensities in the two Gaia photometric bands. To achieve a realistic and physically based star rendering, we first compute the absolute magnitude, which is the brightness of a star at a distance of 10 parsecs, using the apparent magnitude. Then, we correct it using the extinction value found in the catalog, if present, or we apply a simple analytic galactic dust model otherwise. Finally, we convert the absolute magnitude to a flux, from which we can compute a pseudo size. This pseudo size is used for rendering purposes only. The size is passed into the star shader and used in conjunction with the star brightness, the star size, and the star minimum opacity settings to determine its representation in pixels, see Figure 7.22. If the star is close enough, it is shaded using a textured billboard. The star colors are computed from the effective temperature value in the catalog, if present, or using the conversion from BP-RP (blue minus red photometric intensity values) to effective temperature as discussed by Jordi et al. [Jor+10]. Finally, we convert the effective temperature of each star to a triplet of RGB values suitable for rendering using an algorithm based on best fits by T. Helland [Hel].

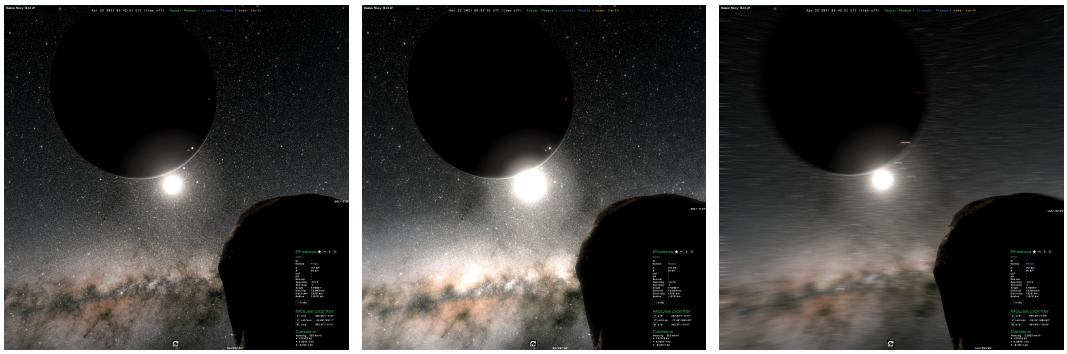
Stars and Proper Motions Rendering

Small star catalogs in Gaia Sky are represented by a single object in the octree. This object contains a buffer with the data for all stars (position, proper motion, color, magnitude, etc.), plus some supporting metadata. All these data are streamed to VRAM during the first draw call and are kept there until the catalog is removed or the application exits. This essentially means that the floating camera approach is not applied to every single star at



(a) (b)

Figure 7.20: The panorama scene graph renderer (a) generates images, which can be encoded into 360°-videos, whereas the output of the planetarium mode (b) is dedicated to fulldome planetarium shows. © 2019 IEEE.

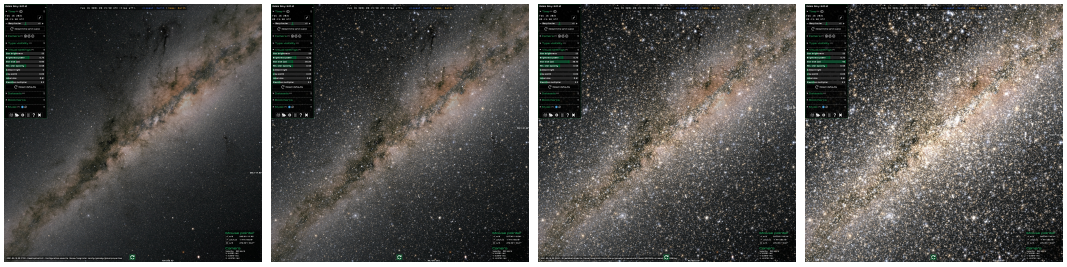


(a) (b) (c)

Figure 7.21: A scene near Mars' moon Phobos with post-processing turned off Figure 7.21a. Bloom 7.21b and camera motion blur (c) are carried out in the post-processing pass.

each frame. Instead, stars are sent to the GPU as VBOs of float attributes. A configurable set of background threads continuously index and update the stars, so that close-by stars are always well indexed and quickly accessible. Accuracy problems would become apparent only when the camera gets close to a particular star. Thus, for the few stars which are close to the camera, Gaia Sky switches to a billboard-based rendering and applies the floating camera transformation, eliminating the need to transform all stars at every frame.

Proper motions are instantaneous angular velocities of stars in the sky. Tangential velocity vectors can be calculated when combining proper motions with parallaxes. Additionally, when radial velocities are also available, 3D velocity vectors can be obtained. Gaia Sky features support for both tangential velocities and full 3D vectors. The representation of proper motions in Gaia Sky is done via a straight-line integration in the vertex stage of the star shader, given the instantaneous proper motion vector and the sim-



(a) $\sigma = 0.25$ (b) $\sigma = 0.5$ (c) $\sigma = 0.75$ (d) $\sigma = 1.0$

Figure 7.22: View toward the center of the Milky Way rendered with user-defined, increasing star pseudo-size σ. The larger σ the larger the stars appear, but due to the additive blending, the central region becomes overdense.

ulation time. This enables real-time star movement when the time speed is set sufficiently high. Since the represented star velocities are only a first-order approximation to the real motion, we limit the explorable time range, allowing us to use a static octree structure. Additionally, proper motion vectors can also be represented (Figure 7.23), with a length proportional to the magnitude of the proper motion vector, and several color options, like color-coded arrows by direction or magnitude, redshift with respect to the Sun and to the camera, etc.

Relativistic Effects Rendering

Both relativistic aberration and gravitational waves act at vertex level in Gaia Sky. The relativistic effects manager is in charge of updating the parameters and matrices needed for the relativistic effects. It is updated every frame with the camera and the current time. During the rendering stage, the shader programs are chosen depending on the active relativistic effects, the respective uniform values are set (velocity direction vector and speed for the relativistic aberration and transformation matrix, wave frequency, wave parameters, and time for the gravitational waves) and the draw call is issued normally. The shader programs include functions to apply the relativistic transformations to the positions of vertices according to the current state, passed via the uniform values. In the vertex shader, the uniform values are gathered and passed on to these functions, which compute a new position for each vertex. This is done for every vertex rendered by Gaia Sky that needs to be affected by the relativistic effects.

7.3.7 Time Control

There are two principal time frames in Gaia Sky. The real wall clock time t and the simulation time frame τ are linked using a warp speed factor, which applies to t and indicates

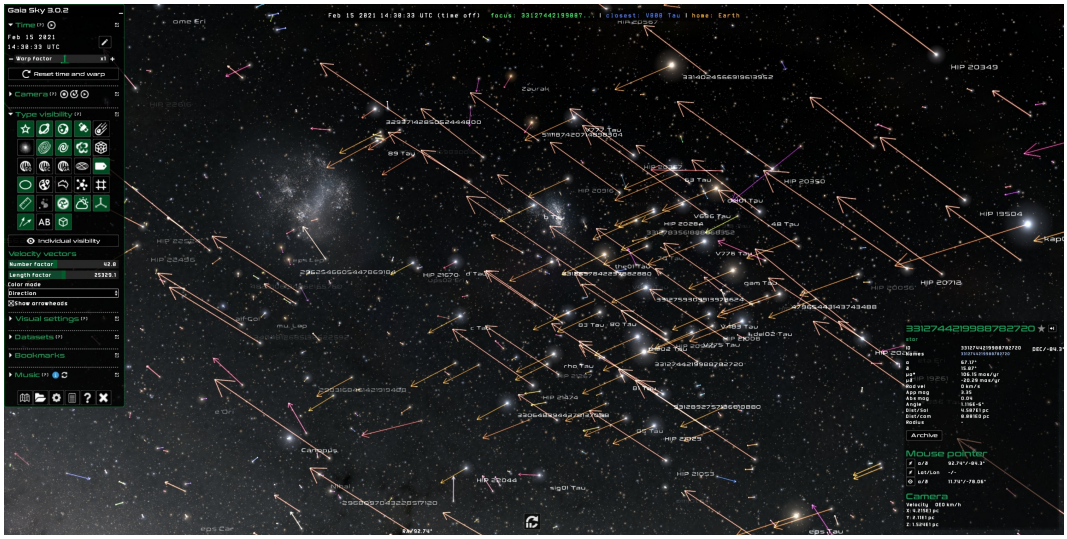


Figure 7.23: View of the Hyades star cluster with overlayed proper motions. Colors are mapped to direction and the arrow lengths are proportional to the speed. In this cluster, we have stars for which we do have their radial velocity information (pointing to the top-left) and stars for which we do not (pointing to the bottom-left).

how fast the simulation time passes with respect to the wall clock time. The warp speed factor is user-controlled and enables time-lapse effects. While t is used to update all the entities that need a direct connection to the real time, such as the integration of the forces acting on a spacecraft, τ is used to update all properties of entities that need a temporal reference frame such as the proper motions of stars, the planets' positions and orientations, or the locations of asteroids.

7.3.8 Scripting Engine

The scripting engine enables the execution of user scripts to manage and control the internal state of Gaia Sky. The script manager handles the high-level asynchronous execution of scripts and defines a maximum number of concurrent scripts by a cap in the thread pool size. Each new script runs in a separate thread and it is interpreted alongside the main thread of Gaia Sky by the interpreter. The scripting interface is of paramount importance for the production of outreach material and the creation of stories. Figure 7.24 shows an example script which includes a camera motion between the Sun and the Earth with some text overlays.

7.3.9 Camera Recording and Playback

Gaia Sky offers a native method for recording and playing camera files in order to aid in the production of audiovisual material. Camera files are recorded at a user-defined frame rate, and contain a series of values defining the time and the camera state at each frame. Each row in a camera file contains the current τ, as well as the camera's position, direction, and up vector. The frame rate is modified automatically and artificially when a camera file is being recorded. Camera files can be played back directly from the UI or using the scripting API. Additionally, a keyframe system is implemented, enabling the visual definition of camera paths by defining camera positions and orientations within the scene. This sequence of key frames is then exported to a regular camera file. The transition between positions and orientations is interpolated using a user-defined algorithm. Currently, Catmull-Rom splines and linear interpolation are available.

7.3.10 Camera Modes

There are four main camera modes in Gaia Sky, which affect camera behavior and capabilities in various ways.

The *focus mode* makes the camera automatically track a focus object. The camera normally points at the object, even though the view direction can be changed, remaining relative to the focus' position. It is possible to lock the position to the focus, so that the relative position of the camera with respect to the focus is maintained, and also its orientation so that the camera rotates with the object. The *free mode* allows for free roaming in the scene. For maximum movement freedom, a game controller with two analog joysticks can be used. The velocity scaling depends on the distance to either the closest model object, or to the origin of the reference system. The *Gaia FOV mode*, shown in Figure 7.3b, projects both fields of view of Gaia on the screen and overlays the CCD configuration in focal plane. This allows the user to see what the current simulation of the Gaia satellite sees. The *spacecraft mode*, shown in Figure 7.12, puts the user in command of a spacecraft with a realistic engine model with thrust, a power multiplier, and an optional drag. The mode brings up the spacecraft GUI, which contains visual elements to control the spacecraft parameters, as well as an attitude indicator widget with the positions of the current velocity and anti-velocity vectors. Finally, there are two camera behaviors available. The *cinematic camera* behavior implements camera movement using the steering behavior principles laid out by Raynolds [Ray99], which yields very smooth camera movements suitable for creating videos and camera paths. The *non-cinematic camera* behavior is a much more responsive implementation, more suited for interactive use.

```
from py4j.clientserver import ClientServer, JavaParameters

gateway = ClientServer(java_parameters=JavaParameters(auto_convert=True))
gs = gateway.entry_point

gs.disableInput()
gs.cameraStop()

gs.setRotationCameraSpeed(20)
gs.setTurningCameraSpeed(20)
gs.setCameraSpeed(20)

gs.goToObject("Sol", 20.0, 4.5)
gs.setHeadlineMessage("This is the Sun, our star")

gs.sleep(4)

gs.clearAllMessages()
gs.goToObject("Earth", 20.0, 6.5)
gs.setHeadlineMessage("This is the Earth, our home")

gs.enableInput()
```

Figure 7.24: Simple Gaia Sky script. First, the environment is prepared, and the values for certain camera properties are set. Then, we move the camera to the Sun and to the Earth, printing some messages on-screen. The API is fully documented in the official Gaia Sky documentation [Sagb].

7.3.11 Video Modes

There are five video output modes implemented in Gaia Sky, each mode corresponding to a different scene graph renderer, as discussed in Section 7.3.6.

The *normal* mode (the default) produces the output to screen as a single scene. The *planetarium* mode, shown in Figure 7.20b, is coupled with the frame output system to produce image sequences in azimuthal equidistant projection which can be encoded into fulldome videos. The *stereoscopic* mode renders the scene twice, where either parallel view, cross-eye, anaglyphic red-cyan, 3DTV, or VR profile (Figure 7.25) can be selected. The eye separation is a function of the distance to the focus object or can be set to a fixed value. The *VR* mode is implemented in the VR spinoff branch of Gaia Sky, and makes use of the OpenVR API, which is responsible for the necessary image transformations for the supported headsets. Finally, in the *panorama* mode (Figure 7.20a), the whole scene around the camera is pre-rendered six times with $90° \times 90°$ field-of-view into a cubemap to create 360°-videos.

7.3.12 SAMP Integration

Gaia Sky also provides a basic integration with the Simple Application Messaging Protocol (SAMP, [Tay+11]), which enables interoperability of astronomical software by sharing data and providing linked views. On startup, Gaia Sky looks for a preexisting SAMP hub. If found, a connection is attempted. If not found, Gaia Sky attempts further lookups at

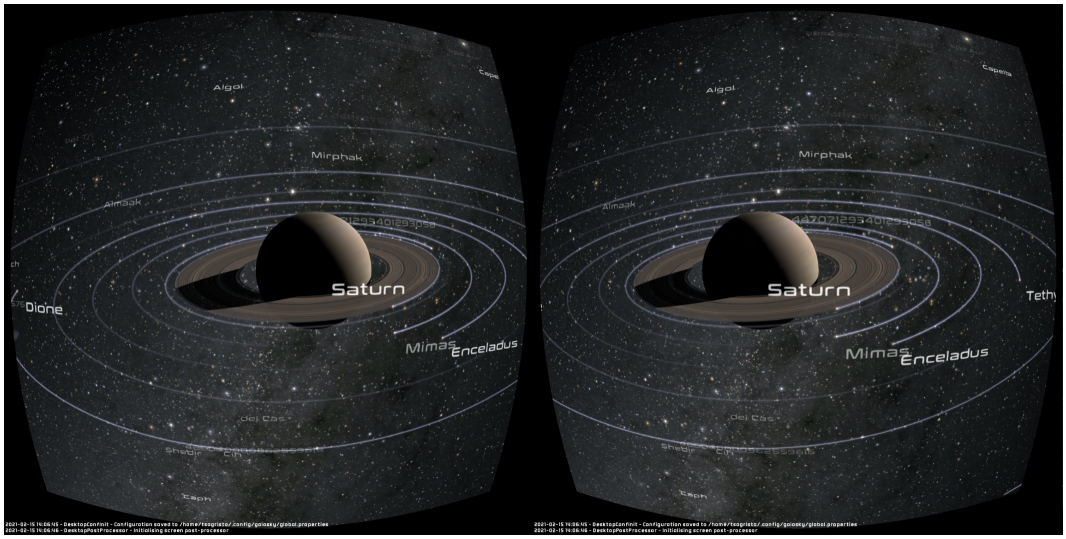


Figure 7.25: View of Saturn using the VR profile in the stereoscopic mode. The scene is rendered twice, side-by-side, and an additional lens transformation is applied in the post-processing stage.

regular intervals of ten seconds. The only format supported by Gaia Sky through SAMP is VOTable, through the STIL data loader. The implemented features are *VOTable loading*, *row highlighting*, *selection broadcast*, and *point at sky coordinate*. Since Gaia Sky only represents 3D data, some assumptions are made upon receiving tables from other programs via SAMP. If Gaia Sky does not find enough information in the table metadata to reconstruct positions, the table is discarded. SAMP enables complex use cases like loading a table into Topcat from VO, creating histograms and plots, and sending the table to Gaia Sky for closer inspection in a galactic context (even VR), while providing inter-application linked views.

7.4 Performance

Gaia Sky targets common everyday systems such as laptops and desktop computers. The minimum system requirements are rather forgiving, due to the possibility to set low settings for graphics quality. A system with an Intel Core i3 CPU, Intel HD 4000 or NVIDIA GeForce 8400 GS with 1 GB of VRAM, 4 GB RAM, and at least 1 GB of free disc space should suffice. The frame rate of Gaia Sky depends largely on the used graphics card, the graphics settings, and the type of objects that are enabled. For instance, certain graphics quality settings will fetch higher resolution textures and use higher sample counts for some effects, which will lead to lower frame rates on most systems. In the LOD approach, the draw distance has obviously a huge impact on the frame rate, as it

determines the amount of data that needs to be fetched in the background, as well as the number of stars on screen. In general, the application runs at very high frame rates on gaming-grade GPUs such as NVIDIA's GTX lines. The GTX 1070, for example, is able to pull between 150 and 200 fps with 5 million stars on screen. Low-power integrated Intel GPUs, such as the UHD 620, common in ultrabooks, run at between 30 and 60 fps, depending on what elements are enabled and/or visible. It is, however, difficult to assess the performance in a general manner, due to the sheer number of options given to the user in terms of visuals and quality settings.

The different SGRs have an obvious impact on the performance. Every loop cycle, the object model is updated only once, but depending on the SGR in use, it may be rendered more than once. The stereoscopic SGR renders the scene twice with different cameras, as does the VR SGR. The panorama SGR renders the scene six times and then processes the equirectangular projection.

The eDR3 raw dataset contains around 1.8 billion sources and weights 800 GB. In the preparation step, which is done only once, the octree is generated. Using the default eDR3 catalog (parallax relative error of 20.0% for bright stars and 1.5% for faint stars), yielding a catalog of roughly 14 million stars, and a subdivision threshold $N_\sigma = 100,000$, a system with 2 TB of RAM (only 30 TB of memory were allocated to the generation process) takes about two hours. Of these, 1.9 hours are spent reading the catalog into memory, 1 *minute* is spent actually processing and generating the octree, and the rest (25 s) for writing the results. In the case of the geometric Bayesian distances catalog, the resulting dataset contains 1.46 billion stars. The generation of this octree took about 18.5 hours, of which 12.5 hours were spent loading the data, 5.16 hours generating the octree, and the rest writing the results.

7.4.1 Performance Comparison

We have evaluated the performance of other similar systems yielding the following results. All evaluations were carried out using the same system, with an i7-7700, an NVIDIA GTX 1070, and 16 GB of RAM.

Open Space. The default star catalog in Open Space contains hundreds of thousands of stars. The frame rate we experienced was around 30–40 fps when inside the Solar System, and around 150 fps when navigating the stars. The frame rate in Open Space is inversely dependent on the speed of time, reaching 1 fps when time speed is 200 days/second. Gaia Sky can handle many more stars at higher frame rates, and a dependency on the time scale does not exist.

Celestia. Using the default settings and reasonably modern hardware, Celestia runs at a stable 60 fps. This includes, however, a limiting magnitude of 7.0 mag, which contains only 14 K stars. When the limiting magnitude is increased to the maximum setting (only 12.0 mag), the frame rate decreased to 40–50 fps. This may pose problems regarding scalability to larger star catalogs. Gaia Sky is able and ready to handle catalogs orders of magnitudes larger, without frame rate issues.

Space Engine. The performance of Space Engine is difficult to compare to that of Gaia Sky due to the heavy usage of graphical effects in the former. Space Engine needs a very powerful GPU to achieve stable frame rates. In that case, most of the time, it runs at 60 fps. However, Gaia Sky can handle many more objects, and the frame rates are typically higher.

7.5 Discussion

In this chapter, we presented a system for visualizing the star catalogs acquired by the Gaia mission, with a special focus on the big data available starting with its Data Release 2, comprising 1.46 billion stars. Due to its large size, this is the first star catalog that requires effective handling of such data. The fact that DR2+ includes brightness information and, in particular, high precision three-dimensional positional data, motivated our approach of magnitude–space level-of-detail, a data access mechanism specially tailored for the requirements of interactive stellar visualization. To handle the large range of scales numerically, we presented a respective CPU/GPU approach for camera computations, partially based on arbitrary-precision floating point calculation. We also presented our recent additions to the system, including relativistic aberration, and simulation of the visual effects of gravitational waves.

There is no absolute up or down, as Aristotle taught; no absolute position in space; but the position of a body is relative to that of other bodies. Everywhere there is incessant relative change in position throughout the universe, and the observer is always at the center of things.

— Giordano Bruno

8 Conclusion

This final chapter provides an overview of the methods and techniques introduced and presented in this thesis, along with a summary and a review of the possible future work that has already been discussed at the end of each chapter.

8.1 Contributions

Even though the two parts of this thesis may seem very different and even unrelated, there exist many ties linking them in the underlying field of astrophysics visualization. For instance, in the first part, we adapt the traditional non-inertial vector field topology of unsteady flows to inertial dynamical systems, which may or may not be translatable with a vector field. This opens the door to direct application in many domains of science, most prominently, but not restricted to, astrophysics, where the dynamics of most systems are inherently inertial. Then, we develop an integrated visual analysis methodology aimed at revealing hidden structure in dynamical systems while providing the means to better determine the appropriate and most interesting initial and transport times, using convenient, easily to interpret aggregation fields. Again, this may have applications in astrophysics, e.g., assisting the analysis and understanding of star cluster simulations. We also introduce an integrated method to quantify the "goodness" of LCS extracted as height ridges in the FTLE field, which moves us in the direction of enabling the topological analysis of complex dynamical systems with quantitative guarantees. This is equally applicable to astrophysics, as it is in many other domains of science. The second part of this thesis concerns itself with the direct visualization and handling of an astronomically large star catalog, while enabling its interactive exploration, analysis, and traversal.

The main contributions of this thesis to the fields of vector field topology and visualization of large astrometric data are as follows.

In Chapter 3, we presented a novel approach for the analysis of inertial dynamics. We have extended the concept of the FTLE to the $2n$-dimensional phase space, containing not only positions, but also velocities, resulting in the PS-FTLE. We have enabled the visualization of this new higher-dimensional field by means of constraining the contributions of position and velocity (PS-FTLE-P and PS-FTLE-V respectively), converting it from a $2n$-dimensional to a manageable n-dimensional problem. We have also presented the stacked PS-FTLE employing dimensional stacking, with the aim of providing context and guidance to the exploratory analysis process. We have also introduced the concept of multiplicity maps in the context of the PS-FTLE, in order to visualize overlapping regions derived from inertial motion. Finally, we have presented an integrated interactive framework that includes all the methods and techniques described above and enables interactive exploration and analysis of complex inertial dynamical datasets.

In Chapter 4, we introduced an approach to the visual analysis of the FTLE, by developing a set of aggregation functions that capture distinct essential properties of the underlying fields. These include basic statistical functions, height ridge properties, anti-aliasing measures, and a global region 'connectedness' assessment.

In Chapter 5, we developed and presented a methodology to quantify the importance of height ridges as extracted from the FTLE field, and their fitness as candidates to Lagrangian coherent structures. The quantification is based on the cross-flux filtering of ridges and a novel LCS consistency measure. We also introduced a novel approach to assist in the automatic determination of the best resolution that captures the LCS at the desired quality level. This process is based on an automatic adaptive domain subdivision that autonomously selects regions of interest based on the cross-flux and consistency quantities. Even though we have proven the technique to work on simple, constructed datasets, we could not meet the convergence criteria with more complex flow simulations due to them requiring very high resolutions. To that regard, more research is needed to ensure the global consistency and usefulness of the method.

Finally, in Chapter 4, we presented an integrated system to represent and visualize very large star and point-based catalogs, like the ones produced by the Gaia mission. In this fashion, we have presented MS-LOD, a level-of-detail data structure tailored to star visualization that takes into account star magnitudes to produce a seamless representation of the galaxy from any point of view. We have also addressed the inescapable floating-point precision issues encountered when rendering high dynamic distance ranges by introducing the floating camera. Finally, we have presented a logarithmic depth buffer to avoid z-buffer resolution problems due to poor depth precision at high distances.

8.2 Future Work

We can conceive many ways in which the works presented in this thesis could be expanded, the most obvious one being the additional assessment and evaluation needed in order to consider our LCS quantitative error control methodology ready for prime time. Application to more datasets is certainly required, and additional adjustments to the technique may also be needed, possibly by introducing a better initial ridge extraction method. In a general sense, the first part of this thesis provides the ground work to enable flow visualization techniques to be applied to additional fields of science and engineering, such as astrophysics. The methods presented have so far been applied to comparatively small datasets. Since these expand and built upon vector field topology and the FTLE, which typically has a very high computational cost (more so when paired with ridge extraction and other LCS visualizations), we do not expect direct application to the field of astrophysics anytime soon. An interesting direction for future research would be in the development of domain-specific acceleration techniques to enable this gap between small, contained systems to very large datasets to be bridged. The second part, with Gaia Sky, is an ongoing endeavor and is in active development. New features are introduced with each new version, and every new Gaia data release is an opportunity to not only release a new batch of catalogs, but also to develop and integrate the new complimentary data products such as spectra, variable stars, solar system objects, integrated star trajectories, exoplanets, or extrasolar star systems, accompanied by respective visualization techniques.

Acronyms

API	Application programming interface
ASCII	American standard code for information interchange
BDRF	Bidirectional reflectance distribution function
BP	Blue photometer
BRDF	Bidirectional reflectance distribution function
BSDF	Bidirectional scattering distribution function
BSP	Binary space partitioning
BTDF	Bidirectional transmittance distribution function
CCD	Charge-coupled device
CG	Computer graphics
CLOD	Continuous level-of-detail
CPU	Central processing unit
CSV	Comma-separated values
CUDA	Compute unified device architecture
DLOD	Discrete level-of-detail
DPAC	Gaia data processing and analysis consortium
DR1	(Gaia) data release 1
DR2	(Gaia) data release 2
DR3	(Gaia) data release 3
ECS	Entity component system
eDR3	Early Gaia data release 3
FBO	Frame buffer object
FITS	Flexible image transport system
FOV	Field of view
FPS	Frames per second

FTLE	Finite-time Lypaunov exponent
FXAA	Fast approximate anti-aliasing
GBIN	Gaia binary format
GCA	Graphics and compute array
GLSL	OpenGL shading language
GP	Graphics processor
GPU	Graphics processing unit
HiDPI	High dots per inch (high pixel density)
HTML	Hypertext markup language
IVP	Initial value problem
JNI	Java native interface
JSON	JavaScript object notation
JVM	Java virtual machine
LE	Lypaunov exponent
LOD	Level-of-detail
MP	Multidimensional projection
MPCDI	Multiple projector common data interchange
MS-LOD	Magnitude–space level-of-detail
NDC	Normalized device coordinates
NFAA	Normal filter anti-aliasing
OCDR2	Open clusters DR2 catalog
ODE	Ordinary differential equation
PBR	Physically based rendering
PDE	Partial differential equation
PS-FTLE	Phase-space finite-time Lypaunov exponent
PVS	Potentially Visible Set
RAM	Random access memory
RAVE	Radial velocity experiment
REST	Representational state transfer
RGB	Red, green, blue (color model)
RK4	Fourth order Runge-Kutta
ROI	Region of interest
RP	Red photometer
SAMP	Simple application messaging protocol
SGR	Scene graph renderer
STIL	Starlink tables infrastructure library
TMA	Texture mapping unit

UCD	Unified content descriptor
UI	User interface
VAO	Vertex array object
VBO	Vertex buffer object
VFT	Vector field topology
VO	Virtual observatory
VR	Virtual reality
VRAM	Video random access memory

Bibliography

[4D2] 4D2U-Project-Team. *Mitaka*. https://4d2u.nao.ac.jp/html/program/mitaka. [Online; accessed 22-February-2022] (cit. on p. 131).

[AGL05] J. Ahrens, B. Geveci, and C. Law. "ParaView: An End-User Tool for Large Data Visualization". *Visualization Handbook*, 2005 (cit. on p. 26).

[Aig+07] W. Aigner, A. Bertone, S. Miksch, C. Tominski, and H. Schumann. "Towards a Conceptual Framework for Visual Analytics of Time and Time-Oriented Data". In: *Proceedings of 2007 Winter Simulation Conference*. 2007, pp. 721–729 (cit. on p. 26).

[AM04] T. Aila and V. Miettinen. "dPVS: an Occlusion Culling System for Massive Dynamic Environments". *IEEE Computer Graphics and Applications* 24, 2004, pp. 86–97 (cit. on p. 121).

[Ame+11] M. Ament, S. Frey, F. Sadlo, T. Ertl, and D. Weiskopf. "GPU-Based Two-Dimensional Flow Simulation Steering using Coherent Structures". In: *Proceedings of Second International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering*. 2011 (cit. on p. 28).

[Asi93] D. Asimov. "Notes on the Topology of Vector Fields and Flows". *Technical report, NASA Ames Research Center. RNR-93-003*, 1993 (cit. on pp. 17, 18).

[Aur+97] E. Aurell, G. Boffetta, A. Crisanti, G. Paladin, and A. Vulpiani. "Predictability in the Large: an Extension of the Concept of Lyapunov Exponent". *Journal of Physics A: Mathematical and General* 30:1, 1997, pp. 1–26 (cit. on pp. 26, 103).

[Axe+17] E. Axelsson, J. Costa, C. Silva, C. Emmart, A. Bock, and A. Ynnerman. "Dynamic Scene Graph: Enabling Scaling, Positioning, and Naviga-

tion in the Universe". *Computer Graphics Forum* 36:3, 2017, pp. 459–468 (cit. on pp. 132, 138).

[Bai+21] C. A. L. Bailer-Jones, J. Rybizki, M. Fouesneau, M. Demleitner, and R. Andrae. "Estimating Distances from Parallaxes. V. Geometric and Photogeometric Distances to 1.47 Billion Stars in Gaia Early Data Release 3". *The Astronomical Journal* 161:3, 2021, p. 147 (cit. on pp. 134, 135).

[Bav+05] L. Bavoil, S. P. Callahan, P. J. Crossno, J. Freire, C. E. Scheidegger, C. T. Silva, and H. T. Vo. "VisTrails: Enabling Interactive Multiple-View Visualizations". In: *Proceedings of IEEE Visualization*. 2005, pp. 135–142 (cit. on p. 26).

[BDH96] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. "The Quickhull Algorithm for Convex Hulls". *ACM Transactions on Mathematical Software* 22:4, 1996, pp. 469–483 (cit. on p. 49).

[Boc+17] A. Bock, E. Axelsson, K. Bladin, J. Costa, G. Payne, M. Territo, J. Kilby, E. Myers, M. Kuznetsova, C. Emmart, and A. Ynnerman. "OpenSpace: An open-source Astrovisualization Framework". *The Journal of Open Source Software* 2:15, 2017, p. 281 (cit. on p. 131).

[Bre82] P. Bretagnon. "Theory for the Motion of All the Planets–The VSOP82 Solution". *Astronomy & Astrophysics* 114, 1982, pp. 278–288 (cit. on p. 150).

[BS02] H. Batagelo and W. Shin-Ting. "Dynamic Scene Occlusion Culling Using a Regular Grid". In: *Proceedings. XV Brazilian Symposium on Computer Graphics and Image Processing*. 2002, pp. 43–50 (cit. on p. 122).

[Bür+07] R. Bürger, P. Muigg, M. Ilcík, H. Doleisch, and H. Hauser. "Integrating Local Feature Detectors in the Interactive Visual Analysis of Flow Simulation Data." In: *Proceedings of Eurographics / IEEE VGTC Conference on Visualization*. 2007, pp. 171–178 (cit. on p. 26).

[BV18] M. A. Breddels and J. Veljanoski. "Vaex: Big Data Exploration in the Era of Gaia". *Astronomy & Astrophysics* 618, 2018, A13 (cit. on p. 130).

[Chi+12] H. Childs, E. Brugger, B. Whitlock, J. Meredith, S. Ahern, K. Bonnell, M. Miller, G. H. Weber, C. Harrison, T. Fogal, C. Garth, A. S, E. W. Bethel, M. Durant, D. Camp, J. M. Favre, O. Rübel, P. Navrátil, M. Wheeler, P. Selby, and F. Vivodtzev. "VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data". In: *Proceedings of SciDAC*. 2012 (cit. on p. 26).

[CM84] W. S. Cleveland and R. McGill. "Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods". *Journal of the American Statistical Association* 79:387, 1984, pp. 531–554 (cit. on p. 12).

[CT65] J. Cooley and J. Tukey. "An Algorithm for the Machine Computation of Complex Fourier Series". *Mathematics of Computation* 19, 1965, pp. 297–301 (cit. on p. 64).

[CT97] S. R. Coorg and S. J. Teller. "Real-time Occlusion Culling for Models With Large Occluders". *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, 1997 (cit. on p. 121).

[Deh01] W. Dehnen. "Towards Optimal Softening in Three-Dimensional N-Body Codes – I. Minimizing the Force Error". *Monthly Notices of the Royal Astronomical Society* 324:2, 2001, pp. 273–291 (cit. on p. 50).

[Der+04] S. Derriere, N. Gray, J. C. McDowell, R. Mann, F. Ochsenbein, P. Osuna, A. Preite Martinez, G. Rixon, and R. Williams. "UCD in the IVOA Context". 314, 2004, p. 315 (cit. on p. 149).

[DGH03] H. Doleisch, M. Gasser, and H. Hauser. "Interactive Feature Specification for Focus+Context Visualization of Complex Simulation Data". In: *Proceedings of the Symposium on Data Visualisation*. 2003, pp. 239–248 (cit. on p. 26).

[Dol+04] H. Doleisch, M. Mayer, M. Gasser, R. Wanker, and H. Hauser. "Case Study: Visual Analysis of Complex, Time-Dependent Simulation Results of a Diesel Exhaust System". In: *Proceedings of Eurographics / IEEE VGTC Symposium on Visualization*. 2004, pp. 91–96 (cit. on p. 26).

[Dur+00] F. Durand, G. Drettakis, J. Thollot, and C. Puech. *Conservative Visibility Preprocessing using Extended Projections*. 2000 (cit. on p. 121).

[Ebe96] D. Eberly. *Ridges in Image and Data Analysis. Computational Imaging and Vision.* Vol. 7. Kluwer Academic Publishers, 1996 (cit. on pp. 27, 53, 90).

[EBN11] J. Elseberg, D. Borrmann, and A. Nüchter. "Efficient Processing of Large 3D Point Clouds". In: *2011 XXIII International Symposium on Information, Communication and Automation Technologies*. 2011, pp. 1–7 (cit. on p. 123).

[ES] Evans and Sutherland. *Digistar*. https://www.es.com/digistar. [Online; accessed 23-February-2022] (cit. on p. 131).

[FG98] A. Fuhrmann and E. Groller. "Real-time Techniques for 3D Flow Visualization". In: *Proceedings Visualization '98 (Cat. No.98CB36276)*. 1998, pp. 305–312 (cit. on p. 16).

[FKN80] H. Fuchs, Z. M. Kedem, and B. F. Naylor. "On Visible Surface Generation by a Priori Tree Structures". In: *Computer Graphics*. 1980, pp. 124–133 (cit. on p. 124).

[FMJ] FMJ-Software. *StarStrider*. https://www.fmjsoft.com/starstrider.html. [Online; accessed 22-February-2022] (cit. on p. 131).

[Fol+90] J. Foley, A. Van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley Systems Programming Series. Addison-Wesley, 1990 (cit. on p. 120).

[FP01] J. Furst and S. Pizer. "Marching ridges". In: *Proceedings of 2001 IASTED International Conference on Signal and Image Processing*. 2001, pp. 22–26 (cit. on p. 27).

[Gai+18] Gaia-Collaboration, A. G. A. Brown, A. Vallenari, and Prusti. "Gaia Data Release 2". *Astronomy & Astrophysics* 616, 2018, A1 (cit. on p. 135).

[Gar+07] C. Garth, F. Gerhardt, X. Tricoche, and H. Hagen. "Efficient Computation and Visualization of Coherent Structures in Fluid Flow Applications". *IEEE Transactions on Visualization and Computer Graphics* 13:6, 2007, pp. 1464–71 (cit. on pp. 58, 92).

[Gar+09] C. Garth, G.-S. Li, X. Tricoche, C. D. Hansen, and H. Hagen. "Visualization of Coherent Structures in Transient 2D Flows". In: *Topology-Based Methods in Visualization II*. Springer, 2009, pp. 1–13 (cit. on p. 27).

[GP15] D. Garaboa-Paz and V. Pérez-Muñuzuri. "A Method to Calculate Finite-Time Lyapunov Exponents for Inertial Particles in Incompressible Flows". *Nonlinear Processes in Geophysics* 22:5, 2015, pp. 571–577 (cit. on p. 25).

[GT14] T. Günther and H. Theisel. "Vortex Cores of Inertial Particles". *IEEE Transactions on Visualization and Computer Graphics* 20:12, 2014, pp. 2535–2544 (cit. on p. 25).

[GT15] T. Günther and H. Theisel. "Finite-Time Mass Separation for Comparative Visualizations of Inertial Particles". *Computer Graphics Forum* 34:3, 2015, pp. 471–480 (cit. on p. 25).

[GT16a] T. Günther and H. Theisel. "Inertial Steady 2D Vector Field Topology". *Computer Graphics Forum* 35:2, 2016, pp. 455–466 (cit. on p. 25).

[GT16b] T. Günther and H. Theisel. "Source Inversion by Forward Integration in Inertial Flows". *Computer Graphics Forum* 35:3, 2016, pp. 371–380 (cit. on p. 25).

[GT17] T. Günther and H. Theisel. "Backward Finite-Time Lyapunov Exponents in Inertial Flows". *IEEE Transactions on Visualization and Computer Graphics* 23:1, 2017, pp. 970–979 (cit. on p. 28).

[Gün+13] T. Günther, A. Kuhn, B. Kutz, and H. Theisel. "Mass-Dependent Integral Curves in Unsteady Vector Fields". *Computer Graphics Forum* 32:3, 2013, pp. 211–220 (cit. on p. 25).

[Hal01] G. Haller. "Distinguished Material Surfaces and Coherent Structures in Three-Dimensional Fluid Flows". *Physica D* 149, 2001, pp. 248–277 (cit. on pp. 23, 24, 35, 57, 85, 86).

[Hal11] G. Haller. "A Variational Theory of Hyperbolic Lagrangian Coherent Structures". *Physica D: Nonlinear Phenomena* 240:7, 2011, pp. 574–598 (cit. on p. 86).

[Hei+16] C. Heine, H. Leitte, M. Hlawitschka, F. Iuricich, L. De Floriani, G. Scheuermann, H. Hagen, and C. Garth. "A Survey of Topology-based Methods in Visualization". *Computer Graphics Forum* 35:3, 2016, pp. 643–667 (cit. on p. 21).

[Hel] T. Helland. *Teff to RGB Conversion*. `https://www.tannerhelland.com/4435/convert-temperature-rgb-algorithm-code/`. [Online; accessed 23-February-2022] (cit. on p. 154).

[HH89] J. Helman and L. Hesselink. "Representation and Display of Vector Field Topology in Fluid Flow Data Sets". *Computer* 22:8, 1989, pp. 27–36 (cit. on p. 16).

[HH91] J. Helman and L. Hesselink. "Visualizing Vector Field Topology in Fluid Flows". *IEEE Computer Graphics and Applications* 11:3, 1991, pp. 36–46 (cit. on p. 16).

[Hla+11] M. Hlawatsch, J. Vollrath, F. Sadlo, and D. Weiskopf. "Coherent Structures of Characteristic Curves in Symmetric Second Order Tensor Fields". *IEEE Transactions on Visualization and Computer Graphics* 17:6, 2011, pp. 781–794 (cit. on p. 28).

[Høg+00] E. Høg, C. Fabricius, V. V. Makarov, S. Urban, T. Corbin, G. Wycoff, U. Bastian, P. Schwekendiek, and A. Wicenec. "The Tycho-2 Catalogue of the 2.5 Million Brightest Stars". *Astronomy & Astrophysics* 355, 2000, pp. 27–30 (cit. on p. 129).

[ICG86] D. S. Immel, M. F. Cohen, and D. P. Greenberg. "A Radiosity Method for Non-Diffuse Environments". *ACM SIGGRAPH Computer Graphics* 20:4, 1986, pp. 133–142 (cit. on p. 109).

[IEE19] IEEE-Computer-Society. "IEEE Standard for Floating-Point Arithmetic". *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, 2019, pp. 1–84 (cit. on p. 126).

[Jar] K. Jardine. *Making a Galaxy Map*. `http://galaxymap.org/drupal/node/256`. [Online; accessed 23-February-2022] (cit. on p. 134).

[JL97] B. Jobard and W. Lefer. "Creating Evenly-Spaced Streamlines of Arbitrary Density". In: *Visualization in Scientific Computing '97*. Ed. by W. Lefer and M. Grave. Springer Vienna, Vienna, 1997, pp. 43–55 (cit. on p. 16).

[JMI15] H. J. Johnson, M. McCormick, and L. Ibanez. *The ITK Software Guide*. Kitware, Inc., Clifton Park, NY, USA, 2015 (cit. on p. 26).

[JOP+01] E. Jones, T. Oliphant, P. Peterson, et al. *SciPy: Open source scientific tools for Python*. `https://www.scipy.org`. [Online; accessed 23-February-2022]. 2001 (cit. on p. 49).

[Jor+10] C. Jordi, M. Gebran, J. M. Carrasco, J. de Bruijne, H. Voss, C. Fabricius, J. Knude, A. Vallenari, R. Kohley, and A. Mora. "Gaia Broad Band Photometry". *Astronomy & Astrophysics* 523, A48, 2010, A48 (cit. on p. 154).

[JPN15] P. Joia, F. Petronetto, and L. G. Nonato. "Uncovering Representative Groups in Multidimensional Projections". *Computer Graphics Forum* 34:3, 2015, pp. 281–290 (cit. on p. 36).

[Kaj86] J. T. Kajiya. "The Rendering Equation". *ACM SIGGRAPH Computer Graphics*. SIGGRAPH '86 20:4, 1986, pp. 143–150 (cit. on p. 109).

[Kas+09] J. Kasten, C. Petz, I. Hotz, B.R. Noack, and H.-C. Hege. "Localized Finite-time Lyapunov Exponent for Unsteady Flow Analysis". In: *Proceedings of International Workshop on Vision, Modeling and Visualization*. 2009, pp. 265–274 (cit. on p. 28).

[Kin+16] G. Kindlmann, C. Chiw, N. Seltzer, L. Samuels, and J. Reppy. "Diderot: a Domain-Specific Language for Portable Parallel Scientific Visualization and Image Analysis". *IEEE Transactions on Visualization and Computer Graphics* 22:1, 2016, pp. 867–876 (cit. on pp. 26, 28).

[Kin+18] G. Kindlmann, C. Chiw, T. Huynh, A. Gyulassy, J. Reppy, and P.-T. Bremer. "Rendering and Extracting Extremal Features in 3D Fields". *Computer Graphics Forum* 37:3, 2018, pp. 525–536 (cit. on p. 28).

[Kla+10] S. Klashed, P. Hemingsson, C. Emmart, M. Cooper, and A. Ynnerman. "Uniview: Visualizing the Universe". In: *Proceedings of Eurographics 2010 - Areas Papers*. 2010 (cit. on p. 131).

[Kli18] S. A. Klioner. "Gaia-Like Astrometry and Gravitational Waves". *Classical and Quantum Gravity* 35:4, 2018 (cit. on p. 144).

[Klö+13] A. Klöckner, N. Pinto, B. Catanzaro, Y. Lee, P. Ivanov, and A. Fasih. "PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation". *Parallel Computing* 38:3, 2013, pp. 157–174 (cit. on pp. 48, 77).

[Kuh+14] A. Kuhn, W. Engelke, C. Rössl, M. Hadwiger, and H. Theisel. "Time Line Cell Tracking for the Approximation of Lagrangian Coherent Structures with Subgrid Accuracy". *Computer Graphics Forum* 33:1, 2014, pp. 222–234 (cit. on p. 28).

[LC87] W. Lorensen and H. Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm". *ACM SIGGRAPH Computer Graphics* 21:4, 1987, pp. 163–169 (cit. on p. 27).

[LFH06] Y. Li, C. w. Fu, and A. Hanson. "Scalable WIM: Effective Exploration in Large-scale Astrophysical Environments". *IEEE Transactions on Visualization and Computer Graphics* 12:5, 2006, pp. 1005–1012 (cit. on p. 138).

[Lib] LibGDX-Development-Team. *libGDX*. `https://libgdx.com`. [Online; accessed 22-February-2022] (cit. on p. 144).

[LKH09] A. Lie, J. Kehrer, and H. Hauser. "Critical Design and Realization Aspects of Glyph-based 3D Data Visualization". In: 2009, pp. 27–34 (cit. on p. 12).

[Lot09] T. Lottes. *FXAA White paper*. `https://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf`. 2009 (cit. on p. 154).

[LV10] J. A. Lee and M. Verleysen. *Nonlinear Dimensionality Reduction*. Springer Science & Business Media, 2010 (cit. on pp. 36, 37).

[LWJ] LWJGL-Development-Team. *LWJGL*. `https://www.lwjgl.org`. [Online; accessed 22-February-2022] (cit. on p. 144).

[LWW90] J. LeBlanc, M. O. Ward, and N. Wittels. "Exploring N-Dimensional Databases". In: *Proceedings of the First IEEE Conference on Visualization*. 1990, pp. 230–237 (cit. on p. 41).

[Mac+16] G. M. Machado, S. Boblest, T. Ertl, and F. Sadlo. "Space-Time Bifurcation Lines for Extraction of 2D Lagrangian Coherent Structures". *Computer Graphics Forum* 35:3, 2016, pp. 91–100 (cit. on p. 21).

[MKA08] T. Müller, A. King, and D. Adis. "A Trip to the End of the Universe and the Twin "Paradox"". *American Journal of Physics* 76:4, 2008, pp. 360–373 (cit. on p. 144).

[Moi+17] A. Moitinho, A. Krone-Martins, H. Savietto, M. Barros, C. Barata, A. Falcão, T. Fernandes, J. Alves, A. F. Silva, M. Gomes, J. Bakker, A. G. A. Brown, J. González, G. Gracia-Abril, R. Gutierrez-Sanchez, J. Hernandez, S. Jordan, X. Luri, B. Merín, A. Vallenari, and A. Sagristà. "Gaia Data Release 1: The Archive Visualisation Service". 605, 2017, A52 (cit. on p. 130).

[Mor+16] K. Moreland, C. Sewell, W. Usher, L. Lo, J. Meredith, D. Pugmire, J. Kress, H. Schroots, K. Ma, H. Childs, M. Larsen, C. Chen, R. Maynard, and B. Geveci. "VTK-m: Accelerating the Visualization Toolkit for Massively Threaded Architectures". *IEEE Computer Graphics and Applications* 36:3, 2016, pp. 48–58 (cit. on p. 26).

[Och+13] F. Ochsenbein, R. Williams, C. Davenhall, M. Demleitner, D. Durand, P. Fernique, D. Giaretta, R. Hanish, T. McGlynn, A. Szalay, M. Taylor, and A. Wicenec. *VOTable Format Definition*. `https://www.ivoa.net/documents/votable/`. [Online; accessed 23-February-2022]. 2013 (cit. on p. 149).

[ONe05] S. O'Neil. *Accurate Atmospheric Scattering*. GPU Gems. Pearson Addison Wesley Prof, 2005 (cit. on p. 148).

[PD09] J. Peng and J. O. Dabiri. "Transport of Inertial Particles by Lagrangian Coherent Structures: Application to Predator–Prey Interaction in Jellyfish Feeding". *Journal of Fluid Mechanics* 623, 2009, pp. 75–84 (cit. on pp. 25, 40).

[Per+97] M. A. C. Perryman, L. Lindegren, J. Kovalevsky, E. Hoeg, U. Bastian, P. L. Bernacca, M. Crézé, F. Donati, M. Grenon, M. Grewing, F. van Leeuwen, H. van der Marel, F. Mignard, C. A. Murray, R. S. Le Poole, H. Schrijver, C. Turon, F. Arenou, M. Froeschlé, and C. S. Petersen. "The HIPPARCOS Catalogue". *Astronomy & Astrophysics* 323, 1997, pp. 49–52 (cit. on p. 129).

[Pob+11] A. Pobitzer, R. Peikert, R. Fuchs, B. Schindler, A. Kuhn, H. Theisel, K. Matković, and H. Hauser. "The State of the Art in Topology-Based Visualization of Unsteady Flow". *Computer Graphics Forum* 30:6, 2011, pp. 1789–1811 (cit. on pp. 21, 22).

[PS] R. Peikert and F. Sadlo. "Height Ridge Computation and Filtering for Visualization". In: *Proceedings of IEEE VGTC Pacific Visualization Symposium 2008*, pp. 119–126 (cit. on p. 27).

[RAV+17] RAVE-Collaboration, A. Kunder, G. Kordopatis, M. Steinmetz, and T. Zwitter. "The Radial Velocity Experiment (RAVE): Fifth Data Release". *The Astronomical Journal* 153:2, 2017, p. 75 (cit. on p. 129).

[Ray99] C. Raynolds. "Steering Behaviors for Autonomous Characters". In: *Proceedings of Game Developers Conference*. 1999, pp. 763–782 (cit. on p. 158).

[RRV14] S. G. Raben, S. D. Ross, and P. P. Vlachos. "Experimental Determination of Three-Dimensional Finite-Time Lyapunov Exponents in Multi-Component Flows". *Experiments in Fluids* 55:10, 2014, pp. 1–6 (cit. on p. 25).

[RS] S. A. Rodionov and N. Y. Sotnikova. "Optimal Choice of the Softening Length and Time Step in N-Body Simulations". *Astronomy Reports* 49:6, pp. 470–476 (cit. on p. 50).

[Saga] A. Sagristà. *Gaia Sky Homepage*. `https://zah.uni-heidelberg.de/gaia/outreach/gaiasky`. [Online; accessed 22-February-2022] (cit. on p. 132).

[Sagb] A. Sagristà. *Gaia Sky Online Documentation*. `https://gaia.ari.uni-heidelberg.de/gaiasky/docs`. [Online; accessed 23-February-2022] (cit. on pp. 151, 159).

[Sag+17] A. Sagristà, S. Jordan, A. Just, F. Dias, L. G. Nonato, and F. Sadlo. "Topological Analysis of Inertial Dynamics". *IEEE Transactions on Visualization and Computer Graphics* 23:1, 2017, pp. 950–959 (cit. on pp. 4, 5).

[Sag+19] A. Sagristà, S. Jordan, T. Müller, and F. Sadlo. "Gaia Sky: Navigating the Gaia Catalog". *IEEE Transactions on Visualization and Computer Graphics* 25:1, 2019, pp. 1070–1079 (cit. on p. 5).

[SBR16] M. Sudharsan, S. L. Brunton, and J. J. Riley. "Lagrangian Coherent Structures and Inertial Particle Dynamics". *Physical Review E* 93, 3 2016 (cit. on p. 25).

[Sch+00] G. Schaufler, J. Dorsey, X. Decoret, and F. X. Sillion. "Conservative Volumetric Visibility with Occluder Fusion". In: *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics*. 2000, pp. 229–238 (cit. on p. 121).

[Sch+07] M. Schlemmer, I. Hotz, B. Hamann, F. Morr, and H. Hagen. "Priority Streamlines: A context-based Visualization of Flow Fields". *Eurographics / IEEE VGTC Symposium on Visualization*, 2007, pp. 1–8 (cit. on p. 16).

[Sch+97] G. Scheuermann, H. Hagen, H. Kruger, M. Menzel, and A. Rockwood. "Visualization of Higher Order Singularities in Vector Fields". In: *Proceedings Visualization*. 1997, pp. 67–74 (cit. on p. 17).

[Sch+99] M. Schulz, F. Reck, W. Bertelheimer, and T. Ertl. "Interactive visualization of fluid dynamics simulations in locally refined cartesian grids". In: *Proceedings Visualization*. 1999, pp. 413–553 (cit. on p. 26).

[Sch69] R. Schumacher. *Study for Applying Computer-generated Images to Visual Simulation*. Air Force Human Resources Laboratory, Training Research Division, Air Force Systems Command, 1969 (cit. on p. 124).

[SH09] T. Sapsis and G. Haller. "Inertial Particle Dynamics in a Hurricane". *Journal of the Atmospheric Sciences* 66:8, 2009, pp. 2481–2492 (cit. on p. 25).

[Shi+06] K. Shi, H. Theisel, T. Weinkauf, H. Hauser, H.-C. Hege, H.-P. Seidel, B. Santos, T. Ertl, K. Joy, D. Fellner, T. Möller, and S. Spencer. "Path Line Oriented Topology for Periodic 2D Time-Dependent Vector Fields". *Eurographics / IEEE VGTC Symposium on Visualization*, 2006, pp. 139–146 (cit. on p. 21).

[Shi+07] K. Shi, H. Theisel, H. Hauser, T. Weinkauf, K. Matkovic, H. C. Hege, and H. Seidel. "Path Line Attributes - an Information Visualization Approach to Analyzing the Dynamic Behavior of 3D Time-Dependent Flow Fields". In: *Topology-Based Methods in Visualization II*. 2007, pp. 75–88 (cit. on p. 26).

[SJS20] A. Sagristà, S. Jordan, and F. Sadlo. "Visual Analysis of the Finite-Time Lyapunov Exponent". *Computer Graphics Forum* 39:3, 2020, pp. 331–342 (cit. on pp. 4, 5).

[SKE14] F. Sadlo, G. K. Karch, and T. Ertl. "Topological Features in Time-Dependent Advection-Diffusion Flow". In: *Topological Methods in Data Analysis and Visualization III*. Springer International Publishing, 2014, pp. 217–231 (cit. on p. 28).

[SLM05] S. Shadden, F. Lekien, and J. Marsden. "Definition and Properties of Lagrangian Coherent Structures from Finite-Time Lyapunov Exponents in Two-Dimensional Aperiodic Flows". *Physica D: Nonlinear Phenomena* 212:3–4, 2005, pp. 271–304 (cit. on pp. 20, 24, 35, 49, 57, 72, 85, 88, 89, 99).

[Sma67] S. Smale. "Differntiable Dynamical Systems". *Bulletin of the American Mathematical Society* 73, 1967, pp. 747–817 (cit. on p. 63).

[SML04] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit - An Object-Oriented Approach To 3D Graphics*. Fourth. Prentice Hall, 2004 (cit. on p. 26).

[SP07] F. Sadlo and R. Peikert. "Efficient Visualization of Lagrangian Coherent Structures by Filtered AMR Ridge Extraction". *IEEE Transactions on Visualization and Computer Graphics* 13:6, 2007, pp. 1456–1463 (cit. on pp. 28, 58, 86, 91, 92).

[SP09] F. Sadlo and R. Peikert. "Visualizing Lagrangian Coherent Structures and Comparison to Vector Field Topology". In: *Topology-Based Methods in Visualization II*. Springer, 2009, pp. 15–29 (cit. on p. 28).

[SPH11] T. Sapsis, J. Peng, and G. Haller. "Instabilities on Prey Dynamics in Jellyfish Feeding". *Bulletin of Mathematical Biology* 73:8, 2011, pp. 1841–1856 (cit. on p. 25).

[SRP11] F. Sadlo, A. Rigazzi, and R. Peikert. "Time-Dependent Visualization of Lagrangian Coherent Structures by Grid Advection". In: *Topology-Based Methods in Visualization II*. Springer Berlin Heidelberg, 2011, pp. 151–165 (cit. on pp. 28, 58).

[Sty] Styves. *NFAA – A Post-Process Anti-Aliasing Filter*. `https://bit.ly/2FNvozp`. [Online; accessed 23-February-2022] (cit. on p. 154).

[SW10] F. Sadlo and D. Weiskopf. "Time-Dependent 2D Vector Field Topology: An Approach Inspired by Lagrangian Coherent Structures". *Computer Graphics Forum* 29:1, 2010, pp. 88–100 (cit. on pp. 21, 28).

[Tay+11] M. Taylor, T. Boch, M. Fitzpatrick, A. Allan, L. Paioro, J. Taylor, and D. Tody. "IVOA Recommendation: SAMP - Simple Application Messaging Protocol version 1.3", 2011 (cit. on pp. 130, 159).

[Tay05] M. Taylor. "TOPCAT & STIL: Starlink Table/VOTable Processing Software". In: *Proceedings of Astronomical Data Analysis Software and Systems XIV*. Vol. 347. Astronomical Society of the Pacific Conference Series. 2005 (cit. on pp. 130, 149).

[TB96] G. Turk and D. Banks. "Image-Guided Streamline Placement". In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. Association for Computing Machinery, New York, NY, USA, 1996, pp. 453–460 (cit. on p. 16).

[Tel92] S. J. Teller. "Visibility Computations in Densely Occluded Polyhedral Environments". PhD thesis. EECS Department, University of California, Berkeley, 1992 (cit. on p. 120).

[The+04] H. Theisel, T. Weinkauf, H.-C. Hege, and H.-P. Seidel. "Stream Line and Path Line Oriented Topology for 2D Time-Dependent Vector Fields". In: *IEEE Visualization*. 2004, pp. 321–328 (cit. on p. 21).

[Tie+18] J. Tierny, G. Favelier, J. A. Levine, C. Gueunet, and M. Michaux. "The Topology Toolkit". *IEEE Transactions on Visualization and Computer Graphics* 24:1, 2018, pp. 832–842 (cit. on p. 26).

[Tri+12a] X. Tricoche, C. Garth, A. Sanderson, and K. Joy. "Visualizing Invariant Manifolds in Area-Preserving Maps". In: *Topological Methods in Data Analysis and Visualization II*. Springer, 2012, pp. 109–124 (cit. on p. 28).

[Tri+12b] X. Tricoche, M. Hlawitschka, S. Barakat, and C. Garth. "Beyond Topology: A Lagrangian Metaphor to Visualize the Structure of 3D Tensor Fields". *New Developments in the Visualization and Processing of Tensor Fields*, 2012 (cit. on p. 28).

[Üff+12] M. Üffinger, F. Sadlo, M. Kirby, C. Hansen, and T. Ertl. "FTLE Computation Beyond First-Order Approximation". In: *Proceedings of Eurographics / IEEE VGTC Conference on Visualization*. 2012, pp. 61–64 (cit. on p. 28).

[ÜSE13] M. Üffinger, F. Sadlo, and T. Ertl. "A Time-Dependent Vector Field Topology Based on Streak Surfaces". *IEEE Transactions on Visualization and Computer Graphics* 19:3, 2013, pp. 379–392 (cit. on pp. 21, 28).

[VH08] L. Van der Maaten and G. Hinton. "Visualizing Data Using t-SNE". *Journal of Machine Learning Research* 9:2579–2605, 2008, p. 85 (cit. on p. 37).

[Vis] VisPy Project Team. *Vispy*. `https://vispy.org`. [Online; accessed 23-February-2022] (cit. on pp. 48, 77).

[VKP00] V. Verma, D. Kao, and A. Pang. "A Flow-Guided Streamline Seeding Strategy". In: *Proceedings Visualization 2000. VIS 2000 (Cat. No.00CH37145)*. 2000, pp. 163–170 (cit. on p. 16).

[War08] M. O. Ward. "Multivariate Data Glyphs: Principles and Practice". In: *Handbook of Data Visualization*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 179–198 (cit. on p. 12).

[WBT97] M. O. Ward, J. T. Blanc, and R. Tipnis. "N-Land: a Graphical Tool for Exploring N-Dimensional Data". In: *Insight Through Computer Graphics: Proceedings of the Computer Graphics International*. 1997, pp. 130–141 (cit. on p. 41).

[Wei+05] T. Weinkauf, H. Theisel, K. Shi, H.-C. Hege, and H.-P. Seidel. "Extracting Higher Order Critical Points and Topological Simplification of 3D Vector Fields". In: *VIS 05. IEEE Visualization, 2005.* 2005, pp. 559–566 (cit. on p. 17).

[Wei07] D. Weiskopf. *GPU Based Interactive Visualization Techniques*. Springer Verlag, 2007 (cit. on p. 26).

[WL93] J. J. van Wijk and R. van Liere. "HyperSlice: Visualization of Scalar Functions of Many Variables". In: *Proceedings of the 4th Conference on Visualization*. 1993, pp. 119–125 (cit. on p. 36).

[XW05] R. Xu and D. Wunsch. "Survey of Clustering Algorithms". *IEEE Transactions on Neural Networks* 16:3, 2005, pp. 645–678 (cit. on p. 37).

Index