New Visualization Techniques for Engineering Simulations

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

angenommen durch die Fakultät für Informatik
der Otto-von-Guericke Universität Magdeburg

von Timo Reinhold Oster, M.Sc.
geb. am 13.06.1987 in Wittlich

Gutachter:
Prof. Dr. Holger Theisel
Prof. Dr. Dominique Thévenin
Prof. Dr. Filip Sadlo

Eingereicht: Magdeburg, den 26. April 2019
Verteidigt: Magdeburg, den 17. Oktober 2019

Abstract

This thesis presents new visualization techniques for engineering simulations in two different disciplines: Turbulent combustion and solid mechanics.

Direct numerical simulations of turbulent combustion are used as a basis to develop and validate higher-level combustion models. A focus of interest in the analysis of such simulations is the flame surface, where most of the chemical reactions take place. The computational power of supercomputers is increasing much faster than the performance of storage infrastructures. This has caused the output and storage of simulation data to become the bottleneck in large-scale simulation runs. We introduce two new techniques for the visualization and analysis of the flame surface in large-scale simulations of turbulent combustion before the background of this storage bottleneck. The first is a space-saving sparse representation for certain types of flames. It allows for the analysis of a larger number of simulation time steps and is the basis for a new flame visualization technique. The second is an algorithm for tracking the flame surface in-situ during the simulation. The storage bottleneck is circumvented by only writing to disk the much smaller results. Both contribute to the continued ability of combustion researchers to analyze the data produced by their increasingly large simulations.

Due to their many degrees of freedom, tensor fields are some of the most challenging types of data to visualize. One possibility to break down their complexity is feature-based visualization, which reduces the data to a set of geometric primitives that represent the occurrence of some kind of interesting behavior. The parallel vectors operator, which yields locations where two vector fields are parallel, is the basis of a number of line-type features in scalar and vector fields. We translate this operator to tensor fields by introducing the parallel eigenvectors operator, which yields locations where two tensor fields have parallel real eigenvectors. We then use this idea to introduce tensor core lines, which mark the centers of "swirling" behavior of the eigenvectors, and are based on vortex core lines in vector fields. Using this new feature, we can detect twist in stress tensor fields from solid mechanics simulations.

Zusammenfassung

Diese Arbeit präsentiert neue Visualisierungsmethoden für Simulationen aus zwei verschiedenen Ingenieurdisziplinen: Turbulente Verbrennung und Festkörpermechanik.

Direkte numerische Simulationen turbulenter Verbrennung sind eine Basis für die Entwicklung und Validierung höherer Verbrennungsmodelle. Ein besonderes Augenmerk bei der Analyse solcher Simulationen liegt auf der Flammenoberfläche, wo der Großteil aller chemischen Reaktionen stattfindet. Die Rechenleistung von Supercomputern wächst inzwischen wesentlich schneller als die Leistung ihrer Speicherinfrastruktur. Infolgedessen ist heute das Speichern der Ausgabedaten der Flaschenhals in großen Simulationen. Wir präsentieren zwei neue Techniken für die Visualisierung und Analyse der Flammenobefläche in großen Simulationen turbulenter Verbrennungsvorgänge vor dem Hintergrund dieses Flaschenhalses. Die erste ist eine platzsparende, ausgedünnte Darstellung für einen bestimmten Typ von Flammen. Diese ermöglicht die Analyse einer größeren Anzahl von Zeitschritten der Simulation und ist die Basis für eine neue Art von Flammenvisualisierung. Die zweite ist ein Algorithmus zur Verfolgung der Flammenoberfläche *in-situ* während der Simulation selbst. Der Flaschenhals des Speichervorganges wird umgangen indem nur die wesentlich kleineren Ergebnisse geschrieben werden. Beide Verfahren tragen dazu bei, dass Verbrennungswissenschaftler auch in Zukunft die Daten analysieren können, die ihre immer größer werdenden Simulationen produzieren.

Tensorfelder gehören wegen ihrer vielen Freiheitsgrade zu den herausforderndsten Daten für die Visualisierung. Eine Möglichkeit, diese Komplexität zu reduzieren ist die Extraktion von Features. Diese reduziert die Daten auf geometrische Primitive, die interessantes Verhalten markieren. Der *parallel vectors operator*, der alle Orte bestimmt an denen zwei Vektorfelder parallel sind, ist die Basis für eine Menge von Linien-Features für Skalar- und Vektorfelder. Wir übertragen diesen Operator auf Tensorfelder und definieren dort den *parallel eigenvectors operator*, der alle Orte bestimmt, an denen zwei Tensorfelder parallele reelle Eigenvektoren haben. Diese Idee nutzen wir anschließend zur Definition von *tensor core lines*, die die Zentren von "wirbelndem" Verhalten der Eigenvektoren markieren und auf Wirbelkernlinien in Vektorfeldern basieren. Mit diesem neuen Feature können wir Verwindungen in Stresstensorfeldern aus Strukturmechaniksimulationen erkennen.

Contents

1

Introduction

Simulations are an integral part of modern engineering. They use a set of models to make predictions about the behavior of a system under given boundary conditions. Compared to experiments, simulations are cheaper and quicker to set up, run, and evaluate, and they often provide access to variables that are hard or impossible to measure in an experiment.

Simulation data – just like data from experiments – is often very large and complex. It is evaluated to validate the simulation and/or to derive new insight. Visualization plays an important role in these tasks. It translates the wealth of data into visual representations that are more easily parsed by humans.

Visualization, and in particular scientific visualization, is a large field of research that has produced numerous techniques to display and analyze the data from engineering simulations and other sources. For many applications, existing simple visualization techniques are sufficient to derive the desired information from the data. However, for more complex problems or technical requirements, more advanced solutions are necessary.

Two active areas of research in visualization are concerned with large data and data of some kind of higher order. Large amounts of data mostly originate

from large or very high-resolution experiments or simulations. Examples for higher-order data are tensor fields, time-dependent vector fields, and ensemble or uncertain data. The contributions of this thesis belong to these two areas of research. We present visualization and analysis techniques for the flame surface in large-scale turbulent combustion simulations, and we introduce novel features that help to understand the complexities of second-order tensor fields using examples from solid mechanics simulations.

1.1 Contributions

The contributions of this thesis are separated into two main parts that at first glance seem to be quite different. The first is concerned with time-dependent scalar and vector fields, the second with static tensor fields. The first deals with very large data sizes while the second mostly handles small- to medium-sized datasets. The first focuses on a concrete application area while the second proposes more general ideas. However, both parts have in common that they propose techniques for the extraction of interesting features from data produced by engineering simulations. In the following, we will give a short summary of our contributions and their motivation.

1.1.1 Analysis and Visualization of the Flame Surface in Turbulent Combustion Simulations

Direct numerical simulations (DNS) are the most accurate tool for the simulation of turbulent combustion. They are used as "numerical experiments" to gain data for development or validation of new combustion models. Because of their high spatial and temporal resolution, DNS are run on large, massively parallel supercomputers and produce huge amounts of data. In recent years, the computing power of supercomputers has increased at a much faster pace than the performance of their storage infrastructure. As a result, the raw data produced by a simulation can not be stored completely in a reasonable amount of time any more. The output of results, rather than their production, has become the bottleneck in large simulations.

A major focus of interest in the modeling of combustion processes is the flame surface. It is the area of the flame where most chemical reactions take place and most heat is produced. We present two approaches for the visualization and analysis of different aspects of the flame surface in large-scale DNS of turbulent combustion, where raw data storage has become the bottleneck.

The first is a space-saving sparse representation for premixed flames. Due to its smaller size, it allows storing and therefore analyzing a larger number of

simulation time steps. This representation can directly be used for a statistical analysis of the flame structure, and it is the basis for a new visualization technique that highlights local differences of the flame structure in relation to the global shape of the flame. If required, the full data can be reconstructed on the original grid with some loss of accuracy for the full flexibility of conventional post-processing.

The second approach we present is an algorithm for tracking the flame surface in-situ during the simulation. This circumvents the storage bottleneck by processing the data while it is still in memory and only writing to disk the much smaller results. We propose a massively parallel algorithm using independent micro-patches that refine and coarsen independently. This allows tracking the surface over long time intervals and provides data about the history of individual points attached to the surface as well as their relative movement. Using this data, which is impossible to obtain via traditional post-processing for large simulation runs, combustion researchers can derive new combustion models, especially incorporating unsteady phenomena.

1.1.2 Line Features in 3D Second-Order Tensor Fields

Tensor fields occur in different scientific disciplines. One important example are stress tensor fields, which are often the result of solid mechanics simulations. They describe the state and distribution of stresses in a solid object.

Tensor fields have more degrees of freedom than vector fields and are therefore more challenging to visualize. Almost all techniques for tensor field visualization have in common that they represent tensors using their eigenvectors and eigenvalues. Visualization techniques for vector fields can sometimes be applied to the eigenvectors of a tensor field with some modifications. Just as they help make sense of the structure of vector fields, they can make sense of the structure of eigenvectors in tensor fields.

We take first steps towards translating a class of features from vector- to tensor fields that has not been considered to date: line features obtained by the *parallel vectors* (PV) operator. This operator yields all locations where two vector fields are parallel. It can be used to compute ridge- and valley lines in scalar fields as well as separation-, attachment-, and vortex core lines in vector fields. We establish the *parallel eigenvectors* (PEV) operator as the equivalent of the PV operator for vector fields. We then use it to translate the concept of vortex core lines to tensor fields by defining *tensor core lines* that mark the centers of "swirling" behavior of the eigenvectors. Using these new features, we demonstrate how to find locations of aligned principal stress directions in objects under two different loads, and how to find areas of twist in an object under stress.

1.2 Thesis Structure

This thesis is separated into three main parts. Part I provides background information that is relevant to understand the context of this work.

- Chapter 2 gives an overview of the field of scientific visualization.
- Chapter 3 introduces the field of turbulent combustion and its visualization, which is relevant to the second part of this thesis.

Part II presents the contributions in visualization of the flame surface in DNS of turbulent combustion.

- Chapter 4 presents a sparse representation for premixed flames that saves storage space and is the basis for a new flame visualization technique.
- Chapter 5 introduces an in-situ algorithm for tracking the flame surface during a massively parallel simulation run.
- Chapter 6 concludes this part of the thesis.

Part III presents new line features for second-order tensor fields.

- Chapter 7 introduces the PEV operator, which yields all locations where two tensor fields have parallel real eigenvectors.
- Chapter 8 applies the PEV operator – with some modifications – to translate the concept of vortex core lines to the eigenvectors of a tensor field.
- Chapter 9 concludes the last part of the thesis.

1.3 List of Publications

The following articles have been published in peer-reviewed journals and conferences as results of this thesis:

T. Oster, D. J. Lehmann, G. Fru, H. Theisel, and D. Thévenin
Sparse Representation and Visualization for Direct Numerical Simulation Of Premixed Combustion
Computer Graphics Forum 33.3, pp. 321–330, 2014

A. Abdelsamie, G. Fru, T. Oster, F. Dietzsch, G. Janiga, and D. Thévenin
Towards Direct Numerical Simulations of Low-Mach Number Turbulent Reacting and Two-Phase Flows Using Immersed Boundaries
Computers & Fluids 131, pp. 123–141, 2016

C. Chi, A. Abdelsamie, T. Oster, and D. Thévenin
Probability of Hotspot Ignition and Ignition Spot Tracking in Turbulent Hydrogen-Air Mixtures Using Direct Numerical Simulations
8th European Combustion Meeting, pp. 925–930, 2017

T. Oster, A. Abdelsamie, M. Motejat, T. Gerrits, C. Rössl, D. Thévenin, and H. Theisel
On-The-Fly Tracking of Flame Surfaces for the Visual Analysis of Combustion Processes
Computer Graphics Forum 37.6, pp. 358–369, 2018

T. Oster, C. Rössl, and H. Theisel
Core Lines in 3D Second-Order Tensor Fields
Computer Graphics Forum 37.3, pp. 327–337, 2018

T. Oster, C. Rössl, and H. Theisel
The Parallel Eigenvectors Operator
Vision, Modeling and Visualization, 2018

1.4 Notation

We will use the following mathematical notation throughout the thesis. Additional notation is introduced whenever it is needed.

s, a, λ	Scalars
$\mathbf{v}, \mathbf{r}, \mathbf{x}$	Column vectors
v_i, r_i, x_i	Components of the respective vectors
$\mathbf{0}$	The vector of all zeros
$\mathbf{A}, \mathbf{S}, \Sigma$	Matrices
$\mathbf{I}$	The identity matrix
$\mathbf{A}^\mathrm{T}, \mathbf{v}^\mathrm{T}$	Transpose of a matrix/vector
$\begin{pmatrix} \mathbf{a} & \mathbf{b} & \mathbf{c} \end{pmatrix}$	Block matrix composed of several matrices/vectors
$\mathbf{v} \cdot \mathbf{w}$	Inner (dot-) product of two vectors
$\mathbf{v} \times \mathbf{w}$	Cross product of two vectors
∇	Nabla-Operator $\left(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n}\right)^\mathrm{T}$

I

Background

2

An Overview of Scientific Visualization

Scientific visualization is the visualization of data from the natural sciences. Although such data can come in a variety of forms, the term is most often used to describe the techniques used for displaying spatial, possibly time-varying data from physics, chemistry, engineering, or biomedical applications. Examples for such data are volumetric scans of patients from medicine, wind speed and pressure measurements from meteorology, and simulations of the air flow around a car.

This thesis presents several new scientific visualization techniques. The techniques are applied to simulation data from two different engineering domains: turbulent combustion and solid mechanics. To set the scene for these contributions, we will therefore use this chapter to give an overview of the most important methods for visualizing scientific data. Such data usually comes in the form of scalar-, vector-, or (second-order) tensor fields. We will take a look at the definition and basic visualization techniques for each of these types of fields in the following sections.

This chapter is necessarily sparse on details and omits a lot of basic knowledge on computer graphics, data representation, numerical algorithms, and visualization in general. A more thorough introduction to the field is given in

Alexandru Telea's book "Data Visualization" [1]. Material for further reading can be found in the references cited throughout the text.

2.1 Scalar Field Visualization

A scalar field is a map $s(\mathbf{x}, t) : D \times T \mapsto \mathbb{R}$ that assigns a scalar value to each position $\mathbf{x}$ and time t in spatial and temporal domains D and T. The spatial domain D is a subset of the (two- or three-dimensional) Euclidean space $\mathbb{E}^n$. The temporal domain T is usually an interval of $\mathbb{R}$. Examples of scalar fields are the temperature in a solid object, the population density on a map, and the attenuation coefficient in a computed tomography (CT) scan. If the scalar field does not change with time, or we are only interested in a single instant, we often just write $s(\mathbf{x})$ and omit the time parameter.

Methods for visualizing scalar fields can be roughly separated into two groups: *image-based* and *geometry-based*. We will briefly cover the most important methods in the following.

2.1.1 Image-Based Methods

Image-based methods map the scalar at each position in space to a property and display it directly. Among such methods are *color-mapping* and *height-mapping* for 2D scalar fields as well as *direct volume rendering* for 3D scalar fields.

Color-Mapping

Color-mapping means assigning each scalar value a color from a predefined color map and displaying the result as an image or texture. Due to its simplicity, it is probably the most widespread method presented here. This technique is only directly applicable to 2D scalar fields, but is commonly used on 3D data by selecting slices or surfaces in a volume, or by displaying the scalar value on the outside surface of the volume.

Height-Mapping

Height-mapping only works on 2D data. It essentially means interpreting the scalar value at each point as a height value and displaying the resulting three-dimensional surface. Because it transforms 2D information into 3D information, it is best suited for interactive settings, where the resulting surface can be rotated and viewed from all directions.

Figure 2.1: Direct volume rendering of a CT scan of a human skull using volumetric raycasting with gradient-based shading. Image source: Wikimedia Commons.

Direct Volume Rendering

Direct volume rendering [2, 3] is the extension of color-mapping to 3D scalar fields. It involves two steps: Applying a transfer function, and accumulating the resulting colors and opacities along viewing rays to produce an image. The transfer function has to be chosen carefully to reveal the structures in the data that are interesting, and make the uninteresting parts transparent. Color and opacity values are then accumulated along viewing rays to simulate the transport of light through a semitransparent medium.

The shading of a solid surface can be approximated by using the gradient of the scalar field as the surface normal (see Figure 2.1). More advanced techniques use a two-dimensional transfer function that also takes the gradient magnitude into account when deciding the color, opacity and shading parameters of a point along the viewing ray [4].

2.1.2 Geometry-Based Methods

Geometry-based methods extract some geometrical structures from the data and display these structures. *Isocontours and -surfaces* belong to this category together with topological features such as *extremal- and saddle points*, as well as *ridge- and valley lines and -surfaces*.

Isocontours and -surfaces

Isocontours and -surfaces are manifolds in the scalar field where the scalar is equal to a constant (iso-) value. In the literature these are also often referred to as *level sets*. They form lines or surfaces that are always closed or end at the domain boundary, and that never intersect each other. For 2D scalar fields, it is common to plot contours for several isovalues at once, which resemble the height lines we know from maps. In 3D, it is rare to display more than two or three different isosurfaces at the same time due to occlusion problems.

Critical Points

Critical points are points in a scalar field where the gradient becomes zero. As such, they are features of the scalar field's topology. Critical points can be classified by the eigenvalues of the Hessian matrix at the critical point: If all eigenvalues are positive the point is a minimum, if all are negative it is a maximum, and if the Hessian has both positive and negative eigenvalues, the point is a saddle.

Ridge- and Valley Lines and -Surfaces

Ridge- and valley lines and -surfaces also belong to the category of topological features. There is no universal agreed-upon definition for these types of features. The two most common, competing definitions are *watersheds* and *height ridges* [5, 6]. Watersheds are global features that separate the scalar field into "areas of influence" of the different minima (or maxima). Imagine a scalar field as a height field. Starting a gradient descent from two points on the same side of a watershed will end up in the same minimum. Starting from two points on opposite sides of the watershed will end up in two different minima.

Height ridges are defined by a local differential analysis of the scalar field. As such, their computation is less costly, but they do not provide a space-filling segmentation of the data. Often, computation of raw ridges and valleys produces a lot of small and insignificant features due to noise. An additional filtering step [5] can be applied to only keep the most significant lines (see Figure 2.2).

Morse-Smale Decomposition

The Morse-Smale decomposition forms the topological skeleton of a scalar field. It is based on the idea of integral lines of the scalar field. These are maximal paths that are everywhere tangent to the gradient of the scalar field

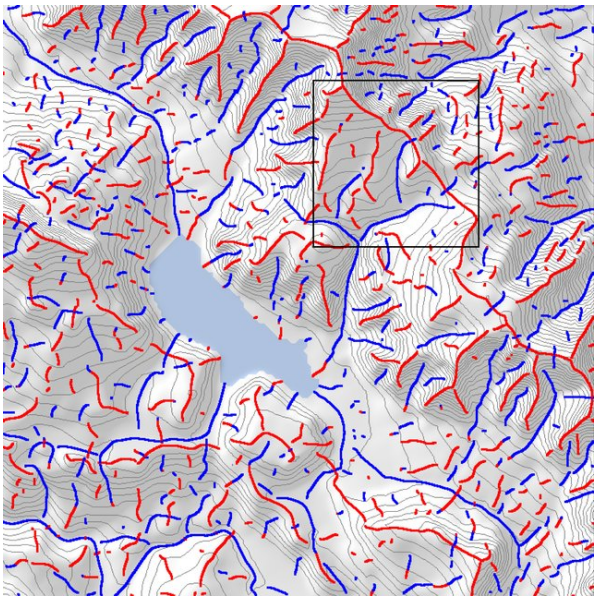

Figure 2.2: Filtered ridge and valley lines of a scalar field. Ridges are shown in red, valleys in blue. The scalar field is visualized using a combination of height mapping (to obtain the shading) and isocontours. Image source: Peikert and Sadlo [5].

(see also integral curves in vector fields, Section 2.2.3). Integral lines always connect two critical points or end at the domain boundary.

The Morse-Smale decomposition separates the scalar field into a disjoint set of cells whose union is the whole domain. Each cell is defined as the set of all integral lines that connect the same minimum and maximum. In this sense, the Morse-Smale decomposition is closely related to the idea of watersheds: The boundaries of the cells are precisely the watersheds of the scalar field and its negative.

2.2 Vector Field Visualization

A vector field $\mathbf{v}(\mathbf{x}, t) : D \times T \mapsto \mathbb{R}^n$ is a map from spatial and temporal domains $D \subset \mathbb{E}^n$ and $T \subset \mathbb{R}$ to the n-dimensional vector space $\mathbb{R}^n$. Just like a scalar field, it assigns a value to each position and time in the domains, but here the value is a vector. Examples for vector fields are the velocity of a fluid flow, the displacement field of a deformed object, and the magnetic field around an electromagnet. If the vector field does not change with time, or we are only interested in the field at a single instant, we say that we have a *steady* vector field $\mathbf{v}(\mathbf{x}, t_c)$, where t_c is constant. If the vector field changes with time, we call it an *unsteady* vector field. If we are talking about the velocity of a fluid flow, we sometimes call it a (steady or unsteady) *flow field*.

Vector field visualization methods can be sorted into roughly six different classes: *Basic methods*, *image-based methods*, *integral curves and -surfaces*, *topo-*

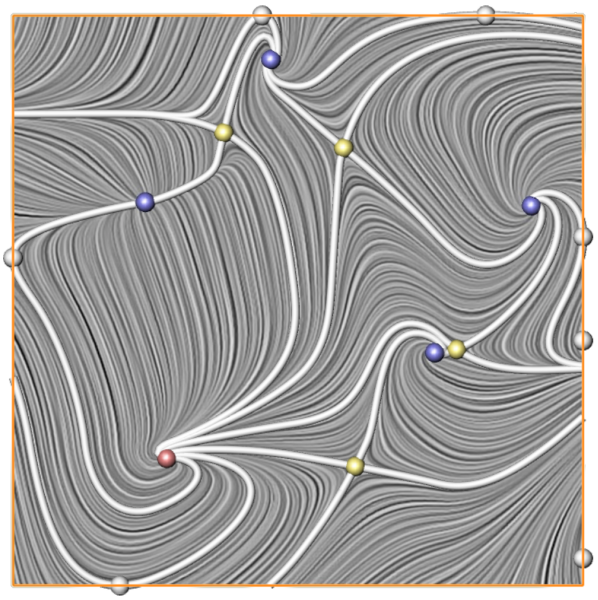

Figure 2.3: line integral convolution (LIC) of a 2D vector field with overlaid vector field topology consisting of critical points, boundary switch points, and separatrices. Image source: Tino Weinkauf [7].

logical features, *vortex extraction* and *Lagrangian coherent structures*. We will visit the most important methods in the following sections.

2.2.1 Basic Methods

Basic methods display the vector data using simple techniques, much like image-based methods for scalar fields. They encompass techniques such as *arrow plots*, or *color-mapping* the velocity magnitude, vorticity and other derived quantities directly.

Arrow Plots

Arrow plots are the simplest way to visualize a vector field. In such a plot, arrow glyphs are placed at multiple locations throughout the domain. The arrows are aligned with the direction of the local vector, and their length is typically scaled based on its magnitude. Such plots can very accurately show the vectors at a limited number of locations, but they quickly become cluttered once too many arrows are plotted, or the arrows become too long and occlude each other. If the magnitudes in a vector field range over many different scales, finding an appropriate scaling factor for arrow length can be challenging. In such cases, it is often better to plot normalized arrows and represent the magnitude only via color.

Color-Mapping

Color-mapping can be applied to vector fields in different ways. The most common one is simply displaying the magnitude of the vector field as a scalar. Other scalars derived from the vector field, such as the vorticity magnitude and divergence, can be visualized in the same way. All of this obviously goes along with a loss of information. Since color has three degrees of freedom, a vector field can also be visualized without information loss by directly mapping the vector values to colors. The most naive way is to simply interpret the three components of a 3D vector as RGB values. Such images theoretically contain the full information of the original vector field. However, they are very hard to interpret, as there is no inherent meaningful connection between the direction of the vector and the color it is mapped to. This can be slightly improved for 2D vector fields by mapping the angle and magnitude of the vector to hue and value of the HSV color space.

2.2.2 Image-Based Methods

Image-based methods visualize the vector data by generating a space-filling texture. Among such methods are *line integral convolution*, *spot noise*, and *texture advection*.

Line Integral Convolution

Line integral convolution (LIC)[8] is the most popular image-based vector field visualization technique. It is based on "smearing" a random noise texture along stream lines of a 2D vector field. More specifically, the color at a certain position is determined as a weighted integral of the color values encountered along a stream line passing through that position. The result is a space-filling image where the direction of the vector field is visible at each location (see Figure 2.3). The basic LIC technique is only applicable to 2D steady vector fields, but extensions have been developed for 3D [9] and unsteady [10] datasets.

Spot Noise

Spot noise is also a technique designed for 2D data and produces results similar to LIC [11, 12]. It works by blending noise sprites that have been stretched and rotated according to the local vector magnitude and direction. In contrast to LIC, spot noise better represents the local vector magnitude, but in regions with low magnitude, the vector direction is not well visible.

Texture Advection

Texture advection works on 2D steady and unsteady vector fields. It is best suited for flow fields, as it simulates the transport (advection) of a texture. The texture is initialized at some point in time, and each point of the texture then moves with the flow. As time progresses, the texture is warped, and the viewer can follow where each part of the texture is transported. This method has a lot in common with the integration-based methods presented in the next section. In fact, texture advection simply displays a time surface with a mapped texture in a 2D vector field.

2.2.3 Integral Curves and -Surfaces

Integral curves and -surfaces are generated by integrating the vector field starting from different kinds of seed structures. Depending on the seeding strategy and the kind of vector field, we can obtain *streamlines*, *pathlines*, *streaklines*, *timelines*, and the accompanying surfaces. Since integral structures play an important role in several parts of this thesis, we will cover them here in more detail.

Streamlines

Streamlines are the simplest form of integral curve in a vector field. They are defined for steady vector fields. Depending on the application area, they are also sometimes called *field lines*. A streamline is a curve that is tangent to the vector field everywhere along its path. Given a streamline $\mathbf{c}(s)$ of the steady vector field $\mathbf{v}(\mathbf{x}, t_c)$, this means that $\dot{\mathbf{c}}(s) \times \mathbf{v}(\mathbf{c}(s), t_c)$ is $\mathbf{0}$ for all s. This criterion is valid for any parameterization of the curve, but we can only use it to check if a given curve is a streamline. To compute streamlines, we typically solve the ordinary differential equation

$$\frac{\partial \mathbf{c}(s)}{\partial s} = \mathbf{v}(\mathbf{c}(s), t_c), \quad \text{with} \quad \mathbf{c}(0) = \mathbf{x}_0 .$$

This yields a streamline with a particular parameterization: an integral curve of the vector field. Due to their definition, two stream lines never intersect at single points. They are either completely disjoint, or they coincide.

Visualizing a steady vector field with stream lines shows the direction of the flow much like a LIC image does, but not in a space-filling manner. This allows for use of streamlines also in 3D data (see Figure 2.4). Streamline visualization can be deceiving when used on single time slices of an unsteady flow field. The connected lines mistakenly suggest paths of fluid elements. However, if the flow field changes with time, the actual paths of fluid elements can deviate significantly from the streamlines of a single time slice. In this case, the more

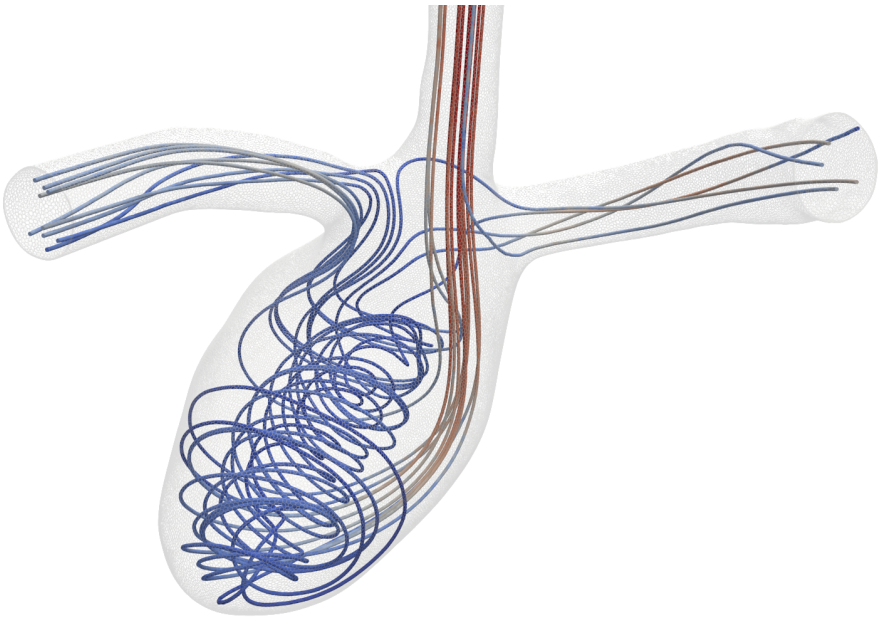

Figure 2.4: Streamlines in a simulated flow through a cranial aneurysm. Image courtesy of Tim Gerrits [13].

appropriate visualization tools are pathlines, streaklines, and timelines, which incorporate the temporal information of the flow.

Pathlines

Pathlines describe the paths of massless particles moving with a flow. They are defined as the solution to the ordinary differential equation

$$\frac{\partial \mathbf{c}(t)}{\partial t} = \mathbf{v}(\mathbf{c}(t), t), \quad \text{with} \quad \mathbf{c}(t_0) = \mathbf{x}_0\,,$$

where $\mathbf{c}(t)$ is the curve of the pathline, t is time, and $\mathbf{v}(\mathbf{x}, t)$ is an unsteady vector field. Looking at this definition, it becomes apparent that for a steady vector field, which does not change with time, streamlines and pathlines are identical.

The set of all pathlines for all possible combinations of start position $\mathbf{x}$, start time t_0 and end time t_e forms the *flow map*. This function, which we write as $\mathbf{\Phi}(\mathbf{x}, t_0, t_e)$, determines where a massless particle starting at position $\mathbf{x}$ and time t_0 ends up after advecting with the flow until time t_e.

Pathlines allow the visualization of the dynamic behavior of an unsteady flow in a static image. To show the temporal information, the time is often color-mapped on the curve. Unlike streamlines, pathlines can and do intersect

each other. For very complex flows, showing a lot of pathlines can therefore quickly become confusing. In such cases it can be better to show animated streaklines instead.

Streaklines

Streaklines are the connected locations of a continuously injected stream of massless particles into a flow. They approximate the behavior of a thin stream of dye injected at a certain position that is often applied in experimental settings to visualize the flow. Formally, a streakline is formed by the connected endpoints of a set of pathlines with the same start position and end time, but continuously increasing start time. Using the flow map $\mathbf{\Phi}$, which we introduced earlier, we can formally define a streakline as

$$\mathbf{c}(s) = \mathbf{\Phi}(\mathbf{x}, s, t)\,,$$

where t is the current time, $\mathbf{x}$ is the injection point, and s is the continuously increasing start time that runs along the curve. Like pathlines, streaklines also become identical to streamlines if the flow is steady.

While a pathline shows the behavior of the flow over a period of time, a streakline only ever shows the position of the injected particles at a single time instant. Streaklines are therefore often animated by continuously increasing the end time t and injecting more particles. This again mirrors the behavior of injected dye observed in an experiment.

Timelines

Timelines are different from all the previous integral lines in that their seeding structure is not a single point, but a whole line. A timeline is formed by placing a line somewhere in the flow, treating it as a set of massless particles, and letting the whole line advect with the flow at once. Formally, a timeline is formed by the connected endpoints of a set of pathlines with the same start and end time, but continuously changing start position. Given a seed curve $\mathbf{s}(s)$, start time t_0 and current time t, we can formally define a timeline in terms of the flow map as

$$\mathbf{c}(s) = \mathbf{\Phi}(\mathbf{s}(s), t_0, t)\,.$$

Much like streaklines, timelines are often animated to show the progressive effect of the flow. As the time t advances, the line is transported and warped by the flow and visualizes the way the flow mixes and perturbs a region of the fluid.

Integral Surfaces

Integral surfaces can be formed from any of the integral lines by using a higher-dimensional seeding structure. For stream-, path-, and streaklines this means using a line as the seeding structure. Timesurfaces are formed by using a surface as the seed. Integral surfaces can be helpful for visualization because they have a better visual coherency than a number of single lines. The curvature, wrinkling and folding of structures induced by the flow become easier to grasp when using integral surfaces. On the flip side, integral surfaces have more of a problem with occlusion compared to lines, as they are more massive.

2.2.4 Vector Field Topology

The topology of a vector field is defined by *critical points*, *separation- and attachment points* and the accompanying *separatrices* connecting them. Together, they form a sort of skeleton of the vector field, from which the behavior of the full field can be inferred.

Critical Points

Critical points of a vector field are locations where the magnitude of the vector becomes zero. They are interesting because they are at the centers of interesting structures in the vector field. Critical points in vector fields can be sorted into different categories. Which category a critical point belongs to depends on the behavior of the vector field in its vicinity, which is encoded in its derivative. The Jacobian matrix $\mathbf{J}(\mathbf{v}) = \nabla\mathbf{v}$ (or, more correctly, $\mathbf{v}\nabla^{\mathrm{T}}$) gathers the partial derivatives of all components of the vector field. The signs of the real parts of its eigenvalues indicate if the vector field is attracting or repelling in the vicinity of the critical point. Depending on these signs, the critical point can be categorized as a *sink* (negative), *source* (positive), or *saddle* (mixed signs). The presence of an imaginary part of the eigenvalues/eigenvectors indicates swirling behavior. A more in-depth discussion of critical points in vector fields is provided by Helman and Hesselink [14].

Separation- and Attachment Lines/Surfaces

Separation- and attachment lines/surfaces occur on no-slip boundaries. They are locations where the flow separates from/attaches to a surface. In a 2D flow, these are isolated points. In 3D, they can be point or line structures. Separation- and attachment structures are similar to critical points of the vector field, specifically to saddles, in that they are end points of streamlines

of the vector field. However, they are not exactly critical points, as the velocity is zero everywhere on a no-slip boundary. Instead, they are topological features of the skin friction field, which describes the shear of the flow near the boundary [15].

Separatrices

Separatrices connect critical points with each other. They are lines in 2D vector fields and can be lines or surfaces in 3D vector fields [14, 16]. Separatrices start at saddle points or separation-/attachment points. They are streamlines with a special property: Streamlines passing through two points on opposite sides of the separatrix will diverge from each other near a saddle or separation-/attachment point in forward or backward flow direction. As such, they separate the flow into distinct regions that streamlines starting in these regions will never leave. Because streamlines and separatrices are an instantaneous observation (they exist in steady vector fields or at single points in time in an unsteady vector field), this does not imply that pathlines will never leave these regions in unsteady flow. For unsteady flows, the equivalent of separatrices are Lagrangian coherent structures, which we will cover in Section 2.2.6.

2.2.5 Vortex Extraction

Vortices are structures in a flow that show a swirling motion around a common center. They will usually stay intact for long periods of time. Vortices are the defining characteristic of turbulent flow. The study of their behavior is a central topic of current fluid dynamics research. Even though the concept of a vortex seems rather simple, hundreds of years of fluid dynamics research still has not resulted in a universally accepted formal definition. This is why there is a plethora of methods for detecting and quantifying vortices in the literature. Most methods can be sorted into two categories: *region-based* and *line-based*. They can be further classified by their invariance to transformations of the reference frame through which the flow is observed. Non-invariant methods are sensitive to all reference frame transformations. Galilean invariant methods are insensitive to any motion of the reference frame with constant speed and direction. Objective methods are insensitive to any smooth translation and rotation of the reference frame. We will cover some significant vortex extraction methods here. An extensive survey on the subject has been done by Günther and Theisel [17].

Region-based methods measure the "vortex-ness" of the flow at a point by a scalar quantity. Applying a threshold to this quantity shows the vortex regions. Notable representatives of this category are the *vorticity magnitude*, λ_2*-criterion* and *Q-criterion*, which are all Galilean invariant methods that work on steady

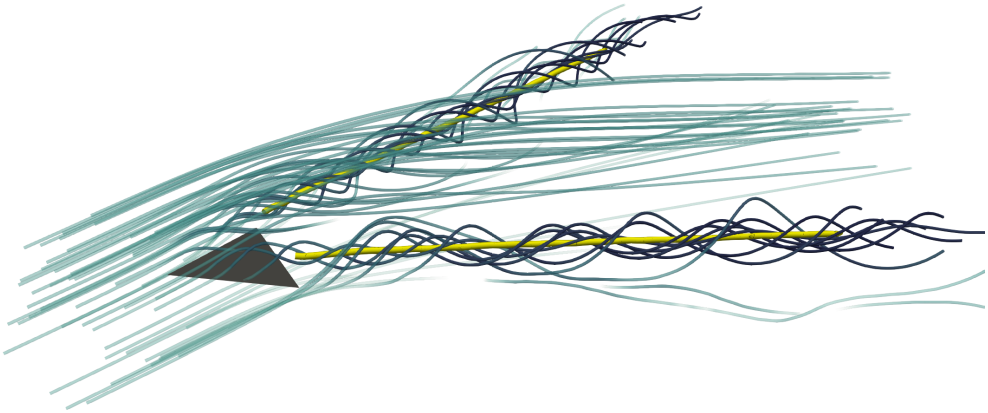

Figure 2.5: Vortex core lines (yellow) and streamlines (blue/green) in the steady flow around a delta wing.

vector fields. Recently, the *instantaneous vorticity deviation* (IVD) has been proposed as an objective criterion for steady vector fields. Its extension, the *Lagrangian-averaged vorticity deviation* (LAVD) also accounts for unsteady behavior of the flow. Region-based approaches are generally simple and efficient to implement, but the results are dependent on the choice of the threshold, and they do not produce explicit representations of the vortices.

Line-based methods explicitly extract the vortex core line that is the center of the swirling behavior. The *reduced velocity* approach for extracting vortex core lines in steady flows was proposed by Sujudi and Haimes [18] and later identified as an application of the PV operator by Peikert and Roth [19]. This approach is only Galilean invariant when applying it to 2D vector fields. A Galilean invariant approach for finding the *cores of swirling particle motion* in unsteady flows was proposed by Weinkauf et al. [20]. Recently, Günther et al. [21] provided a framework for objective vortex core detection. This is realized by determining a locally *near-steady frame* for observing the flow.

Vorticity Magnitude

The vorticity is commonly used in fluid dynamics literature to characterize the rotational behavior of the flow. It is defined as the curl of the flow field $\nabla \times \mathbf{v}$. The result is a vector field that points in the direction of the local axis of rotation and whose magnitude indicates the rotation strength. Vorticity magnitude is sometimes used to detect vortices. However, a constant threshold over the whole domain is often not sufficient to detect and distinguish all vortices, and it might yield false-positives in shear flow. This is why other methods often perform better.

Q-Criterion

The Jacobian $\mathbf{J}$ of a flow field can be decomposed into a symmetric part $\mathbf{S}$ (often called *strain rate tensor*) and an asymmetric part $\mathbf{\Omega}$ (often called *vorticity tensor*) where

$$\mathbf{S} = \frac{\mathbf{J} + \mathbf{J}^T}{2}, \quad \mathbf{\Omega} = \frac{\mathbf{J} - \mathbf{J}^T}{2}.$$

For divergence-free (i.e., incompressible) flows, the Q-criterion states that a region belongs to a vortex if the vorticity tensor is stronger than the strain rate tensor, i.e.,

$$Q = (\|\mathbf{\Omega}\|^2 - \|\mathbf{S}\|^2) > 0\,.$$

The resulting regions are not as sensitive to the scaling of the data as the vorticity magnitude.

λ_2-Criterion

The λ_2-criterion [22] uses invariants of the Jacobian to detect pressure valleys in incompressible flows. It identifies vortex core regions by investigating the eigenvalues of the tensor $\mathbf{S}^2 + \mathbf{\Omega}^2$. If the local tensor has at least two negative eigenvalues (i.e., if the middle eigenvalue λ_2 is smaller than zero), a point is considered to belong to a vortex core region. In most cases, λ_2-criterion and Q-criterion yield similar results.

IVD/LAVD

Recently, the instantaneous vorticity deviation (IVD) was proposed by Haller et al. [23] as an objective vortex measure for steady flow fields. It is based on the observation that while the vorticity itself is not objective, the difference between two vorticity vectors is. Based on this, they define the IVD as the difference of the local vorticity to the average vorticity in a local neighborhood. The Lagrangian-averaged vorticity deviation (LAVD) extends this to unsteady flows by integrating the IVD along a pathline over a certain time interval.

Reduced Velocity/Parallel Vectors

The first line-based method we present here was proposed by Sujudi and Haimes [18] and works on steady flow fields. It is based on the observation that vortex centers look like critical points when looking at a slice of the vector field that is orthogonal to the vortex core line. In regions with swirling flow, the Jacobian has two complex conjugate and one real eigenvector, which points along the local axis of rotation. The reduced velocity or Sujudi/Haimes criterion therefore states that a vortex core line is located where the projection

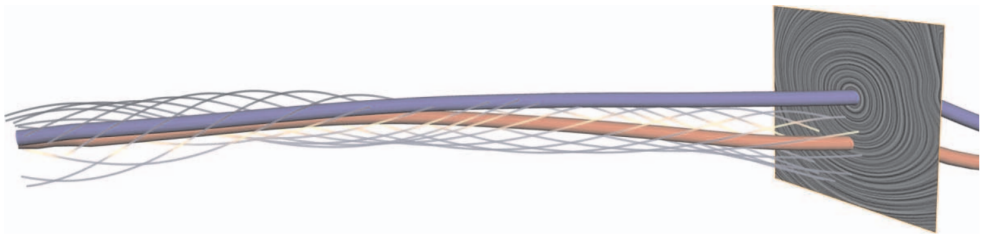

Figure 2.6: Streamlines and pathlines swirl around different cores. The red tube shows the core of pathlines of a 2D vector field (rendered in 3D spacetime). The blue tube shows the path of the vortex core extracted from the individual time slices. Source: Weinkauf et al. [20].

of the local velocity onto a plane orthogonal to the single real eigenvector is zero. Peikert and Roth [19] later showed that this is equivalent to locations where the velocity vector $\mathbf{v}$ is parallel to its acceleration $\mathbf{Jv}$, and so is an application of the PV operator, which does not require the explicit computation of eigenvectors. Defining the criterion in this way, it is equivalent to finding locations where stream lines have locally vanishing curvature.

Cores of Swirling Particle Motion

The method of Sujudi/Haimes only considers steady flow fields, or instantaneous snapshots of unsteady flows. This means that this method finds the centers of swirling streamlines. In unsteady flows, streamlines and pathlines appear to swirl around different cores if the vortex moves over time (see Figure 2.6). Weinkauf et al. [20] therefore developed an approach to find the cores of swirling pathlines in unsteady flows. They express pathlines as streamlines in a flow field with one more dimension where time has been included as an additional explicit state variable. For 3D unsteady flows, the pathlines are streamlines in 4D space-time. They derive a criterion similar to Sujudi/Haimes for 4D vector fields that can be reduced to a parallel vectors operation on two derived 3D vector fields.

Near-Steady Frame

The optimal reference frame for observing a vortex is a frame that follows the vortex center over time [24]. In such a frame, the observed vector field around the vortex becomes almost steady as ambient effects of larger flow structures are eliminated. This is the basis of the approach proposed by Günther et al. [21]. For each point in the flow, a locally optimal reference frame is computed in which the flow becomes almost steady. The reference frame is determined for a finite-sized neighborhood via a simple linear optimization.

Once the vector field (and its derivatives) have been "objectified", any region- or line-based vortex extractor that is designed for steady vector fields can be used to extract objective unsteady vortices.

2.2.6 Lagrangian Coherent Structures

Lagrangian coherent structures (LCS) are structures that show a locally maximal attracting or repelling behavior of massless particles. As the name suggests, these structures also tend to stay coherent over longer time periods. They can be thought of as a counterpart to vortices. Whereas vortices represent the centers of swirling behavior, LCS tend to be located at the boundaries between vortices. They act as transport barriers that have a minimal transverse flux of material and are therefore very important for the investigation of mixing and transport processes. There are a number of very different methods for the investigation of LCS. A survey and comparison of some notable methods can be found in [25].

One popular approach for detecting LCS is the investigation of the local Lyapunov exponent [26]. The Lyapunov exponent describes the rate of separation of infinitesimally close trajectories over an infinite time interval. Since real-world data is usually finite in time and space, two approximations of the Lyapunov exponent are commonly used in practice: the *finite-time Lyapunov exponent* (FTLE) [27] and the *finite-size Lyapunov exponent* (FSLE) [28]. It has been shown that ridges of the FTLE or FSLE field correspond well to the locations of LCS [27, 29].

The basis for the computation of both FTLE and FSLE is the right Cauchy-Green deformation tensor

$$\mathbf{C}(\mathbf{x}, t_0, t_e) = \mathbf{F}(\mathbf{x}, t_0, t_e)^{\mathrm{T}} \mathbf{F}(\mathbf{x}, t_0, t_e),$$

where $\mathbf{F}$ is the deformation gradient obtained from the derivative of the flow map

$$\mathbf{F}(\mathbf{x}, t_0, t_e) = \nabla \mathbf{\Phi}(\mathbf{x}, t_0, t_e).$$

The Cauchy-Green tensor describes the deformation the infinitesimal neighborhood of a point $\mathbf{x}$ at time t_0 experiences when advecting with the flow until time t_e. The maximum eigenvalue $\lambda_{\max}$ of this tensor is a measure for the maximum separation of two particles starting in this infinitesimal neighborhood.

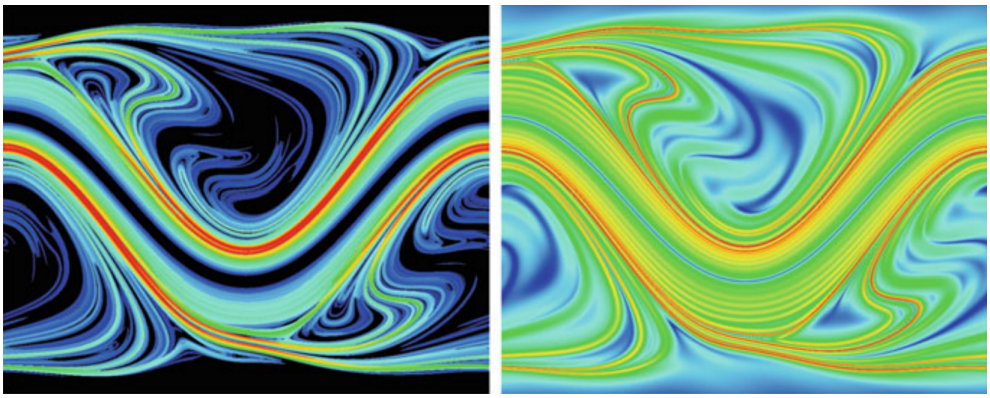

Figure 2.7: Comparison of FSLE (left) and FTLE (right) of the flow field of a meandering jet. Image source: Peikert et al. [31].

Finite-Time Lyapunov Exponent

The finite-time Lyapunov exponent measures the maximum separation of neighboring particles after advecting with the flow for a finite time:

$$\mathrm{FTLE}(\mathbf{x}, t_0, \tau) = \frac{1}{|\tau|} \ln \sqrt{\lambda_{\max}\left[\mathbf{C}(\mathbf{x}, t_0, t_0 + \tau)\right]}.$$

A straightforward way of approximating the FTLE of a flow is to compute the flow map $\mathbf{\Phi}$ on a discrete grid and estimate the deformation gradient $\mathbf{F}$ via central differences. Since the ridges of an FTLE field can become very sharp with increasing integration time τ, this is usually not very accurate. Additionally, we are usually not interested in an accurate estimation of regions without ridges. For this reason, there are multiple numerical schemes for more precise or efficient FTLE computations. A good overview of the existing methods is provided in Alexander Kuhn's PhD thesis [30].

Finite-Size Lyapunov Exponent

The finite-size Lyapunov exponent measures the time a pair of infinitesimally close particles need to separate by a constant amount in space:

$$\mathrm{FSLE}(\mathbf{x}, t_0, r) = \frac{1}{|\tau_r|} \ln r\,,$$

where τ_r is the minimum time interval for which

$$\sqrt{\lambda_{\max}\left[\mathbf{C}(\mathbf{x}, t_0, t_0 + \tau_r)\right]} = r\,.$$

FSLE is popular in the oceanography community, but has not been adopted much in the general visualization community. As a result, FSLE computation

schemes in the literature are mostly straightforward. Either the maximum separation $\lambda_{\max}$ is estimated by a discrete sampling of the flow map in different directions [32, 33], or the Cauchy-Green tensor is estimated based on central differences [31].

FTLE and FSLE yield similar results, given the right parameters (see Figure 2.7). However, one or the other might be more appropriate depending on the application. While FTLE operates on a finite integration time τ, which must be estimated a-priori, it shows the behavior of the flow for all scales of spatial separation. In contrast, FSLE needs the choice of a separation size, which might be more intuitive. It yields information about separation at different spatial scales, but it does not show separation that is smaller than the given threshold r.

2.3 Second-Order Tensor Field Visualization

Although tensors are a very general concept in mathematics, when we talk about tensor fields in scientific visualization, we generally mean a map $\mathbf{T}(\mathbf{x}, t) : D \times T \mapsto \mathbb{R}^{m \times m}$ from spatial domain $D \in \mathbb{E}^n$ and temporal domain $T \in \mathbb{R}$ to the space of second-order tensors, which can be represented by matrices from $\mathbb{R}^{m \times m}$. While a vector describes an effect acting on a point, a second-order tensor describes an effect on a vector. Often, this means that the tensor describes some sort of differential effect that acts on the infinitesimal neighborhood of a point. Second-order tensor fields occur in a variety of different scientific contexts. Some examples are stress and strain tensors in solid mechanics, and diffusion tensors occurring in diffusion tensor imaging (DTI), a special magnetic resonance imaging (MRI) modality used to visualize fiber tracts, e.g., in the human brain. In reality, these tensor fields vary in time. However, in practice datasets with temporally varying tensor fields are relatively rare, and visualization methods are mostly designed for instantaneous tensor fields $\mathbf{T}(\mathbf{x})$.

The most important tensor field visualization methods can be roughly classified into five different categories: *direct*, *image-based*, *glyph-based*, *line-/surface-based* and *topology-based*.

2.3.1 Direct Methods

Direct methods display some properties of the tensor directly. Usually, one or more scalar quantities are derived from the vector field and displayed using methods from scalar field visualization. Notable examples for direct methods are *color-mapping* and *direct volume rendering*.

Color-Mapping

Just like scalar and vector fields, 2D tensor fields or slices of 3D datasets can be displayed by color-mapping some scalar properties of the tensor. When investigating mechanical stress tensors, some norm of the tensor is often displayed. When visualizing diffusion tensors from DTI data, the 3D direction of the major eigenvector is often encoded using a radial or spherical color map [34]. Such a visualization is not very intuitive and requires the viewer to be familiar with the interpretation of the resulting images.

Direct Volume Rendering

For 3D data, the equivalent of color-mapping is direct volume rendering. The additional degrees of freedom of a tensor compared to scalar or vector data means that some information invariably gets lost when using normal direct volume rendering techniques. This means an intelligent mapping of the tensor to color, opacity and shading needs to be performed to retain the most important aspects of the data. For diffusion tensors, different possibilities for such mappings have been explored by Kindlmann et al. [35]. They base their mappings on the different kinds of anisotropies indicated by the ratio of the tensor's eigenvalues that can be found in diffusion tensor data (namely linear anisotropy, planar anisotropy, and isotropy). This produces visualizations that represent the important features in DTI data, but is not necessarily applicable to other application domains.

2.3.2 Image-Based Methods

Like the equivalent techniques for vector fields, image-based visualization of second-order tensor fields works by generating space-filling images that are derived from the underlying tensor data. Techniques in this category have yet to be adopted into mainstream visualization tools, so we will only discuss two notable examples: *HyperLIC* by Zheng and Pang [36] and *LIC with variable input textures* by Hotz et al. [37]. Both techniques are modified versions of LIC based on the eigenvectors of the tensor fields.

HyperLIC

HyperLIC was introduced by Zheng and Pang [36] as a visualization technique for diffusion tensor data. In this data, a central characteristic is the diffusion anisotropy represented by the ratio of the eigenvalues. While LIC accumulates the values of an input texture along streamlines of a vector field, HyperLIC conceptually accumulates values in a strip- or tube-like volume that follows the local eigenvector direction and whose cross section is scaled according

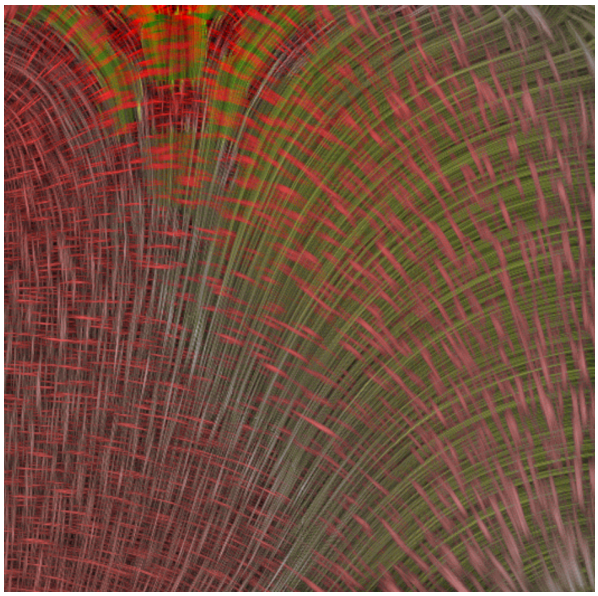

Figure 2.8: LIC with variable input textures. Image source: Hotz et al. [37].

to the local eigenvalue ratio. This produces LIC-like results in areas with a strongly dominating eigenvalue, and blurry areas with no sense of direction in isotropic areas where all eigenvalues are similar. As this technique is designed for diffusion tensors, which are positive definite, it lacks a way of indicating eigenvalue sign and therefore is not well suited for the visualization of indefinite tensors.

LIC with Variable Input Textures

A technique which is better suited for symmetric indefinite tensors, which can have negative eigenvalues, was presented by Hotz et al. [37]. As opposed to HyperLIC, which accumulates the values of the input texture in a volume instead of along a curve, this method uses a standard LIC on the "eigenvector fields" of the tensor field. The remaining information in the tensor is visualized by carefully varying the spot size, spot density and color of the input noise texture as well as the convolution length based on the local eigenvalues. Images from major and minor eigenvectors are overlaid to visualize both at the same time. Figure 2.8 shows an example of this technique applied to a slice of the stress tensor field from a two point load dataset with a pushing and pulling force.

2.3.3 Glyph-Based Methods

Glyph-based methods for tensor field visualization place small geometric objects in space to represent certain characteristics of the local tensor. They

have the advantage of being able to display all features of a tensor at once, but their visual complexity can make them hard to read. Glyph-based methods generally differ by the restrictions they place on the tensor. There are various glyph designs for *symmetric positive definite*, *symmetric (indefinite)*, and *general tensors* in both 2D and 3D. Some research has also focused on how to place and distribute glyphs to better emphasize global structures in the data, as opposed to the simple regular grid-based approach [38, 39].

Symmetric Positive Definite Tensors

Symmetric positive definite tensors are tensors that always have positive real eigenvalues, and whose eigenvectors are orthogonal. The domain most explored in scientific visualization for these tensors is DTI data, where they represent the diffusion of Hydrogen atoms in organic tissue (specifically in neural fibers of the brain). The simplest glyphs for visualizing such tensors are ellipses [40], cylinders [41] or boxes [42]. These glyphs are formed by placing some prototypical base shape (a unit sphere or cube) centered at the origin and then transforming it according to the tensor. The resulting glyph is then placed at the sampling position the tensor originated from. Most often, the glyphs are also scaled to achieve a size fitting with the glyph density. Such simple glyphs accurately represent how the tensor transforms input vectors to output vectors. The disadvantage is that they have various visual ambiguities that make their interpretability less than ideal [43].

Kindlmann [43] solved this problem by carefully designing glyphs based on superquadrics. These superquadrics change their base shape depending on the relationship of the three eigenvalues of the tensor. In this way, ambiguities intrinsic to the usage of constant base shapes are eliminated.

Symmetric Tensors

General symmetric tensors always have real orthogonal eigenvectors, but their eigenvalues can be negative. This poses a challenge for glyph design. Simply transforming a symmetric base shape with the tensor will produce the same image for eigenvalues with equal magnitude but opposite sign. Different glyphs for symmetric tensors in different application domains have been proposed in the literature [34, 44, 45]. Often, color is used to indicate the sign of the eigenvalue. Alternatively, the glyph base shape is modified to represent the difference between positive and negative eigenvalues. Schultz and Kindlmann [46] built on top of all of this work to develop a set of superquadric-based tensor glyphs that clearly indicates eigenvalue sign by a combination of color and concave shape (see Figure 2.9).

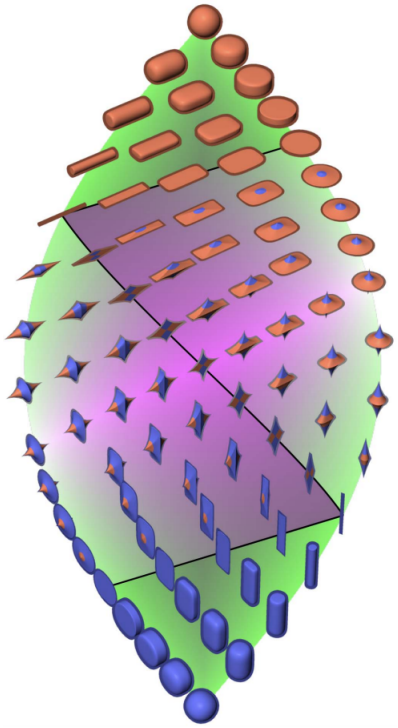

Figure 2.9: Superquadric glyphs for symmetric tensors. Includes glyphs for positive definite tensors as a subset (top triangle). Image source: Schultz et al. [46].

General Tensors

Apart from different eigenvalue signs, the eigenvectors of general (asymmetric) second-order tensors need not be orthogonal, and they can be complex. Gerrits et al. [47] built on top of the work of Kindlmann, Schultz et al. to design a set of glyphs for general tensors in 2D and 3D that can represent non-orthogonal eigenvectors and additionally incorporates information about complex eigenvalues via color.

2.3.4 Line-/Surface-Based Methods

Line- and surface-based methods for the visualization of second-order tensor fields are very similar to integral lines and surfaces for vector fields. They consider the eigenvectors of the tensor field as the equivalent of a vector field and base the visualization on the resulting field lines. Of course these methods can only visualize real eigenvectors and are therefore generally applied to symmetric tensor fields only. Because of the differences between real vector fields and eigenvector fields, modified integration methods are necessary to form these field lines. The most important methods based on this principle are *tensor field lines*, *hyperstreamlines*, *tensorlines* and *hyperstreamsurfaces*.

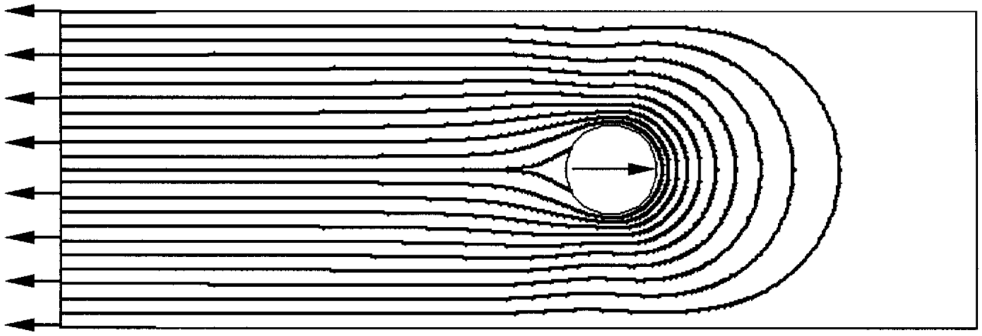

Figure 2.10: Major principal stress trajectories in a plate with a loaded hole. Image source: Kelly and Tosh [52].

Tensor Field Lines

Tensor field lines are lines that are everywhere tangent to an eigenvector of the tensor field. They are the basis for all the methods in this category. Tensor field lines have been known as *stress trajectories* (see Figure 2.10) in the context of solid mechanics since the 1800s. Early stress visualizations using this technique were drawn by hand based on photoelastic measurement techniques [48, 49]. In a computer, these lines are obtained by integrating a vector field generated from the eigenvector field by choosing an orientation and magnitude at each location [50, 51]. Since this choice is not always unique in the vicinity of degenerate points where two or more eigenvalues are equal, special care has to be taken to avoid a sudden flip of direction. The resulting lines show the continuous change of direction of eigenvectors in the tensor field, but they are not well suited to judge the magnitude of the eigenvalues.

Hyperstreamlines

To enhance the information content of tensor field line visualizations, Delmarcelle and Hesselink introduced hyperstreamlines [53]. These hyperstreamlines follow the field lines of one of the eigenvectors of the tensor field, while their cross-section is a cross shape or ellipse aligned with the other two eigenvectors and scaled by the corresponding eigenvalues. The eigenvalue corresponding to the eigenvector parallel to the hyperstreamline is color-coded on the surface. In this way, the full information of the tensors along the hyperstreamlines is visualized. It is important to note here that the term hyperstreamline is sometimes used in the literature to mean what we introduced as tensor field lines. In this work, we use it exclusively for lines with a variable cross-section.

Tensorlines

In the analysis of diffusion tensor fields obtained from DTI scans of the human brain, tensor field lines are tracked to obtain the paths of neural fibers. Because this data suffers from noise and partial voluming effects due to limited resolution, near-isotropic areas can occur where fibers with different directions cross. In these areas, the eigenvector direction is not clearly defined and often dominated by noise. Just following the vector field obtained from a single eigenvector will result in random paths that do not represent the paths of actual brain fibers. To counteract this problem, Weinstein et al. [54] introduced tensorlines. The core of the algorithm is a modified streamline integration that not only takes into account the direction of the major eigenvector but is also guided by the direction of the previous step in near-isotropic regions.

Hyperstreamsurfaces

The concept of hyperstreamlines was extended to hyperstreamsurfaces by Jeremić et al. [45]. They are formed analogous to streamsurfaces by using a curve instead of a point as a seed structure for integration. In this case, the other eigenvectors and eigenvalues can not be sensibly displayed by varying a cross section like with hyperstreamlines. Instead, only the eigenvalue of the integrated eigenvector is color-coded on the surface.

2.3.5 Topological Methods

Similar to scalar- and vector fields, topological structures in tensor fields are defined by some mathematical degeneracy. In contrast to the topology of vector fields, it is not the magnitude of the tensor that is important, but the relationship between the eigenvalues and eigenvectors. For symmetric tensor fields, the topology is formed by *degenerate points and lines* and their *separatrices*, as well as *neutral and traceless tensors*. For *general (asymmetric) tensor fields*, the topology is formed by degenerate structures and circular points.

Degenerate Points and Lines

The core of topological analysis of symmetric tensor fields are degenerate structures. These are structures where two eigenvalues are equal, and the eigenvector directions are not uniquely defined. They are the locations where tensor field lines intersect. In 2D tensor fields, such structures can be classified into trisector and wedge points, depending on the behavior of the tensor field lines in their vicinity. In 3D, degenerate features form lines. Zheng et al. [55]

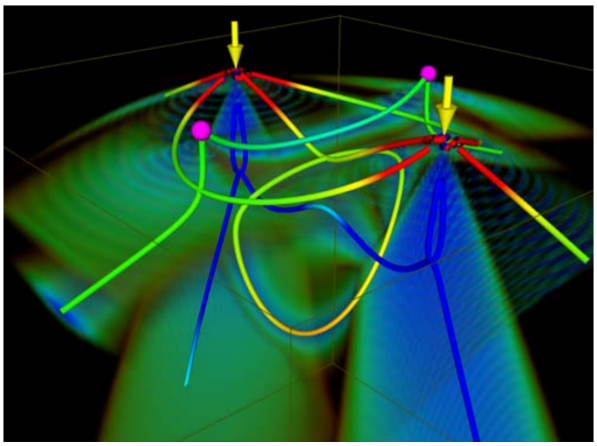

Figure 2.11: Topology of the stress tensor in a double point load dataset. Image source: Zheng and Pang [58].

showed that the type of degenerate point can switch at isolated points along these lines. These are the points where the degenerate line is parallel to the plane spanned by the eigenvectors corresponding to the dual eigenvalue. Degenerate lines in symmetric tensor fields were first studied by Delmarcelle, Hesselink et al. [56, 57]. Numerical algorithms for their robust extraction were developed by Zheng et al. [58, 59] and later adapted for noisy DTI data by Tricoche et al. [60]. Figure 2.11 shows the topology of a common test case in structural mechanics where two point loads are applied to the surface of a solid block.

Separatrices

The tensor field lines that pass through a degenerate point form separatrices that separate the neighborhood of the point into sectors [56]. These structures are akin to the separatrices of vector fields that separate the area around a saddle point. In 2D, these separatrices are lines, in 3D, they form surfaces. An algorithm for extracting such surfaces from 3D tensor fields was proposed by Zheng et al. [55]

Neutral and Traceless Tensors

A different kind of topological feature are neutral and traceless tensors, which were first investigated by Palacios et al. [61]. A neutral tensor is a tensor where the middle eigenvalue is the average of the major and minor eigenvalue. Neutral tensors in 3D tensor fields form surfaces. These surfaces mark the transition between areas of linear tensors (one dominating eigenvalue) and planar tensors (two dominating eigenvalues). In stress tensors, these can be interpreted as the transition between predominantly tensile stress and predominantly compressive stress. In diffusion tensors from DTI, they indicate

the separation between regions with clear fiber direction and regions where fiber tracts cross.

Traceless tensors are tensors whose sum of eigenvalues is zero. Traceless tensors also form surfaces in 3D tensor fields. They mark the transition between areas of positive and negative trace. In stress and strain tensors, these are equivalent to regions of expansion and compression. Recently, Roy et al. [62] proposed a more robust extraction method for these surfaces as well as degenerate lines.

Topology of General (Asymmetric) Tensor Fields

The topology of general tensor fields has been studied to a lesser extent. Zheng and Pang [63] first introduced *circular points* as the main topological structure in 2D general tensor fields. These are points where the tensor is a perfect rotation matrix. Analogous to degenerate points in symmetric tensor fields, these are also points where all vectors are valid eigenvectors. Degenerate structures where two eigenvalues are equal also exist in general tensor fields, but here they mark the transition between regions of real and complex eigenvectors. This means that they form lines instead of points in 2D, and their meaning is very different. Zhang et al. [64] later extended the study of 2D general tensor fields by defining eigenvector and eigenvalue manifolds to classify general tensors and distinguish different types of circular points.

3

Introduction to Turbulent Combustion

Combustion is one of the cornerstones of our civilization. Its applications range from providing light to carrying objects into space. The majority of high-tech combustion processes occur under turbulent conditions. The nature of turbulent flow, and therefore also turbulent combustion, is still being actively studied by the scientific community. Reducing the pollutant emissions and increasing the efficiency of combustion processes is essential in todays world where we are confronted more and more often with the realization that our resources are finite and the damage we do to our environment can not easily be undone.

Simulations are an invaluable tool in the design of improved combustion processes. They allow to test different setups quickly and for a relatively low cost. Efficient simulations need models of the relevant physical and chemical processes. As the demands on the accuracy of such models rises, more detailed insight into the low-level phenomena of turbulent combustion is needed. Obtaining this insight via experiments is challenging. Often only a small number of variables can be observed at the same time and observations are frequently limited to a 2D slice.

Another approach is direct numerical simulation (DNS), which is sometimes

referred to as a "numerical experiment". In DNS, the Navier-Stokes equations are directly solved on a very fine grid, without using a higher-level turbulence model. This is computationally very expensive, which is why DNS is typically performed on supercomputers using hundreds or thousands of cores. In the simulation, all variables are available in full spatial and temporal resolution.

Analyzing the data from such simulations poses a different challenge. Due to the high spatial and temporal resolution, the raw data produced by a single DNS run can range from terabytes to petabytes. Data of this size can not be written to disk or transferred over a network in a reasonable amount of time, even if the enormous storage space that is required was available. This limits the post-analysis of the data to spatially or temporally downsampled versions which lose a lot of important information. In recent years, the subject of *in-situ* analysis and visualization has therefore gained popularity. The idea is to process the data while it is still in memory during the simulation. The raw data is then discarded and only the results, which are typically much smaller in size, are stored on disk.

This thesis contains two new in-situ focused approaches for analysis and visualization of turbulent combustion DNS. To provide some important context, this chapter provides a short introduction into the field of turbulent combustion research, the basics of turbulent combustion modeling and simulation, and an overview of relevant research concerning in-situ and post-processing of turbulent combustion data in particular and large-scale simulations in general.

3.1 Combustion

Combustion is the exothermic chemical reaction of a fuel and an oxidizer into oxidized products and heat. We will only concern ourselves with the combustion of gases, which is the most common case used in industrial applications. The oxidizer is usually oxygen, while fuels can range from simple ones such as hydrogen or methane to complex organic fuels.

A combustion process is a complex system of elementary chemical reactions transforming various chemical species into each other while absorbing or releasing heat. It involves reactants and products but also various intermediate species. These intermediates can be more or less stable and are often radicals with unpaired electrons. The reaction rate of each elementary reaction in an infinitesimal volume is dependent on the amounts (i.e., mass) of the different chemical species as well as the current temperature. The composition of chemical species is typically represented via their mass fractions, i.e., the fractions of the total mass of the mixture that are occupied by each species. This introduces density as an additional variable representing the total mass

per unit volume. Inert gases, which do not participate in the reactions directly can still influence them by their diluting presence.

The chemical reactions in a flame are intrinsically linked to the fluid motion of the gas. As the gas is deformed and transported by the flow, the local concentration and temperature gradients change, which in turn control the diffusion of chemical species and heat. Through diffusion, the local mixture and temperature changes, which in turn influences the reaction rates. As the chemical reactions produce or consume heat, the gas expands or contracts, which in turn influences the fluid's velocity (the expanded gas has to go somewhere) and viscosity (molecules that are farther away from each other interact less). This again influences the mixing and transport of the gas, and so the cycle goes on.

Gaseous combustion can be classified by the type of flow and the mixing of reactants. Is the flow laminar or turbulent, and are fuel and oxidizer premixed or non-premixed? We will discuss the characteristics of flames in each of these regimes in the following sections. This discussion is largely based on the book "Theoretical and Numerical Combustion" by Poinsot and Veynante [65].

3.1.1 Laminar Flames

If the reacting gases in a flame move at low velocities, the flow is often laminar. Laminar flow is characterized by its predictability and lack of chaotic motion. Layers of the fluid slide past each other without significant mixing. In many cases, the flow is steady in this regime.

Even though laminar flames are the exception in industrial applications, their simplicity and deterministic behavior makes them an ideal basis for the theoretical study of combustion processes. They are the reference case that turbulent combustion is compared against and that turbulent combustion models are derived from.

Laminar Premixed Flames

In a premixed flame, fuel and oxidizer are brought into a uniform mixture before ignition. The defining feature of premixed combustion is the flame front that propagates into the fresh gases and leaves behind hot combustion products. In the simplest case, the flame front is planar and travels along its normal direction. In this case, the phenomenon is essentially one-dimensional (see Figure 3.1). The profiles of the combustion variables along the normal direction show high concentrations of fuel and oxidizer on the fresh gas side and high combustion product concentrations and temperature on the burnt gas side. In between the two extrema is the flame front that shows the gradual transition between fresh and burnt gases. The concentration of intermediate

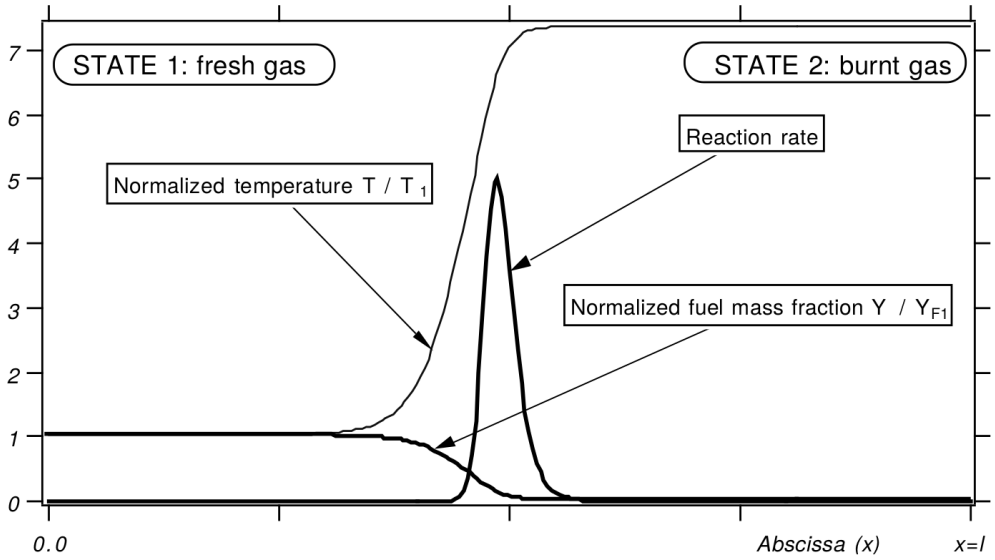

Figure 3.1: Basic configuration of a one-dimensional laminar premixed flame. Image source: Poinsot and Veynante [65]

species is highest in the flame front. Intermediates generally do not occur in the fresh gases, but some radicals may exist on the burnt gas side due to dissociation of combustion products at high temperatures.

The characteristic properties of a premixed flame front are *flame speed*, *flame stretch*, and *flame thickness*. These three quantities depend on each other and the underlying flow field.

The flame speed is the speed at which the flame front travels. It can be expressed relative to a laboratory frame of reference, or relative to the flow. Sometimes, it is also expressed as the speed at which fresh gases are turned into combustion products. The flame speed can vary considerably along the surface of a flame, and depending on conditions, the flame front can travel against the flow at considerable speeds.

The flame thickness describes the width of the reaction zone normal to the flame front. Depending on the application, it is determined in different ways. One common definition uses the slope at the point of highest temperature gradient. Another one is based on the distance between the isosurfaces of minimum and maximum temperature. Different alternative definitions exist as well, some of them based on radical species. The flame thickness is inversely proportional to the flame speed in canonical configurations. A thinner flame travels more quickly and slow-moving flames tend to be thicker. Flame thickness is an essential measurement for combustion simulations, as it determines the grid resolution necessary to resolve the flame front.

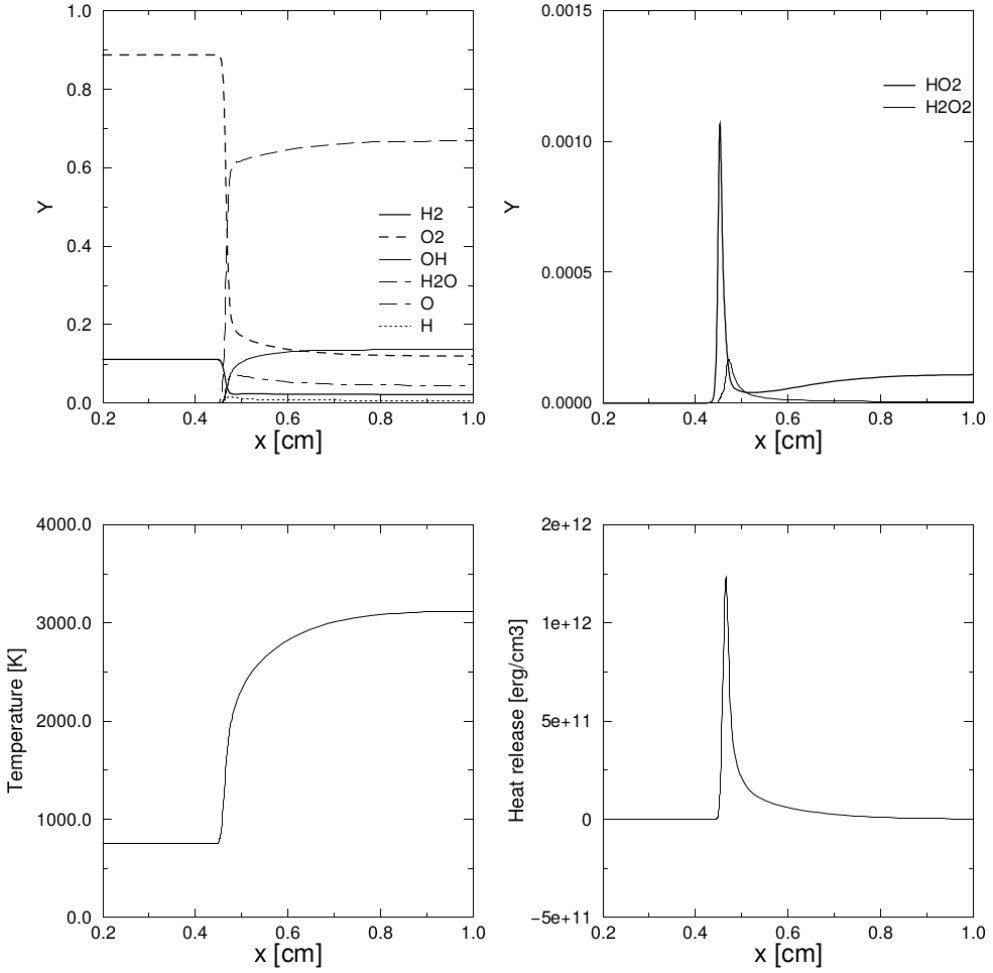

Figure 3.2: Profiles of species (Y), temperature and heat release of a laminar premixed H_2-O_2 flame. Image source: Poinsot and Veynante [65]

Both flame speed and flame thickness are dependent on the chemical reaction taking place and the stretch of the flame front. Flame stretch is the change in surface area an infinitesimal element of the (idealized) flame surface experiences over an infinitesimal time interval. It can be induced by a non-uniform flow, but also by the expansion or contraction a curved flame front experiences due to its propagation in normal direction. Figure 3.3 shows some examples of stretched laminar premixed flames. If the flame experiences positive stretch, it is tangentially expanded and compressed in normal direction. This decreases the flame thickness and feeds new fresh gases to the reaction, which increases the flame speed. However, it also increases heat dissipation. If the flame is stretched too quickly, this can lead to quenching. Negative stretch (compression) leads to less fresh gas being transported near the reaction zone, which increases the flame thickness and reduces flame speed.

Due to their defining characteristic for the behavior of the flame, flame speed, thickness and stretch are the basis for a lot of combustion models, and a focus of study in many combustion experiments and simulations.

Laminar Non-Premixed (Diffusion) Flames

In non-premixed combustion, fuel and oxidizer are supplied separately. Combustion can only occur where both mix in a sufficient ratio via diffusion. This is why non-premixed flames are also called diffusion flames. Figure 3.4 shows a 1D cross-section of a typical laminar diffusion flame.

The prototypical example of a diffusion flame is a jet flame where fuel gas is injected into ambient air and ignited. An example of this type we are all familiar with is the flame of a candle. Wax is evaporated from the wick via the heat of the flame. This fuel gas burns as it comes into contact with ambient air and creates a laminar diffusion flame. The heat generated from the flame is sufficient to ignite the fresh fuel being supplied from the wick and sustain the reaction. Figure 3.5 shows a simplified example of such a flame.

Diffusion flames are easier to set up than premixed flames from a practical perspective, as no perfect mixing of fuel and oxidizer is required beforehand. They are also safer because they prevent flashbacks into the gas supply. However, it is harder to ensure a clean and complete consumption of fuel in non-premixed combustion. This is why diffusion flames are generally less efficient and produce more soot and pollutants that result from suboptimal burning conditions.

Because the conditions for combustion are not satisfied initially, transport and mixing become the main issues in a non-premixed flame. Fuel and oxidizer need to mix sufficiently to create a combustible mixture. Temperature produced by nearby combustion needs to be high enough to ignite the mixture. Combustion products need to be transported away from the combustion zone

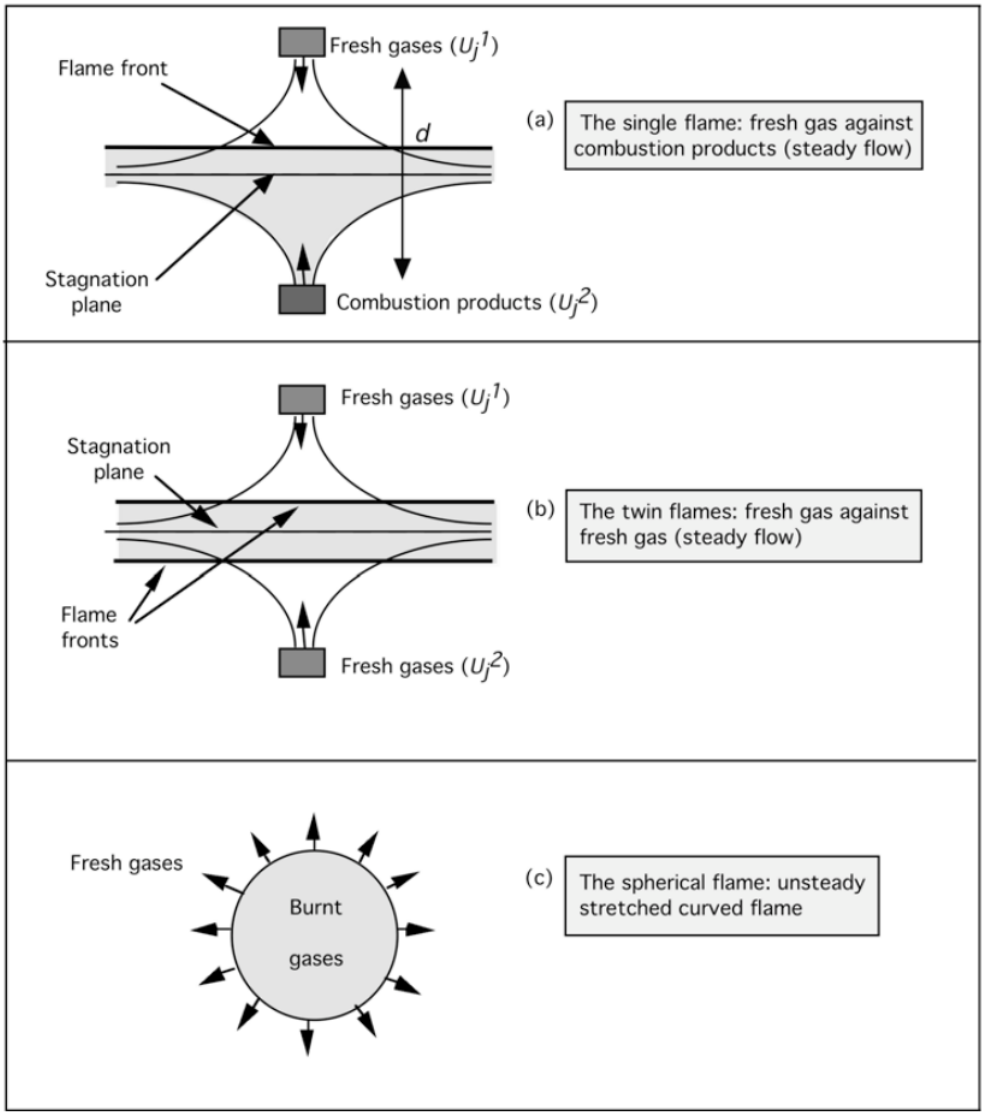

Figure 3.3: Examples of stretched laminar premixed flames. Image source: Poinsot and Veynante [65]

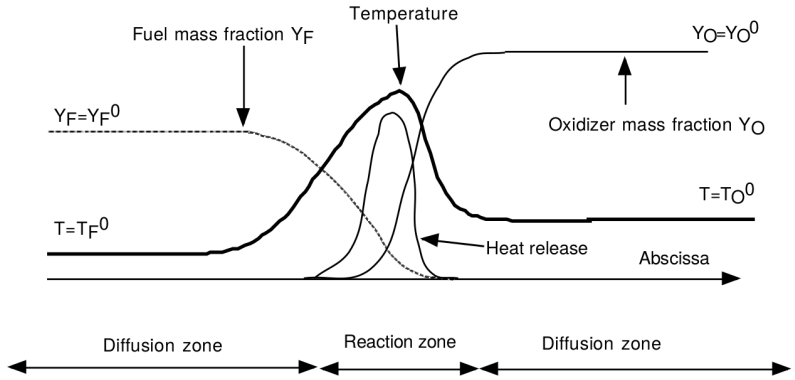

Figure 3.4: Basic configuration of a one-dimensional laminar diffusion flame. Image source: Poinsot and Veynante [65]

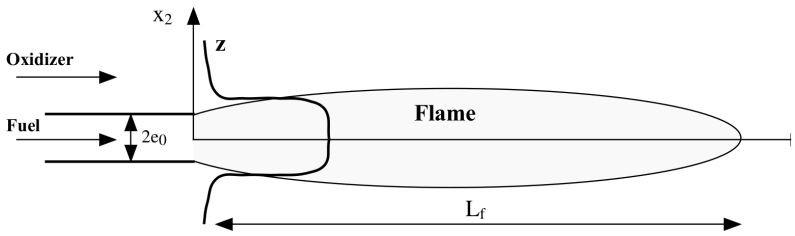

Figure 3.5: Simple 2D laminar diffusion jet flame. Image source: Poinsot and Veynante [65]

fast enough to prevent the flame from suffocating, but not too fast to dissipate the heat required for continued combustion.

Diffusion flames do not propagate like premixed flames. Their position is fixed at the interface between fuel and oxidizer. For this reason, the speed of the flame is not a relevant quantity for analyzing the behavior of diffusion flames. Instead, the defining variable is the mixture fraction between fuel and oxidizer. The flame is generally strongest where the mixture fraction is near stoichiometry, i.e., where fuel and oxidizer are mixed in such a ratio that both are consumed completely by the reaction. Mixture fraction and temperature are the dominating variables controlling a diffusion flame, which is why simple combustion models are based on these two quantities.

Unlike for premixed flames, where it mainly influences the speed of the reaction and can only quench the flame in extreme cases, diffusion flames are very sensitive to flame stretch. A certain stretch is necessary to supply fresh gases to the reaction zone and transport away combustion products that would otherwise suffocate the flame. However, for high flame stretch heat might be dissipated too quickly to sustain the reaction, leading to extinction.

Because diffusion flames are much more sensitive to the flow conditions, and combustion occurs over a wider range of fuel/oxidizer mixture fractions, modeling and simulating diffusion flames is more challenging than premixed flames.

3.1.2 Turbulent Flames

Although laminar flames are the basis for the study of combustion, most industrial combustion applications are turbulent in nature. This makes turbulent combustion an important area of research. Unfortunately, it is also a particularly challenging one. Turbulent flow itself is still not well understood by the scientific community, and understanding and modeling complex chemical reactions still poses a significant challenge. In turbulent combustion, both of these phenomena are combined and interact with each other, which makes understanding even more difficult.

Turbulent flow is a chaotic and essentially statistical process. It occurs when the inertial force of the flow is dominant over the damping effect caused by the fluid's viscosity. This happens at higher fluid velocities, where small imperfections lead to disturbances that are not immediately absorbed by viscous effects.

Turbulent flow is characterized by a mixture of *eddies* of different sizes and energies. The sizes of eddies in a flow range from the integral length scale, that is roughly equivalent to the size of the domain the flow resides in, to the Kolmogorov length scale, that describes the smallest eddies in the system. The energy in a turbulent flow passes along the turbulent spectrum from the largest length scales to the smallest ones. Large eddies spin off smaller eddies, which spin off even smaller eddies and so on until the energy is dissipated as heat at the Kolmogorov scale. This is why turbulent flow will decay over time if it is not continuously supplied with energy.

The interaction between flow and chemistry in a turbulent flame is a two-way street. The reaction transforms and heats the gas, changing its density and viscosity and thereby influencing the flow. On the other hand turbulent flow transports and mixes the gases, thereby influencing the conditions for combustion.

The effect of turbulence on the flame depends on the size and strength of the eddies comprising the flow. Large eddies move slowly and only wrinkle the surface in large scales. This only has an indirect effect on the structure of the flame and the chemical reactions, but influences the global flame shape. Smaller eddies are faster and might disturb the structure of the flame locally. This might have the effect of destroying a clean flame front and causing local extinction, or it might enhance local mixing and improve the conditions for combustion. If the eddies are too small, they might be dissipated before they

Figure 3.6: Example of a turbulent non-premixed flame: The burner of a hot air balloon.

are able to influence the reaction significantly. Which scales interact with the flame in which way is also dependent on the speed at which the chemical reaction happens, the flame thickness, and in the case of premixed combustion, the flame speed. Building accurate models for turbulent combustion requires investigating the interplay of all these variables and more.

Turbulent Premixed Flames

In premixed flames, turbulence has the primary effect of enhancing combustion. The turbulent flow wrinkles the flame surface and increases its surface area. A higher surface area means that reaction happens at more places at once. The global flame speed increases and fresh gases are consumed more rapidly. However, at high turbulence intensities, the heat produced by the reaction will be dissipated too quickly and the flame might be extinguished.

Due to the massive change in viscosity on the burnt gas side, the turbulence intensity in premixed flames is typically much stronger in the cold fresh gases. This difference is so significant that the flow on the burnt gas side may become laminarized, depending on the initial turbulence intensity.

In practice, premixed combustion occurs for example in spark-ignited internal combustion engines. Here, air-fuel mixture is sucked into the cylinder,

compressed and then ignited. Starting from the ignition point, the flame front travels through the cylinder, being wrinkled by the turbulent flow of the gas and transforming fresh gases into hot combustion products.

Turbulent Diffusion Flames

The typical diffusion flame setup is the jet flame. A stream of fuel gas, often turbulent, is injected into ambient air. Mixing between fuel and air is provided by the turbulent shear layer that arises from the difference in velocities between the two gases. This leads to a reaction zone that is typically very thin at the outlet and becomes thicker as more mixing occurs downstream, where a substantial volume is occupied by combustion products. A practical example for a diffusion jet flame is shown in Figure 3.6.

A central problem of turbulent diffusion flames is flame stabilization. Typically, combustion does not start directly after the fuel gas leaves the inlet, but only after some mixing has occurred. This mixed gas then needs to be ignited by the already-burning mixture further downstream. This lateral propagation of the flame needs to happen at least at the same velocity as the flow, otherwise the flame will be blown off. To make this process more stable, some setups use small premixed pilot flames between fuel and air stream to provide a steady source of heat.

If mixing occurs faster than combustion, or if the flame is locally extinguished due to high flame stretch, pockets of premixed gas can form that are later ignited when coming into contact with high-temperature regions. This means that parts of a diffusion flame might actually be in the premixed regime. This highlights the complexity of diffusion flames compared to premixed flames. Due to the additional problem of mixing, diffusion flames are much more sensitive to turbulence. Depending on their setup, many different conditions and mechanisms can be in effect at the same time. This makes non-premixed flames very challenging to model and simulate.

3.2 Modeling and Simulation of Turbulent Combustion

Simulations are used to make predictions about the behavior of a system. In turbulent combustion and other engineering application they provide a quick and relatively cheap way of testing setups compared to performing experiments. A simulation is only useful if it is sufficiently accurate. This is why good models of the underlying processes are so important. On the other hand, simulations also need to be efficient. If running the simulation is more expensive and time-consuming than setting up a comparable experiment, the simulation loses its value.

For turbulent combustion, finding a good trade-off between accuracy and efficiency is very much an open problem. Both turbulent flow and chemical reactions are complex to model and expensive to simulate on their own. Solving both at once in a turbulent combustion simulation predictably leads to a vast increase in the required computing time and resources. Frequently simplifying assumptions are made to gain efficiency. Which assumptions are valid under which circumstances is the central question when building turbulent combustion models.

This section provides an overview of modeling strategies used in turbulent combustion simulations. It places a particular focus on DNS, as it is most relevant for the content of this thesis.

3.2.1 Chemical Schemes

Modeling a chemical reaction can be a complex task. Most reactions do not consist of a single step but rather a complex system of intermediate reactions. Many of these reactions can be reversible and their reaction rates depend on the current composition and pressure or temperature of the gas. The more complex the fuel, the more intermediate steps are possible and need to be considered when modeling the reaction. A system of intermediate steps that models a whole reaction is called a chemical scheme (see, e.g., Table 3.1). Each individual reaction step has a number of parameters controlling the reaction rate in dependence on the current conditions.

Reaction modeling is the task of breaking down the wealth of possible intermediate reactions into a subset that describes the whole reaction with sufficient accuracy for a given application, and determining the right parameters for each one. For complex hydrocarbon fuels the number of different species considered can easily go into the hundreds, while thousands of intermediate steps contribute to the reaction. Even for seemingly simple reactions, such as hydrogen and oxygen to water, the reaction schemes can be surprisingly complex. The scheme by Miller et al. [66] displayed in Table 3.1 has 9 species and 19 different reversible reactions. Adding this to an already computationally intensive fluid dynamics simulation that only involves five different variables (density, temperature and three velocity components) increases the computational load immensely. This is amplified even more by the fact that chemical reactions, especially the intermediate ones in a reaction scheme, typically happen at much smaller time scales than the flow. This means that compared to non-reacting flows, not only do the number of equations and variables increase immensely, but the simulation time steps also become much smaller.

There are some strategies to increase efficiency. The simplest one is to use single-step chemistry. Here, the reaction is assumed to be infinitely fast

Table 3.1: Example scheme for a seemingly simple chemical reaction: hydrogen and oxygen react to produce water [66]. M is a placeholder for any third molecule that is needed to absorb and dissipate excess energy to stabilize the product.

Elements:	H, O, N	
Species:	H_2, O_2, OH, O, H, H_2O, HO_2, H_2O_2, N_2	
Reactions:	$H_2 + O_2 \rightleftharpoons 2OH$	$2OH \rightleftharpoons O + H_2O$
	$H_2 + OH \rightleftharpoons H_2O + H$	$H_2 + M \rightleftharpoons 2H + M$
	$H + O_2 \rightleftharpoons OH + O$	$O_2 + M \rightleftharpoons 2O + M$
	$O + H_2 \rightleftharpoons OH + H$	$H + OH + M \rightleftharpoons H_2O + M$
	$H + O_2 + M \rightleftharpoons HO_2 + M$	$HO_2 + H \rightleftharpoons H_2 + O_2$
	$H + 2O_2 \rightleftharpoons HO_2 + O_2$	$2HO_2 \rightleftharpoons H_2O_2 + O_2$
	$H + O_2 + N_2 \rightleftharpoons HO_2 + N_2$	$H_2O_2 + M \rightleftharpoons 2OH + M$
	$OH + HO_2 \rightleftharpoons H_2O + O_2$	$H_2O_2 + H \rightleftharpoons H_2 + HO_2$
	$H + HO_2 \rightleftharpoons 2OH$	$H_2O_2 + OH \rightleftharpoons H_2O + HO_2$
	$O + HO_2 \rightleftharpoons O_2 + OH$	

and the flame front to be infinitely thin. Fuel and oxidizer are immediately transformed into products and heat once the conditions for combustion are met. In this case, no intermediate reactions are considered, which greatly decreases the number of variables and equations to solve and allows larger time steps. Due to the strong assumptions made when using this method, the results can be very inaccurate, but it can be useful to get a qualitative result. More accurate but also more expensive methods precompute a lookup table covering the relevant portion of the parameter space or reduce the parameter space to lower-dimensional manifolds. Explaining these methods in detail is out of the scope of this work.

Modeling chemical reactions is a complex field in its own right. Turbulent combustion researchers mostly use existing chemical libraries and schemes in their codes. Often, the trade-off between accuracy and efficiency has to lean heavily towards efficiency to keep simulation times and memory usage in a reasonable range.

3.2.2 The Flamelet Assumption

Moving from the modeling of reactions in an idealized uniform mixture to turbulent flames, we need a model for the structure of the flame. The most common assumption is that the flame front is thin compared to the scales of the turbulent eddies. This means that the flame front can be approximated

by an isosurface. In the case of premixed flames, it is an isosurface of the *combustion progress variable*, which describes the progress of the reaction from reactants to products as a number between 0 and 1. This variable is commonly defined as $(T - T_u)/(T_b - T_u)$, where T_u and T_b are the temperature of the burnt and unburnt gases. In the case of diffusion flames, the flame front is approximated by an isosurface of the fuel-oxidizer mixture fraction.

Assuming a thin flame front allows to treat the turbulent flame like a set of locally laminar flames, so-called *flamelets*, that are wrinkled but not disturbed by the flow. Orthogonal to the surface, the flame is assumed to behave like the 1D laminar equivalent. Models for laminar combustion can then be directly used to determine the behavior of the flame at each location on the idealized flame surface. This includes models for the relationship between flame speed, flame stretch, flame thickness, reaction rates, heat release and so on.

The flamelet assumption is frequent in combustion modeling, as it greatly simplifies the flame structure and limits the effects of turbulence on the behavior of the flame that need to be considered. Of course, the conditions for this assumption are not always fulfilled:

- Parts of the flame might be locally quenched. In the case of premixed flames, this allows fresh gases to penetrate into the burnt gas side and vice-versa, something that does not happen for laminar flames. In the case of diffusion flames, this allows the formation of pockets of premixed gas that are later ignited and do not confirm to the ideal diffusion flame any more.
- The ideal flame front might be disturbed by small-scale mixing. In this case, the flame structure can be very different than that of laminar flames. The distinction between burnt/unburnt or fuel/oxidizer side becomes less clear and the flame structure can no longer be adequately described by a simple isosurface.

Due to the simpler flame structure and the smaller sensitivity to turbulence, the flamelet assumption is more often valid for premixed than for diffusion flames. More complex models that allow some interaction of turbulence with the small-scale flame structure are subject of ongoing research.

3.2.3 High-Level Models: RANS and LES

Computational fluid dynamics (CFD) knows three main categories of models for simulating turbulent flow. They are, in decreasing order of abstraction, Reynolds-averaged Navier-Stokes (RANS), large eddy simulation (LES) and direct numerical simulation (DNS). For simulating turbulent combustion, these are extended by combustion models according to their character. Part II of this thesis focuses heavily on DNS, which is discussed in detail in the next

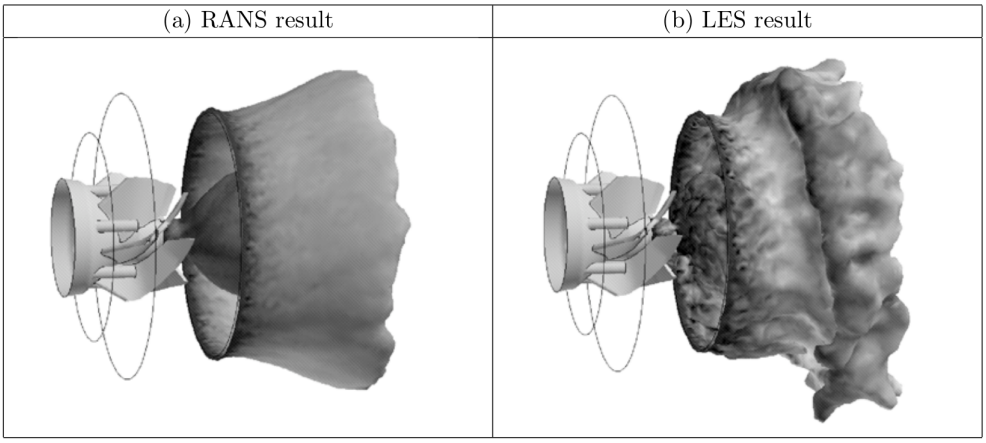

Figure 3.7: Comparison of instantaneous temperature isosurfaces in a swirled combustor when using RANS and LES. Image source: Poinsot and Veynante [65]

section. Since DNS is one of the main tools for validating and building RANS and LES models, we will briefly discuss the ideas behind both here.

Reynolds-Averaged Navier-Stokes

Reynolds-averaged Navier-Stokes equations describe the time-averaged behavior of the system. It is based on the decomposition of a turbulent flow into mean and fluctuating parts. The result are average values for all simulation variables at each location. Images of RANS simulations show very smooth fields that have almost no small-scale features (see e.g., Figure 3.7). The typical RANS equations assume a steady-state system, but the unsteady RANS (URANS) variant can account for some unsteady behavior if it is slow compared to the turbulent timescales.

Combustion models for RANS can be based on quantities such as the mean flame surface area per unit volume, turbulent mixing rates, or probability density functions based on one-point statistics.

Since RANS simulations only resolve time-averaged values, their results have to be interpreted with care. Mean values reported by RANS at a certain location say nothing about the possibly large fluctuations that occur there. However, RANS is the most popular and widespread method for simulating turbulent combustion because of its low computational cost.

Large Eddy Simulations

A step up from RANS in terms of accuracy and computational cost are LES. They reproduce unsteady, but low-pass filtered effects of the system. The

simulation runs on a lower-resolution grid and only resolves the larger scales explicitly. Small-scale sub-grid effects are modeled. LES produce an unsteady, but "blurry" representation of reality (see Figure 3.7).

Combustion models for LES face the challenge that the flame front is generally much thinner than the grid resolution. Combustion phenomena that control the evolution of the flame front happen almost exclusively at sub-grid scales. LES approaches deal with this by describing the flame front in terms of some filtered variable that can be resolved on the grid. Some approaches artificially thicken the flame front, others assume the flame front as an isosurface of some smooth variable. In any case, small-scale wrinkling of the flame front cannot be resolved in LES and needs to be expressed via models.

With the increase in computing performance in recent years, LES has gained popularity and is seeing more widespread use. However, LES of turbulent combustion is still maturing, and accurate sub-grid models for combustion are the subject of ongoing research.

3.2.4 Direct Numerical Simulations

The most accurate and detailed numerical results in CFD are produced by DNS. In these simulations, all time and length scales are resolved on a regular grid. On this grid, the Navier-Stokes equations are solved directly without using a model for turbulence. This produces an accurate representation of reality, which is why DNS is also often referred to as a "numerical experiment".

Its brute-force approach means that DNS is conceptually relatively simple compared to RANS and LES, but it is the most demanding of the three in terms of computational resources and time. As a consequence, DNS are typically used only for small domains of a few centimeters at most. Although methods for more complex setups and geometries are under active development, the typical case is a rectangular box with simple boundary conditions. This is why non-reacting DNS has typically been used to study turbulent flows near walls, between parallel plates or behind simple rectangular geometries.

Like for RANS and LES, the choice of a chemistry model for DNS is largely dependent on the available computing resources and performance demands. If we apply the ideal of solving everything without high-level models, then the logical choice would be using complex chemistry with full chemical schemes. Unfortunately, this is rarely feasible. Even adding single-step chemistry to a non-reacting DNS can result in a hundred-fold increase in computing time due to the smaller time steps and grid sizes required. Using complex chemical schemes only amplifies this problem. For complex fuels, the only choice for getting results in a reasonable time frame is therefore to make a massive trade of accuracy in favor of performance. Even then, turbulent combustion DNS of non-trivial cases can only be run on large supercomputers using thousands of

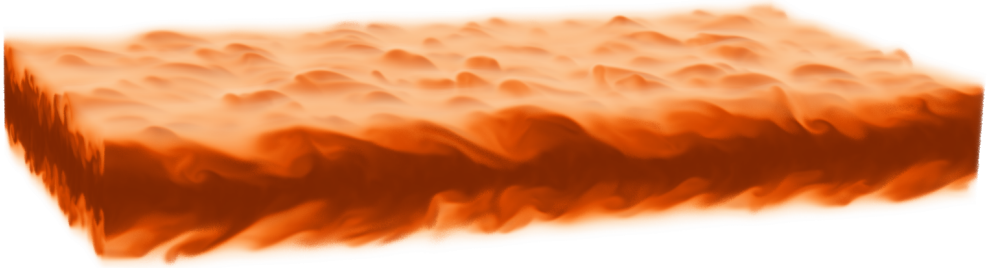

Figure 3.8: Direct volume rendering of the mixture fraction in a DNS of a turbulent non-premixed flame.

parallel cores. See S3D [67, 68] and DINO [69] for two modern DNS codes.

Despite the large computational cost and restriction to simple case setups, DNS is an important tool for combustion modeling. It provides high-resolution 3D unsteady data for all variables, which is still impossible to obtain via experiments. This data allows an in-depth analysis of the behavior of turbulent flames that can be leveraged to build, validate and improve higher-level models.

DNS is used mainly for two purposes: Gaining data to validate and fine-tune RANS and LES models, and gaining a deeper understanding of turbulence/chemistry interactions to derive new models. A wide variety of analysis approaches are applied to these effects. These range from very simple validation by looking at aggregated quantities to the investigation of complex effects such as the different mechanisms for re-ignition of locally extinguished parts of the flame. Providing an exhaustive discussion would go beyond the scope of this chapter, but we will take a look at some examples to get an idea of the kind of properties that are investigated.

The simplest forms of analysis are applied for the validation of RANS and LES models. Here, the same case is simulated once using RANS or LES, and once using DNS. Validation can happen at a high level by comparing aggregated quantities such as the average or maximum temperatures, or on a lower level by comparing point-wise quantities or statistics. In this case it is important to take into account the differences in representation between DNS and the higher-level models. In the case of RANS, DNS results have to be temporally averaged in order to be meaningfully comparable. For LES, the DNS data needs to be low-pass filtered before comparison.

More complex analysis is needed when checking the validity of basic assumptions underlying the high-level models, such as the flamelet assumption. In the case of RANS, this can be done by computing different kinds of statistics, depending on the combustion model used. Simple point statistics can be computed without regard for the flame structure. If the flame front is thin,

flamelets may be extracted as profiles orthogonal to an isosurface representing the flame surface, or along trajectories of the temperature or mixture fraction gradient. Statistics may also be derived from ensembles of iso-levels of characteristic variables such as temperature or mixture fraction. The result are distributions as a function of this variable. Finally, space-averaged statistics might be used to determine quantities such as the average amount of flame surface area per unit volume.

Other quantities that are often investigated due to their significance in combustion modeling are the flame speed (if applicable), thickness, surface stretch and curvature. The relationship of all of these quantities with the structure of the flame is still not completely understood. This is why DNS results are frequently compared to laminar flames as a baseline. For example, Sankaran et al. [70] analyzed the flame thickness and curvature in a premixed jet flame statistically and compared the results to a laminar flame. Hawkes and Chen [71] investigated the statistical similarity of methane-air flames in the "thin reaction zones" regime, which uses slightly weaker assumptions than the flamelet regime, to strained laminar flames. The approach presented in Chapter 4 facilitates such studies by providing a representation of a premixed flame from which such statistics can readily be computed, as well as enhancing it by an additional possibility of visual analysis.

A promising area of current research is concerned with the mechanisms of local extinction and re-ignition and the effect of unsteady, non-instantaneous effects on the flame. These are especially interesting for diffusion flames, as they are more sensitive to turbulence and local extinction has a major significance for the applicability of flamelet models. Many works in this area use Lagrangian approaches to study the temporal behavior of flame elements.

Yeung et al. [72] tracked ensembles of points attached to material- and flame surfaces in premixed and diffusion flames. They recorded the histories of strain rates acting on these surface points to determine under what circumstances a flame surface remains close to an initially coincident material surface. Statistics extracted from the ensemble of surface points allowed them to determine in what way the flame surface grows or shrinks over time and how the strain experienced by a flame surface element affects the flame dynamics.

Sripakagorn et al. [73] tracked points attached to an isosurface of the mixture fraction over time to detect and classify local extinction and re-ignition events. They showed that extinction happens primarily if the flame surface is subject to large amounts of strain over a certain time period. They also identified three different mechanisms for re-ignition of previously extinguished parts of the flame.

Scholtissek et al. [74] tracked complete flamelets represented as trajectories of the mixture fraction gradient emanating from points on the mixture fraction

isosurface. They analyzed the histories of these flamelets and derived a new flamelet model that accounts for curvature-induced tangential transport between adjacent flamelets.

Such Lagrangian approaches are becoming more prevalent in the combustion community. They provide an important tool for the derivation of new combustion models that incorporate unsteady effects. Chapter 5 of this thesis presents an approach for tracking the complete flame surface that is intended to support such investigations in large-scale DNS.

3.3 Visualization for Turbulent Combustion Simulations

Apart from the mostly statistical forms of analysis that have been traditionally employed by turbulent combustion researchers, visualization has become an important tool for gaining understanding from simulation data. Simple visualization techniques such as slicing, isosurface rendering and scatter plots are routinely used by combustion researchers to gain an overview of the data. Special techniques for visualization-supported analysis have been developed to answer more complex research questions. They are essential for understanding the increasingly complex phenomena incorporated into modern RANS and LES models.

As the computing power of supercomputers is steadily increasing, so is the size and complexity of problems simulated with DNS. Storage and network infrastructure are developing at a much slower pace. This has brought us to a situation where the bottleneck in the simulation pipeline has shifted from the CPU to the hard disk. A single snapshot of a 3D turbulent combustion DNS run occupies tens to hundreds of gigabytes. With thousands of time steps per simulation the raw data produced by a single simulation can easily reach into the tera- or even petabytes.

The problem with this is twofold. First, most supercomputers today simply do not have the hard disk space to store more than a couple of simulation runs completely. Second, writing the raw data to disk after each iteration slows down the simulation by an order of magnitude or more. As a result, researchers often store only a few snapshots with large temporal gaps in between. This approach limits the analysis of the data to instantaneous quantities. Unsteady effects become almost impossible to observe and rare events such as local flame extinction are frequently missed.

In recent years, in-situ approaches have been established as a way to overcome this problem. The idea is to process the data in parallel to the simulation while it is still in memory. Only the results of the visualization/analysis, which are typically orders of magnitude smaller than the raw data, are then stored to disk. Performing such in-situ processing is challenging because it

requires a higher degree of technical sophistication and often comes with the drawback of reduced interactivity. However, it can take advantage of all the raw data available during the simulation and therefore enables a much more detailed analysis.

This section gives an overview of the state of the art in visualization for turbulent combustion simulations. We will start with simple and advanced post-processing techniques and then go on to general purpose frameworks for in-situ applications and in-situ approaches specialized to turbulent combustion.

3.3.1 Post-Processing

The classic approach for gaining insight from simulation data is to store it on a hard disk, possibly transfer it to a dedicated analysis workstation, and then apply any visualization and analysis methods as a post-process. Often basic visualization techniques are sufficient for most analysis tasks. However, several methods that support specific questions in turbulent combustion research have been developed.

Basic Visualization

Combustion researchers have always used simple visualization methods to analyze their data. Plots of aggregated or point-wise variables over time, space, or other variables give an idea about the state and behavior of the system. Scatterplots reveal correlations and probability density functions that are the basis for modeling turbulent combustion phenomena.

Every commercial and open source software package for scientific visualization offers basic facilities to explore the three-dimensional structure of the data. Color-mapping on cutting planes (slicing) and isosurface rendering are among the most commonly used techniques for visualizing scalar and velocity data. For example, the shape of the flame is represented by an isosurface of the temperature or mixture fraction (see Figure 3.9). Isosurfaces or slices of the velocity magnitude and vorticity are often used to get a rough idea about the shape and turbulence level of the flow. Scientists sometimes use direct volume rendering to show the full 3D data. This is particularly attractive as it allows visualizing several variables at the same time by carefully choosing transfer functions. If the detailed structure of the flow field is important, it is often visualized with streamlines, although this is less useful for high levels of turbulence.

Multiple simple techniques can be combined to build more powerful visualization pipelines. As an example, a combustion researcher might first extract the isosurface of the stoichiometric mixture fraction as a representative

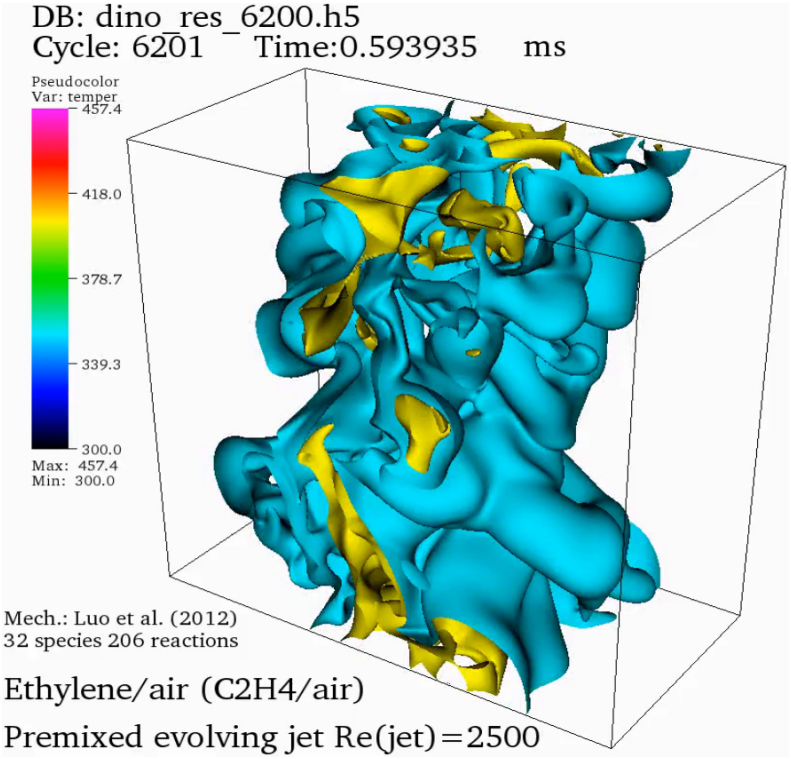

Figure 3.9: Temperature isosurfaces of a premixed jet flame. Simulated with DINO [69] and visualized using VisIt [75].

of the flame front. She then applies a temperature threshold to extract all regions of the flame surface that are currently considered extinguished. In these regions, she plots a histogram of the instantaneous strain rate of the surface and compares it with a histogram of the same variable for the burning regions of the flame, to investigate the statistical significance of the strain rate for local extinction.

As the size of the raw data increases, visualization often has to be performed on a supercomputer as well. Many supercomputing centers have clusters with dedicated visualization nodes or entirely separate clusters for visualization. Visualization software such as ParaView [76] and VisIt [75] have the capability to run on supercomputing clusters and split the work of processing and rendering datasets over many processes.

Special Methods for Turbulent Combustion

Basic visualization techniques can go a long way towards assisting researchers in analyzing their simulation data. However, specialized methods are some-

times needed if the research question is very complex or the computational demands are high. Here we discuss some visualization and analysis tools that were developed specifically for turbulent combustion applications.

Zistl et al. [77] developed a toolbox focused on the analysis of DNS data. It combines steps such as geometrical analysis of the flame surface and flame structure, quantification of turbulent flow properties and turbulence/flame interaction, and statistical analysis. This allows for a more streamlined process when performing common research tasks on the data.

Another group of visualization tools focus on the identification and tracking of volumetric features. Bremer et al. precompute a merge-tree representation and an accompanying segmentation of volumetric [78, 79] and surface features [80] defined by thresholding. This allows an interactive post-hoc exploration of the threshold parameter space and resulting tracking graph representing the evolution of features over time. Wang et al. [81] focuses on an efficient parallel algorithm to identify and track volumetric features in a distributed memory environment where features may span several processors. Schnorr et al. [82] track cells in a space-filling topological segmentation defined by the Morse-Smale decomposition of a scalar variable (see Section 2.1.2) by solving two graph optimization problems. These cells, which are called dissipation elements in combustion literature, carry significance in flamelet modeling.

As a first step towards analyzing unsteady features, some DNS codes have begun saving large amounts of Lagrangian particles (pathlines) and the accompanying time series of simulation variables in addition to snapshots of the (Eulerian) state of the simulation. These pathlines allow the tracking of volumetric features even in snapshots with large temporal gaps, as shown by Sauer et al. [83]. They can also be visualized by themselves, to get an idea about the dynamic behavior of the system. Wei et al. [84] developed a technique that combines a visualization of particles in physical space with a visualization in the temperature – mixture fraction phase space. Trajectories are clustered in phase space to identify different chemical behaviors, and the classes are then displayed in physical space for a visual analysis (see Figure 3.10).

3.3.2 In-Situ Processing

With disk space and I/O speed being the bottleneck in today's large-scale simulations, in-situ approaches to visualization and analysis have gained popularity through the last 15 years. Here, the visualization and analysis of the data is performed in parallel or interleaved with the simulation, without first writing it to a hard disk. This allows for the analysis of the complete simulation output, rather than the infrequent snapshots that are used in post-processing approaches. Ma gave an overview of the problems and opportunities inherent in in-situ visualization [85].

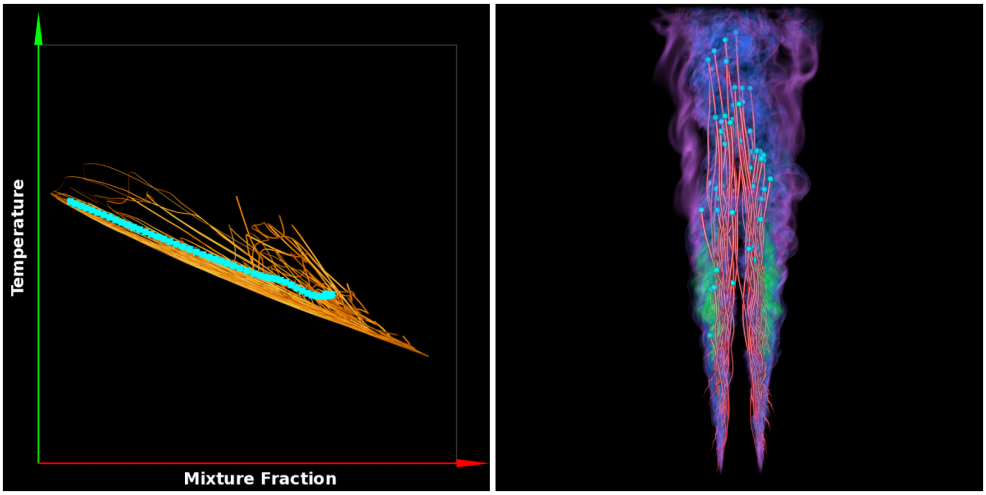

Figure 3.10: Hybrid visualization of particle trajectories in phase space (left) and physical space (right) of an ethylene/air jet flame. Image source: Wei et al. [84].

A central complication of in-situ approaches is the loss of flexibility and interactivity. Large simulations typically need to be submitted to a job queue before they are started on a supercomputing cluster. This means that there might be a significant waiting time before any results can be viewed. It is often unreasonable to expect that an analyst is present to look at the data while some interesting feature or behavior can be observed during the simulation. Additionally, simulations can take several days or even weeks to complete. This makes an exploratory analysis of the data very challenging. Many in-situ approaches therefore have a batch processing character, where visualization and analysis tasks are defined beforehand and are then carried out during the simulation with little to no user input. The user then explores the significantly smaller results, which ideally still contain all relevant information.

Another problem is performance. Since large simulations can already take a very long time, researchers are reluctant to accept a significant increase in computing time in return for visualization. The simulation data is distributed across the nodes of the supercomputer to optimize the simulation time. This is not necessarily an optimal distribution for visualization tasks. The result can be a significant communication overhead, which is a major bottleneck in supercomputing clusters. In-situ algorithms need to be carefully crafted to find a good balance between communication overhead and efficiency gained by data reorganization.

A lot of groundwork is being done to create the foundations for successful in-situ processing. We will first give an overview of these technical contributions and then present some examples for specific in-situ visualization techniques.

Technical Foundations

In-situ processing is a complex task not only from a conceptual, but also from a software engineering perspective. In-situ algorithms need to run on large supercomputers in parallel or even on the same nodes as the simulations. They need to somehow get the relevant data from the simulation and they need to be efficient enough to not slow it down by an unreasonable amount. Communication between computing nodes flows over relatively slow network connections, requiring different parallel algorithms than for shared-memory multi-core architectures. Additionally, computing nodes often do not have dedicated graphics hardware, which makes software rendering a necessity. All this requires a solid technical foundation of systems and algorithms that facilitate the visualization and analysis tasks that eventually derive insight from simulations.

The two most popular open source software packages for scientific visualization, ParaView and VisIt, both include an extension for in-situ visualization. In the case of ParaView, it is called Catalyst [86]. VisIt's in-situ library is LibSim [87]. Both require a certain amount of instrumentation of the simulation code to function. If a simulation code has been interfaced with the visualization software, it communicates its data to parallel visualization server processes that are launched together with the simulation and eventually to a client that allows a similar interaction as if the data was loaded from a file. In addition, the user can define non-interactive batch visualization tasks that are executed regularly and produce images or data files.

Larsen et al. presented a simpler and more flexible system that only supports batch visualization with Strawman [88]. It is based on EAVL [89], a visualization library that is designed fundamentally to run on massively parallel architectures. Libraries with similar goals exist in DAX [90] and PISTON [91]. All three have merged their efforts into a single project: VTK-m [92]. The goal is to develop a native visualization library for supercomputers and other highly parallel architectures that can be a basis for scientific visualization in a future with steadily growing data sizes and a growing need for in-situ capabilities.

Many existing tools for in-situ processing require changes to the simulation code in order to receive data from it. This can be an additional obstacle for the adaption of these tools by simulation scientists. One possibility to reduce this invasiveness is to link the visualization and analysis tasks to the I/O library the simulation code uses to write out its data [93–95]. This unifies the tasks of writing snapshots to disk, analyzing and visualizing them into the same framework and effectively decouples them from the simulation itself. The Freeprocessing system [96] takes an even less invasive approach. It places itself between the I/O calls and the simulation code at library load time,

requiring no changes to the simulation code at all.

Another focus of research has been dedicated to improving performance. Researchers have tried sophisticated load-balancing algorithms to take advantage of unused resources during a simulation run for visualization and analysis tasks [97]. A popular alternative to processing the data on the same computing nodes as the simulation is to offload it onto a separately allocated number of staging or processing nodes [98–103]. Using this approach, which is often called *in-transit* processing, the simulation is only slowed down by the time it takes to copy the data to the staging nodes, where processing can happen in parallel to the next simulation step. Some work has also been done to optimize existing parallel rendering algorithms for use on supercomputing clusters [104–109].

In all of the works presented above, efficiency and convenience of in-situ processing was the focus. The second major problem, namely the lack on interactivity, has been addressed to a lesser extent. Maybe that is because it is much harder to solve. The challenge is to enable an exploratory analysis of the simulation data, i.e., with little to no prior knowledge of what one is looking for. This stands in direct conflict with the goal of in-situ approaches to reduce the amount of data that has to be stored to disk. In order to reduce the data, one must know what is or is not important. A small number of works has tried to address this issue.

One obvious possibility is to compress the data using some generic concept of "importance" that is not dependent on a particular application. Lakshminarasimhan et al. [110] developed such an algorithm whose core idea is to sort the data before compression to take advantage of the special characteristics of scientific data. With a compression ratio of about 1:7, this approach allows to store a much larger amount of data while still retaining complete freedom for an exploratory analysis of the data.

Another approach has been proposed by Kageyama and Yamada [111] and Ahrens et al. [112]: Produce a database of visualizations that the user can explore later. During the simulation, rendered images are produced for a range of parameters of the same visualization. For example, isosurfaces of different variables with different iso-levels are rendered from different camera angles and stored into the database. The user can then select a set of parameters after the fact, retrieve the best fitting image from the database, and even compose multiple images into a combined visualization.

All these contributions address different issues that need to be solved for in-situ processing to become successful. Since the field is still in its infancy, no clear favorites have emerged yet and a definitive in-situ framework has yet to be developed.

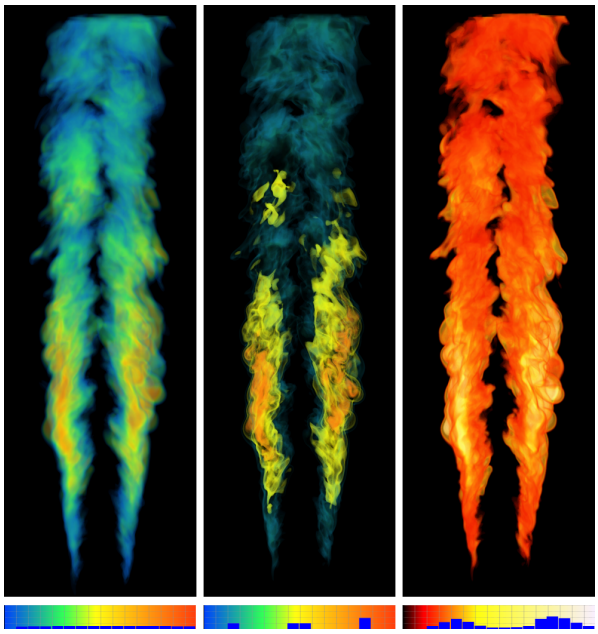

Figure 3.11: Retroactive adjustment of the transfer function in direct volume renderings of a turbulent jet flame. Left: original transfer function. Middle: Opacity modulation. Right: Opacity modulation and recolorization. Image source: Tikhonova et al. [114].

Special Visualization and Analysis Methods

Although most work in the field of in-situ visualization has been done to lay the groundwork for efficient basic visualization, there have been some developments of more specialized in-situ methods. We will discuss some examples here that are applicable to the field of turbulent combustion.

The first in-situ visualization methods focused on rendering images for monitoring the simulation. Yu et al. [113] used their parallel direct volume rendering and image compositing algorithm [104] to display scalar variables in combustion simulations together with particles advected with the flow. This gives an idea about the evolution of the simulation but does not allow for quantitative analysis of the data. Tikhonova et al. [114] developed a more flexible approach by rendering the data into an intermediate representation that stores an attenuation function instead of color values. The transfer function can then be changed after the fact, allowing for a certain degree of interactivity (see Figure 3.11). This approach fits well with the image databases discussed in the previous section [111, 112].

Moving on to more analytic approaches, several works have addressed the extraction and tracking of volumetric features. Chen et al. [115] presented an in-situ system for tracking volumetric features defined by thresholding. Zhang et al. [116] used the distributed online clustering (DOC) algorithm [117] to identify and track features defined by clusters in state space during a simulation run. This can for example be used to identify and track burning

regions in combustion simulations. The IFDT framework [118] takes a more flexible approach by letting a user interactively pick individual structures to track. The system includes a machine learning component that learns to identify the features picked by the user based on their visual properties. This is intended to be more robust than detection based on thresholds which might not be applicable over long time periods. Ye et al. [119] combined image-based visualization with feature tracking by storing rendered depth maps of multiple isosurfaces. These are later used to create visualizations and to track features in image space based on their depth information. Chapter 5 of this thesis presents an in-situ tracking algorithm for surfaces which explicitly captures temporal correspondence and tangential movement of surface points.

The idea of precomputing an intermediate representation of the data in-situ and then exploring it post-hoc has also been applied in some more specialized visualization systems. Landge et al. [120] adapted the computation of segmented merge trees [78, 79] for in-situ applications. This allows for an interactive exploration of the space of possible thresholds or iso-levels for scalar variables. Ye et al. [121] proposed a system for exploring joint field/particle datasets as are for example produced by some combustion DNS codes. They compute probability density functions (PDFs) to represent the field data, and reorganize the particle data into a more efficient scheme. Researchers can then explore the PDFs in a post-hoc tool and define queries to select matching particle data. A space-saving representation for post-hoc analysis of premixed combustion data is presented in Chapter 4 of this thesis, although not as an in-situ algorithm.

II

Analysis and Visualization of the Flame Surface in Turbulent Combustion Simulations

This chapter is based on the publication:
T. Oster, D. J. Lehmann, G. Fru, H. Theisel, and D. Thévenin. "Sparse Representation and Visualization for Direct Numerical Simulation of Premixed Combustion". In: *Computer Graphics Forum* 33.3 (2014), pp. 321–330

4

Sparse Representation and Visualization for Turbulent Premixed Flames

The development and validation of flamelet models is one of the central applications of DNS. As already mentioned in Section 3.2.2, the flamelet assumption is one of the most important simplifications applied for modeling turbulent combustion. It states that a turbulent flame behaves like a collection of strained laminar flames ("flamelets") located side-by-side on the flame surface. This assumption is most often applicable to premixed flames and essentially allows to treat the flame as a 2D manifold.

The analysis of DNS data in the context of flamelet modeling can have two purposes: validation of existing flamelet models, or development of new models. In both cases, it is necessary to extract flamelet data from the DNS. This is often done by sampling the simulation variables along lines orthogonal to the flame surface (see, e.g., [77]). The single flamelets obtained from this are then used for statistical analysis, for example to check model assumptions or discover correlations that can be incorporated into new models.

In this chapter we propose a sparse representation for premixed flames that explicitly encodes flamelets and readily supports flamelet-related analysis

tasks. This representation is significantly smaller than storing the full DNS data, which enables the analysis of a larger number of time steps per simulation run. We also propose a novel visualization based on this data that augments traditional statistical analysis with a visual component. If necessary, full scalar fields on the original grid can be reconstructed from the sparse representation to retain full flexibility for post-processing.

In spirit, the approach we present is similar to existing works that propose computing a smaller representation of the data that can later be used for analysis. Lakshminarasimhan et al. presented the ISABELA compression algorithm [110] and a tool for querying the compressed data [123]. Bremer and Landge [78–80, 120] presented multiple works based on the computation of segmented merge trees that facilitate a post-hoc exploration of thresholds on scalar fields. Ye et al. [121] condensed high-resolution grid data into block-wise probability density functions that can later be explored efficiently.

The rest of this chapter is organized as follows: We first introduce our approach for constructing a sparse representation of the flame in Section 4.1. We then introduce our novel visualization techniques based on this representation in Section 4.2. Section 4.3 describes the reconstruction of full scalar fields and evaluates the compression ratio and reconstruction quality we achieve. Finally, we provide a discussion and conclusion in Section 4.4.

4.1 A Sparse Representation for Premixed Flames

As mentioned before, a flamelet lives on 1D lines orthogonal to the flame surface. If we extract the flame surface and store all flamelets as 1D profiles of the simulation variables orthogonal to this surface, we have a full representation of the flame front. Since the variation of the simulation variables is typically small tangential to the surface, we can store a sparse selection of flamelets and still represent the flame with adequate accuracy. This is the idea behind our sparse representation for premixed flames.

The transformation of raw DNS data into our sparse representation consists of three steps:

1. Seed points on the flame surface.
2. Sample the simulation variables along lines orthogonal to the surface, emanating from the seed points.
3. Approximate the resulting profiles by models, reducing each profile to a set of model parameters.

We extract the flame surface as an isosurface and distribute seed points on it using random sampling adaptive to the surface curvature. 1D profiles are then extracted orthogonal to the flame surface. The profiles are centered

on the surface points and have a limited length to include the maximum flame thickness. In this way, only the data of the flame front is captured, and the large, almost constant areas containing fresh gases and hot products are discarded. We only store the profiles of variables related to chemistry, namely the temperature, heat release and species mass fractions. We ignore the velocity and pressure fields, as they do not exhibit the same structure as the chemical variables and can not be accurately represented by 1D profiles in the flame front only.

Variables are grouped into three classes: *Reactants* have high values outside of the burnt region and low values inside, *products* are the opposite, and *intermediates* occur near the flame surface between unburnt and burnt regions but have low values on either side. This behavior can be modeled with few degrees of freedom, reducing the amount of data even further. The information that has to be stored in the sparse representation consists of the locations and directions of the profile lines, the model parameters for each variable and profile line, and the full flame surface mesh for visualization. We now describe how the lines are seeded, how the profiles are extracted from those lines, and how these profiles are then approximated by simple models.

4.1.1 Strategy for Seeding Profile Lines

Commonly, the flame surface in premixed combustion is defined as the 0.5-isosurface of a *combustion progress variable* [65]. This variable varies between 0 and 1 and is defined as $(T - T_u)/(T_b - T_u)$, where T_u and T_b are the temperature of the burnt and unburnt gases. The choice of the iso-value is very robust for premixed combustion in the flamelet regime. Our experiments show that variations of ± 0.1 lead to isosurfaces with Hausdorff distances less than 2 % of the domain size.

We extract the isosurface using the Computational Geometry Algorithms Library (CGAL) [124]. This yields a mesh with approximately uniformly-spaced vertices and edges of approximately the same length as the voxel size in the original data. Anchor points $\mathbf{p}_i$ for profile extraction are then distributed on this surface. Since the mesh typically has a large number of vertices, we select only some of them as seed points, using a rejection sampling based on local surface curvature. We estimate the principal curvatures κ_1 and κ_2 at each vertex (see, e.g., [125]). The curvatures are then transformed into a seeding density by a logarithmic function:

$$\rho(\kappa_1, \kappa_2) = \frac{1}{q} \ln\left(1 + \sqrt{\kappa_1^2 + \kappa_2^2}\right). \tag{4.1}$$

Here, $q > 1$ steers the seeding density. For each initial vertex, a uniformly distributed random number $r \in [0,1]$ is now generated, and the point is

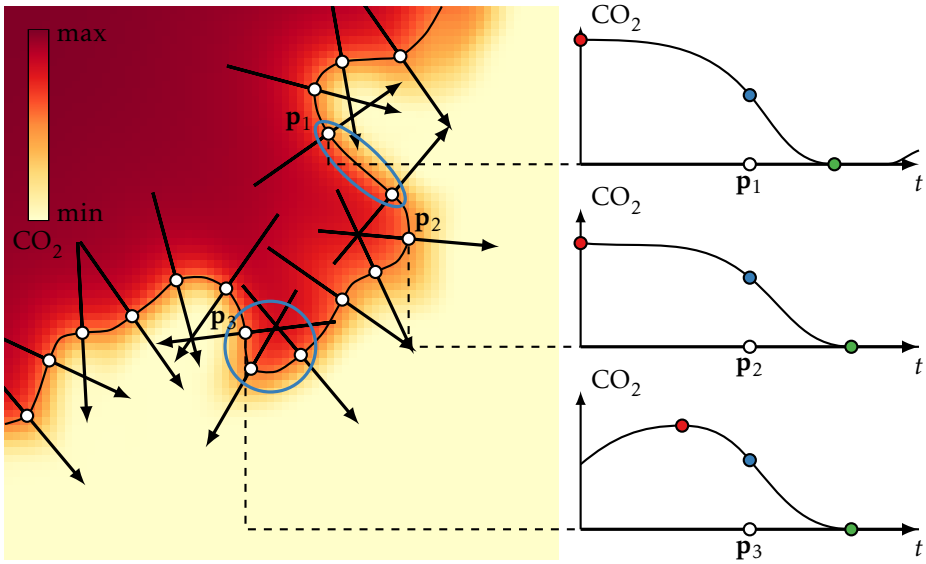

Figure 4.1: Cross section scheme of the flame surface with seeded points $\mathbf{p}$. Profiles are sampled from profile lines $\mathbf{p}(t)$ (arrows) at different locations on the flame surface. The minimum (●), inflection (●) , and maximum (●) of the sigmoid shape are indicated on the profiles.

selected if $\rho < r$. This results in few seed points in areas without curvature, and all vertices being selected in areas where $\rho > 1$. Areas of higher curvature get more seeds than less curved ones (see blue circles in Figure 4.1). Hence, the storage size of the sparse representation depends on the surface shape more than on the resolution of the data. Adjusting q changes the total number of seed points and balances size vs. quality.

4.1.2 Extracting Profile Lines

Once the points $\mathbf{p}_i$ are seeded, profiles of all variables are sampled at regular intervals along lines $\mathbf{p}_i(t) = \mathbf{p}_i + t \cdot \mathbf{n}_i$, with $\mathbf{n}_i$ being the unit surface normals determined from the gradient of the scalar field (Figure 4.1). Due to the high resolution and accuracy of DNS data, trilinear interpolation is sufficient. The length of the line is chosen empirically to cover the maximum flame thickness occurring in the dataset. The resulting profiles are approximated by simple models to further reduce the required storage space.

4.1.3 Model-Based Data Approximation

Although the main advantage of DNS is its high precision due to the lack of modeling assumptions, we use models to describe its outcome in a lower-

dimensional form. These models are sufficient to facilitate the analysis of the scalar fields we intend. Additionally, they reduce the space needed to store the sparse representation.

Reactants, intermediates, and products each have very similar profiles that can be locally approximated by simple models. Reactants and products tend to exhibit profiles with a sigmoid shape, transitioning from a constant high/low value in the unburnt region to a constant low/high value in the burnt region, passing an inflection point in between. This behavior can be expressed by a model based on a sigmoid function. Intermediate species have a maximum near the flame surface, decreasing on both sides. They are approximated by a model based on a Gaussian bell curve.

Small fluctuations that deviate from these models may occur but are not relevant for the general shape. Greater deviations appear if a sample line crosses the flame surface multiple times, which happens if multiple parts of the flame front come close to each other. In such cases, the characteristic behavior occurs multiple times across the profile. To handle this, the instance closest to the anchor point is identified and isolated from the others.

Model for Reactants and Products

Sigmoid shapes are commonly expressed using the logistic function $1/(1+e^{-t})$, which we extend with parameters γ, adjusting the slope at the inflection and a, determining the limits of the function at positive/negative infinity:

$$\mathfrak{s}(t, a, \gamma) = \frac{2a}{1 + e^{(-2\frac{\gamma t}{a})}} - a \,. \tag{4.2}$$

While this function can roughly approximate the profiles of reactants and products, it has too few degrees of freedom to reproduce the different curvatures of the profiles at both sides of the inflection point. For this reason, we use two pieces of this function, joining smoothly at the inflection point (Figure 4.2, top).

$$\mathfrak{S}(t, a_l, a_r, \gamma, x_m, y_m) = \begin{cases} \mathfrak{s}(t, a_l, \gamma) + y_m, & \text{if } t \leq x_m \\ \mathfrak{s}(t, a_r, \gamma) + y_m, & \text{if } t > x_m \,, \end{cases} \tag{4.3}$$

where (x_m, y_m) is the location of the inflection point, γ is the slope at the inflection and $y_m - a_l$ and $y_m + a_r$ are the limits of the function approaching negative and positive infinity.

Model for Intermediate Species

The profiles of intermediates resemble Gaussian bell curves. Since minimum and maximum of these profiles can vary, it is necessary to extend the standard

bell curve with additional parameters y_m, the value at the maximum, and y, the limit at infinity.

$$\mathfrak{g}(t, x_m, y_m, \sigma, y) = (y_m - y)\, e^{-\frac{1}{2}\left(\frac{t-x_m}{\sigma}\right)^2} + y \,. \tag{4.4}$$

Since the profiles tend to have a steeper slope on the unburnt side and some of them do not reach zero on the burnt side, a two-sided model is once again needed to accurately capture this behavior. We join the two bell curves smoothly at their maximum point, resulting in a model with six parameters:

$$\mathfrak{G}(t, x_m, y_m, \sigma_l, y_l, \sigma_r, y_r) = \begin{cases} \mathfrak{g}(t, x_m, y_m, \sigma_l, y_l), & \text{if } t \leq x_m \\ \mathfrak{g}(t, x_m, y_m, \sigma_r, y_r), & \text{if } t > x_m \,, \end{cases} \tag{4.5}$$

where (x_m, y_m) is the location of the maximum, σ_l and σ_r determine the slope of the left and right part of the function and y_l and y_r are the limits of the function approaching negative and positive infinity (Figure 4.2, bottom).

Fitting the Models to the Profiles

We now approximate the profiles by fitting the models. For the sigmoid model, we need the position, value and first derivative at the inflection point as well as the minimum and maximum values of the profile on both sides of the flame front. For the Gaussian model, the location and value of the maximum near the flame front have to be known, as well as the minimum values y_l and y_r. To robustly determine these feature points, we have to account for the two types of deviations that may occur in the profiles as described above: small fluctuations, and the profile entering and leaving the reaction zone multiple times.

To eliminate small fluctuations, we filter the profiles with a Gaussian kernel [126]. The kernel size depends on the size of the fluctuations but can be chosen quite small. For the test data (see Section 4.3) we used a size of 6 voxels, which translates to $\sigma = 1$.

Extrema are found at zero-crossings of the first derivative of the filtered profile. Because of the Gaussian filtering, the extrema might shift from their original positions. We correct this by mapping extrema back to their corresponding positions in the unfiltered profile. Each maximum of the filtered profile is mapped back to the largest maximum of the unfiltered profile within a radius of at most one filter kernel width. Minima of the filtered profiles are mapped to their corresponding positions in the unfiltered profiles using the same approach. During this mapping, we must ensure that the order of extrema along the profile stays the same, and extrema to not switch positions. For finding inflection points, we use the same approach but on the second derivative.

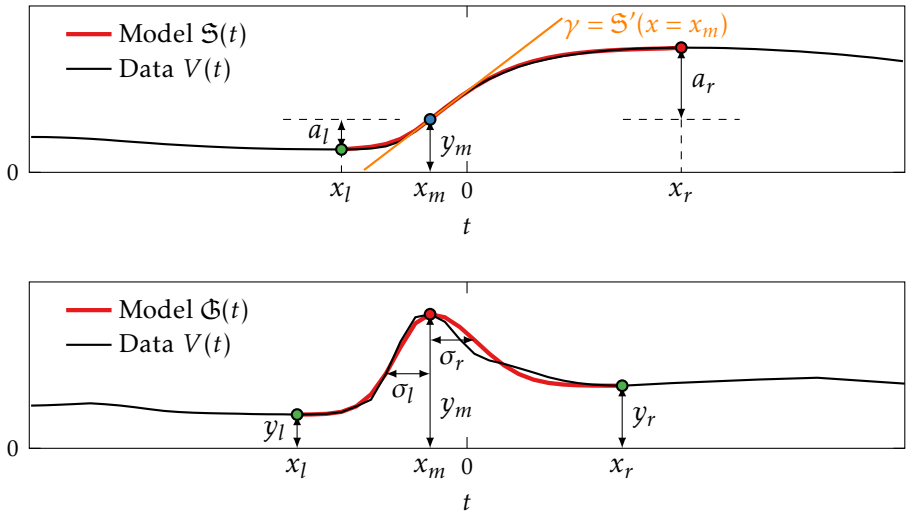

Figure 4.2: Sigmoid model (top) and Gaussian model (bottom) with examples of models fitted to a profile.

Due to the possibility of crossing the flame surface multiple times (see $\mathbf{p}_3$ in Figure 4.1), we can find more feature points on the profile than we need for our models. We have to identify the ones closest to the anchor point. For the sigmoid model, this is the inflection nearest to the center of the profile. The position and value of this point determine x_m and y_m of the sigmoid model, while γ is determined by the first derivative at the inflection. The values of the first minimum and maximum left and right of the inflection (depending on the sign of γ) determine a_l and a_r. The positions of these extrema, x_l and x_r, are the boundaries of the portion of the profile that the model is fitted to. The rest of the profile, possibly containing other crossings of the flame surface, is not considered, as those other crossings are already captured by other profile lines seeded there.

For the Gaussian model, the maximum nearest to the center determines x_m and y_m, while the values of the closest minima to both sides determine x_l and x_r, as well as y_l and y_r. Further extrema are ignored. For both models, if there are no extrema on either side of x_m, the values at the ends of the profiles are chosen instead.

While the feature points on the profile already determine all parameters for the sigmoid model, the values for σ_l and σ_r of the Gaussian model still have to be found. We obtain an initial guess for $\sigma_{l/r}$ by transforming the intervals of the profile between x_l and x_m and between x_m and x_r into the interval $[0,1]$ and regarding them as halves of two symmetric PDFs. The variance of a discrete random variable X with the PDF $p(x)$ and expected value μ is given by: $\mathrm{Var}(X) = \int p(x) \cdot (x-\mu)^2 \, dx$. Since for a normal distribution, the variance is σ^2,

we can directly use this equation on our transformed profiles. We then refine this estimate with a simple optimization scheme such as Newton's method to find the optimal fit.

With this, the original data is now described by the flame surface mesh, the $\mathbf{p}_i$ and $\mathbf{n}_i$ of the profile lines, and the model parameters for each profile line and variable. For the sigmoid model, this is $(a_{li}, a_{ri}, \gamma_i, x_{mi}, y_{mi}, x_{li}, x_{ri})$ for each profile line. For the Gaussian model, the parameters are $(x_{mi}, y_{mi}, \sigma_{li}, y_{li}, \sigma_{ri}, y_{ri}, x_{li}, x_{ri})$. This comprises our sparse representation of the flame, with the flame surface represented by the surface mesh and flamelets represented by the profile lines orthogonal to the surface.

4.2 Construction and Visualization of Feature Surfaces

The flamelets extracted as profiles in the previous step can directly be used by combustion researchers for statistical analysis of the whole flame front. We will not go into detail about this here, as it is outside the scope of this work. In this section, we present a novel visualization based on the feature points extracted on the profiles that augments the statistical approach with a visual analysis component.

The model parameters x_l, x_m and x_r (see Figure 4.2) describe three classes of feature points on the profiles: *minimum* (min), *maximum* (max) and *inflection point* (infl). These feature points span feature surfaces of the respective variables that can have physical or chemical significance. For example, the surfaces of maximum heat release or maximum concentration of a radical are sometimes used as alternative definitions of the flame surface. The feature surfaces of maximum and minimum temperature bound the flame front and indicate the flame thickness. Investigating the shapes and local distances of these surfaces gives insight into the local combustion process and how it is affected by turbulent flow. We now describe the construction of those feature surfaces and their visualization.

4.2.1 Feature Point Construction

The positions of the feature points on the profiles represent intersections with corresponding feature surfaces in the original data. By shifting the anchor points $\mathbf{p}_i$ onto the intersections with a specific feature surface, we transform them to a feature point set representing this surface (see Figure 4.3, left). Considering a profile line $\mathbf{p}_i(t)$ anchored on the flame surface S, the position of a feature point $\mathbf{p}_i(t_f)$ with $f \in \{\text{min}, \text{max}, \text{infl}\}$ is given by

$$\mathbf{p}_i(t_f) = \mathbf{p}_i + t_f \cdot \mathbf{n}_i.$$

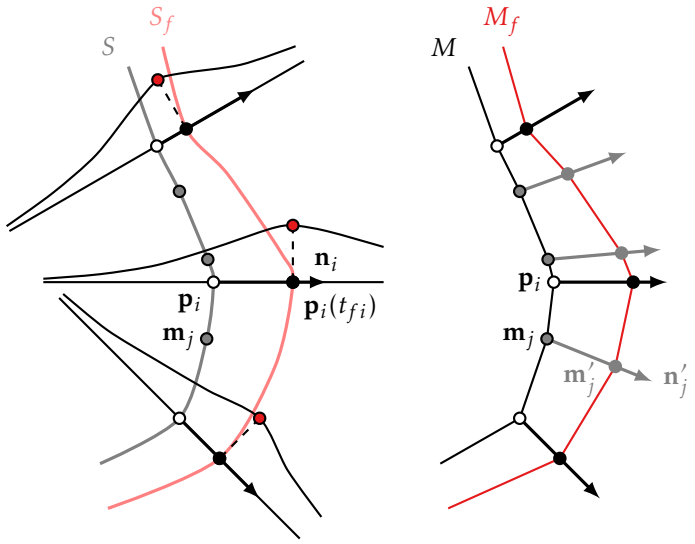

Figure 4.3: Construction of feature points and feature mesh (simplified 2D representation). Left: Construction of feature points (●) by shifting anchor points (○) along their normals. Right: Positions for the other mesh vertices on the feature mesh (●) are obtained via diffusion of the known normals and shift values.

Here, t_f is the shift value for the respective feature surface given by x_l, x_m or x_r of the corresponding profile. For example, the shift values $t_{\max}$ for the surface of maximum heat release are obtained by the x_m values of the heat release profiles. Constructing these feature points is the first step in constructing a feature surface, which we detail in the following section.

4.2.2 Feature Surface Construction

Remember that during the transformation to the sparse representation, an isosurface mesh M representing the flame surface S was extracted. The profile lines were seeded at vertices of this mesh. For each vertex $\mathbf{p}_i$ on the flame surface mesh we therefore know the position of a point on the feature surface S_f by the corresponding shift value t_{fi} and the direction of the profile line $\mathbf{n}_i$ (Figure 4.3, left). The idea is to transform the flame surface mesh M into a feature surface mesh M_f representing the feature surface. The simplest way to implement this transformation would be to move the vertices $\mathbf{m}_j$ of M along the corresponding normal vectors $\mathbf{n}_j$ to a related feature point, given by the shift value t_{fj}. Unfortunately, the values for t_f and $\mathbf{n}$ are not known everywhere on the mesh, but only at vertices where profile lines have been seeded (see Section 4.1.1 and Figure 4.3). Thus, the first step of the transformation is to approximate this information for the other vertices.

We use a diffusion-driven approach to obtain directions $\mathbf{n}'_j$ and shift values t'_{fj} for each mesh vertex from the original $\mathbf{n}_i$ and t_{fi}. For this, we fix the original values at the vertices corresponding to points $\mathbf{p}_i$ and diffuse them over the rest of the mesh until convergence. Different diffusion methods can be used to obtain results of varying smoothness. For simplicity, we use an explicit weighted averaging scheme, iteratively replacing the values at each vertex with the sum of its immediate neighbors, weighted with the neighbor's inverse distance. After this process has finished, we have directions and shift values for each vertex $\mathbf{m}_j$ to obtain approximated feature mesh vertices $\mathbf{m}'_j$ (gray dots in Figure 4.3, right). This process is a preprocessing step that has to be performed only once before visualization and does not further impede performance.

4.2.3 Feature Surface Visualization

With a method to construct feature surfaces from the sparse representation, we can now visualize these surfaces in different ways. Domain experts want to visually examine the feature surfaces, and investigate the differences between feature surfaces of different variables or feature point classes. In the following, we introduce our approach for enabling such an analysis task.

Pairwise Distance Visualization

Given two different feature surfaces, a comparative visualization must highlight differences and similarities. In the context of flamelet analysis, the differences between surfaces along the normal direction is most interesting. We therefore propose a visualization to explore pairwise distances between feature surfaces.

A visualization of distances between feature surfaces of two different variables V and W must allow for quickly identifying regions of small or large distance, as well as the distances' orientation. We achieve this by displaying the local distance between two feature surfaces color-coded on the flame surface mesh. This mesh serves as a neutral and common base for comparison, which is related to both feature surfaces.

As mentioned, corresponding vertices $\mathbf{m}_f^V$ and $\mathbf{m}_f^W$ of two different feature meshes can be obtained from the vertex $\mathbf{m}$ by shifting it by two different values t_f^V and t_f^W along the local normal direction $\mathbf{n}'$. Thus, the distance between the vertices is simply the difference between the two shift values. This distance is computed for each vertex and linearly mapped onto a color map.

We use a color map adapted to our application (see Figure 4.4, bottom right): black for values near zero (the meshes intersect), red to yellow for growing

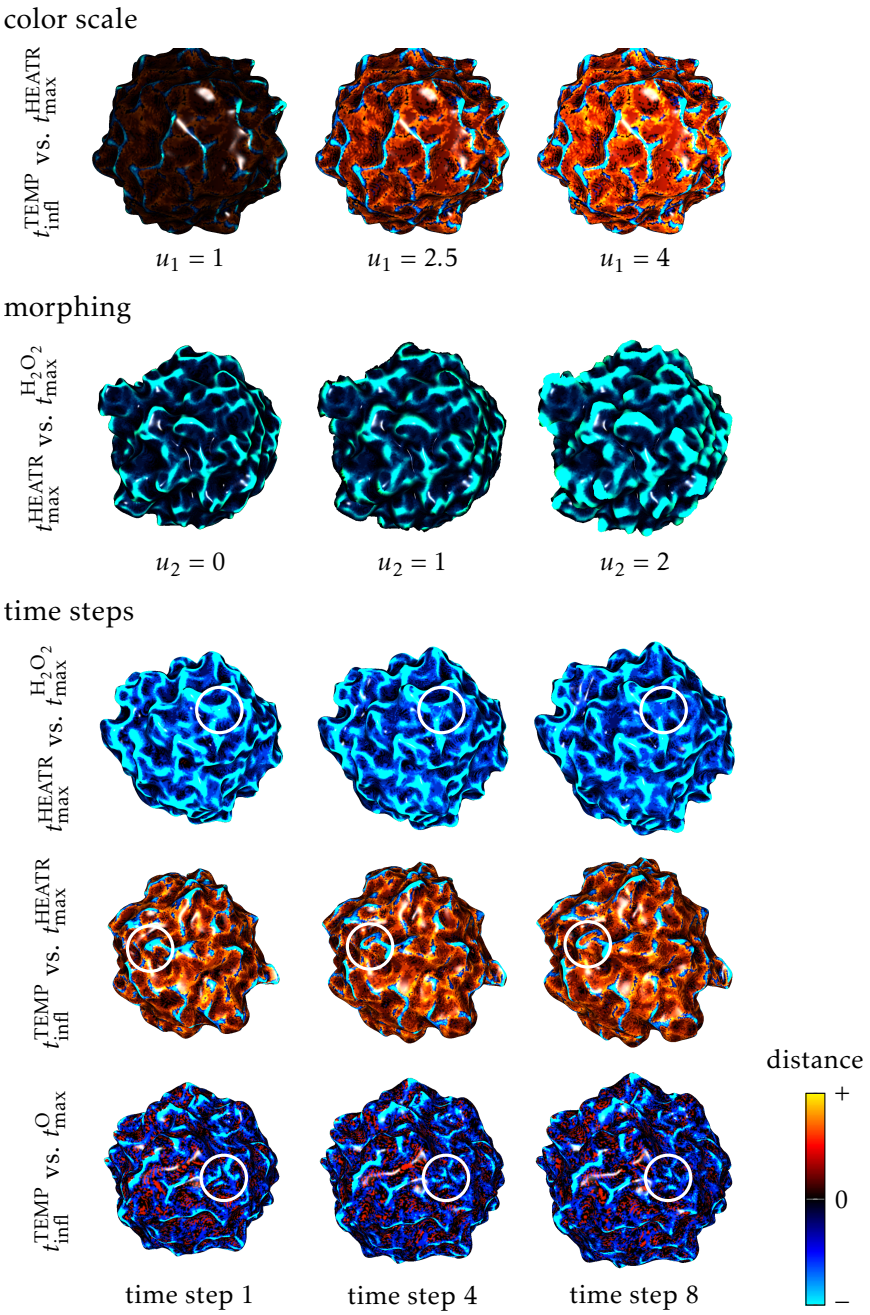

Figure 4.4: Parameters for the visual exploration of feature surfaces. We show dataset Hydrogen. White circles highlight interesting features changing over time.

positive distances (one mesh is outside of the other locally), and blue to cyan for negative distances (the opposite applies).

We introduce parameter u_1 as a scaling factor for adjusting the color contrast and controlling how much of the data is mapped inside the displayed color range and how much is clamped to the maximum/minimum color. This enables a quick visual search for both extreme difference values (by choosing a low value for u_1), or an overview of areas with positive or negative difference values (by choosing a high value for u_1). The effect of varying parameter u_1 is shown in Figure 4.4 (top).

Feature Mesh Visualization

We use standard computer graphics techniques to render the feature surfaces. The distance values are mapped to the mesh as vertex colors, and Phong shading [127] is used to enhance the perception of surface curvature. Larger specular highlights improve the curvature perception but obstruct the view on the mesh color. We therefore let the user control the specular reflectance factor to suit their needs.

To allow for the investigation of the feature meshes' shapes, we provide a user-controlled linear morphing between the flame surface mesh M and the two chosen feature meshes M_f^V and M_f^W. A parameter $u_2 \in [0,2]$ steers the morphing, showing the original flame surface M for $u_2 = 0$, the first feature surface M_f^V for $u_2 = 1$ and M_f^W for $u_2 = 2$ (see Figure 4.4, middle). The morphing itself is trivial. Since the corresponding vertices between all the meshes are known and their topology is identical, they just have to be linearly translated as the value of u_2 changes.

Finally, we enable the user to quickly slide through the different time steps and investigate the temporal behavior of the feature surfaces and their relations (see Figure 4.4, bottom). This allows for a quick interactive visual analysis that would have been impossible to achieve on the original raw simulation data, due to the large number and storage size of time steps.

4.2.4 Evaluation of Diffusion Quality

By computing the shift values t_f only for some of the vertices of M, and obtaining them at the other vertices by diffusion we introduce an error. We quantify this error as the normalized absolute difference between a ground truth and the values at each vertex after the diffusion process. The ground truth is obtained by computing the model parameters t_f for each vertex of M as described in Section 4.1.3. The diffusion error $e_{\text{diff}}^{V,f}$ for variable V and

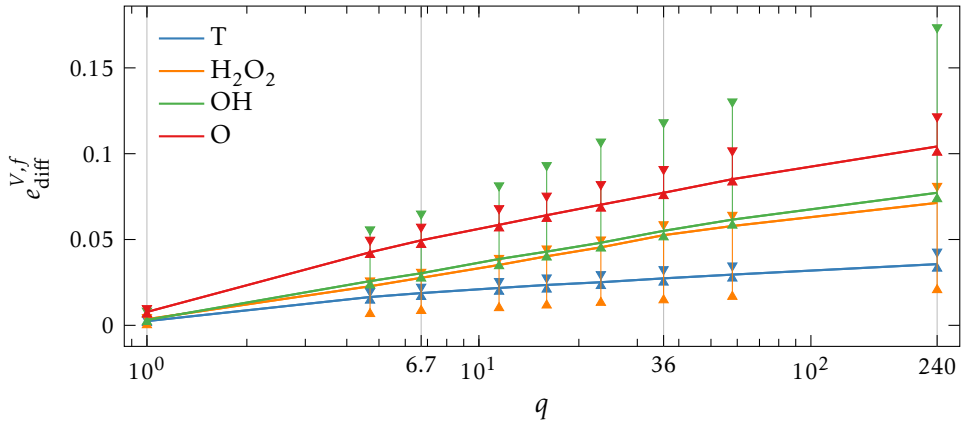

Figure 4.5: Diffusion error (median, max and min) for selected variables of data set Hydrogen. Qualitative results for the values indicated by vertical lines are shown in Figure 4.6.

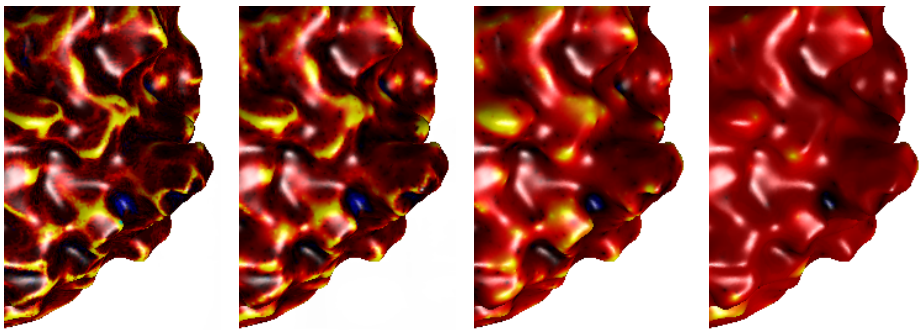

Figure 4.6: Visual comparison of diffusion results for different seeding densities q.

feature f is defined as

$$e^{V,f}_{\text{diff}} = \frac{1}{\max(t^V_f) - \min(t^V_f)} \sum_{\mathbf{m}_i \in M} \left| t^V_{fi} - t'^V_{fi} \right| , \tag{4.6}$$

where t^V_{fi} is the true shift value for variable V and feature f at vertex $\mathbf{m}_i$, t'^V_{fi} is the corresponding value obtained by diffusion, and $\max(t^V_f) - \min(t^V_f)$ is the range of true shift values over all vertices. We computed this error metric for all variables and time steps of data set Hydrogen, using different seeding densities q. We show the results in Figures 4.5 and 4.6.

4.3 Reconstructing Full Scalar Fields

If desired, the scalar fields on the original grid can be reconstructed from the sparse representation by interpolation. We sample the fitted models in regular intervals between the respective x_l and x_r of each line and variable. We then apply standard interpolation methods to this set of points to reconstruct the data on the original grid.

We compared two local interpolation methods for scattered data. The first method is a k approximate nearest neighbors (kANN) [128] interpolation scheme that weighs the values of the k approximate nearest neighbors using Shepard's inverse distance weights [129]. The second interpolation method first generates a tetrahedral mesh from the data points using a Delaunay triangulation. The values inside the mesh cells are then linearly interpolated. This always produces a continuous solution if there are no degenerate mesh cells.

Interpolation methods providing higher smoothness exist. However, these are more computationally expensive and it is not guaranteed that they produce results closer to the original data than the simpler methods.

We evaluated the accuracy of the sparse representation on single time steps of three data sets. Each time step of data sets Syngas I and Syngas II has 200^3 voxels and 13 variables each. Syngas III has 100^3 voxels and 3 variables. All data sets are from DNS computations of turbulent premixed spherical syngas flames. Syngas I contains a flame with strong wrinkles. Syngas II has a flame with smaller wrinkles. Syngas III contains a flame that has been torn into smaller parts by turbulence.

First, we investigate the error from approximating the original data by our models. We computed the average root mean square (RMS) error between the original data of the profiles and the fitted models. The data values in the range x_{li} and x_{ri} on each profile line are considered. We used normalized RMS errors e^V_{fit} in order to make the variables V comparable:

$$e^V_{\text{fit}} = \frac{1}{n \cdot (\max(V) - \min(V))} \sum_{i=1}^{n} \sqrt{\frac{1}{x_{ri} - x_{li}} \sum_{\{j | x_{li} \le x_{ji} \le x_{ri}\}} \left(P^V_{ij} - u^V_{ij}\right)^2} \,. \tag{4.7}$$

Here, u^V_{ij} is the value of the fitted model corresponding to the original profile value P^V_{ij}, while $\{j | x_{li} \le x_{ji} \le x_{ri}\}$ are the indices of all points on the profile between x_{li} and x_{ri}. We computed this error for all possible profile lines in all data sets. As Figure 4.7 shows, the errors are quite low, ranging from 0.1 % to 6.3 % of the respective variable's range.

For investigating the overall error after reconstruction, we computed the reduction ratio c for different q and compared it to the deviation from the

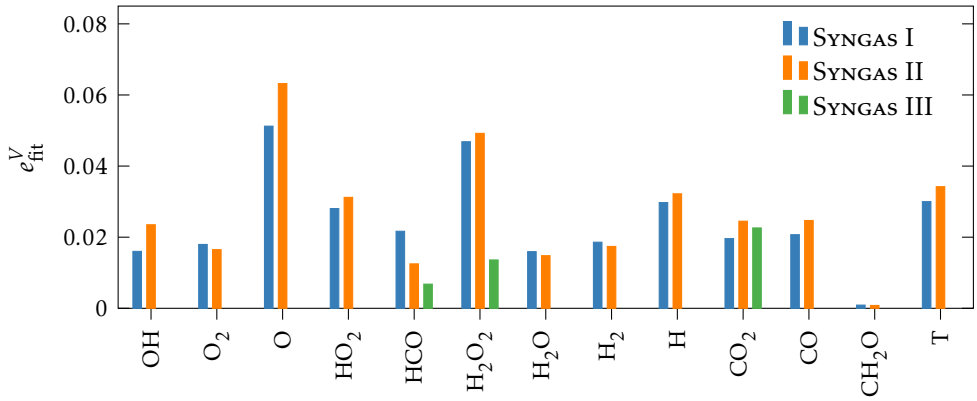

Figure 4.7: Mean RMS fitting error for all variables. Note that not all data sets contain the same variables.

original data after reconstruction. The reduction ratio is defined as the storage space needed for the original data divided by the space needed by the sparse representation. We used a normalized error metric to compute the reconstruction quality:

$$e^V_{\text{reconst}} = \frac{1}{|H| \cdot (\max(V) - \min(V))} \int_{\mathbf{x} \in H} |R(\mathbf{x}) - V(\mathbf{x})| \, \mathrm{d}\mathbf{x} \,, \tag{4.8}$$

where V is the original scalar field, R is the reconstructed data, H is the convex hull of all points used to reconstruct the data, and $|H|$ is the volume of H. The range of values of variable V is described by $\max(V) - \min(V)$.

We compared the results of linear and kANN interpolation for reconstruction. Our experiments show that for the kANN interpolation a combination of five nearest neighbors weighted with a Shepard weighting function using an exponent of 20 gave the lowest errors. Therefore, we illustrate the results for these parameters only. We also compared our results to the error introduced by naively downsampling the data to the same storage size needed by the sparse representation. This is currently still the most common way of reducing the size of DNS data. For comparison with a dedicated compression algorithm, we used the well-established 3D set partitioning in hierarchical trees (SPIHT) algorithm [130] implemented in the QccPack library [131] on our data.

Figure 4.8 shows the reduction ratio c vs. the reconstruction error e^V_{reconst} for all tested methods for selected variables. Please note that for Syngas II, higher reduction ratios are achieved than for Syngas I, due to the flame in the former being relatively smaller. We also show qualitative results of the reconstruction in Figure 4.9. For small reduction ratios, the downsampling approach performs better, because it does not introduce errors due to model assumptions.

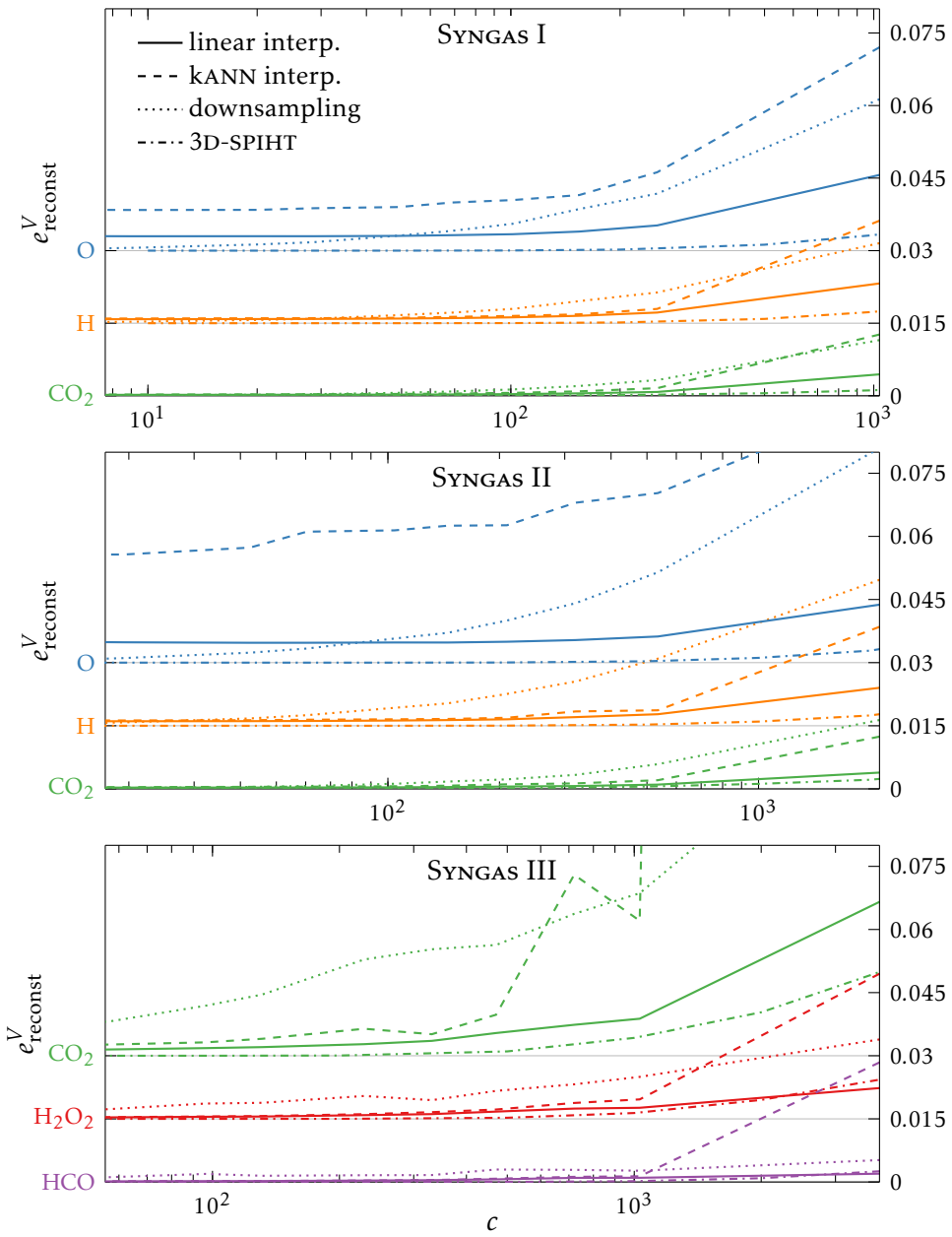

Figure 4.8: Error vs. reduction ratio for selected variables. The plots corresponding to each variable are shifted by a constant increment. The horizontal lines signify the zero-levels of the respective plots.

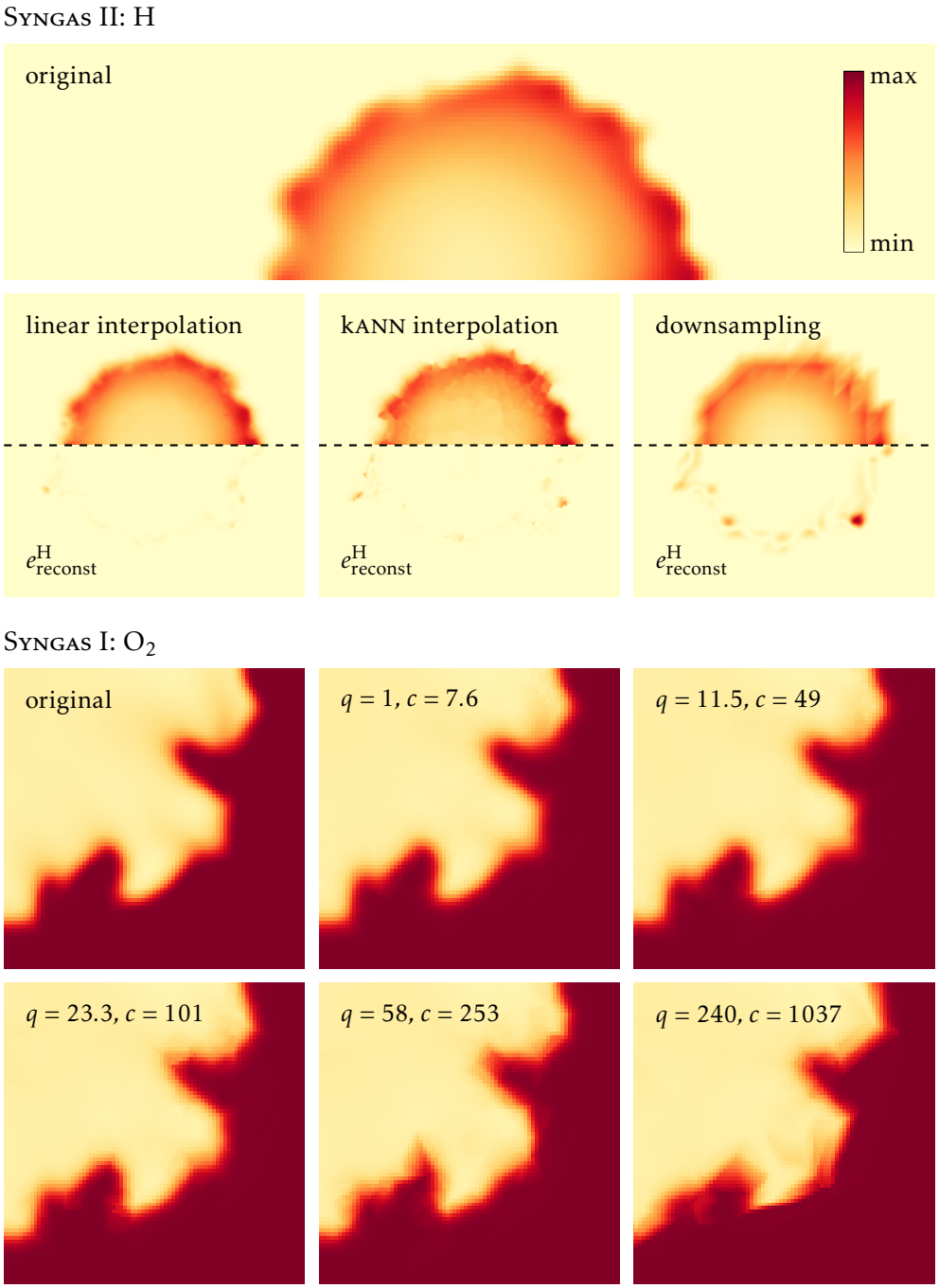

Figure 4.9: Top: Comparison of reconstruction results for H of Syngas II with $q = 36$ (ca. 2500 profile lines, $c = 321$). Bottom: Reconstruction results for O_2 of Syngas I using linear interpolation with different sample densities (detail view).

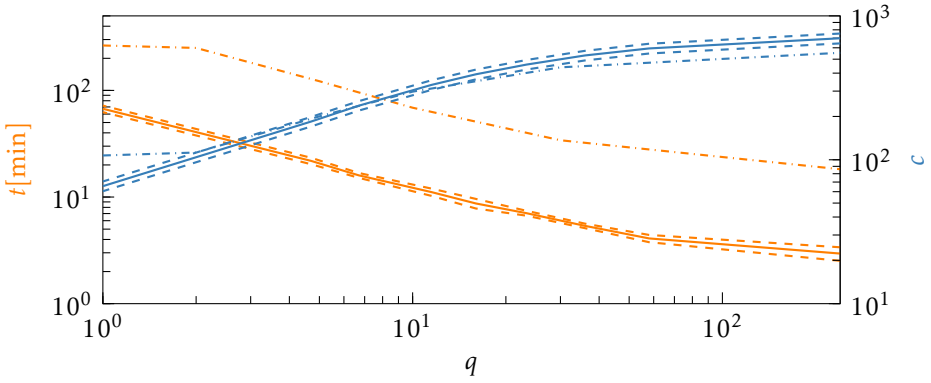

Figure 4.10: Reduction ratio c and computation time t for different seeding densities q. Plot shows mean (—) and standard deviation (- -) over eight time steps of data set Hydrogen and values for the synthetic data set (-·-) .

For higher reduction ratios, which are needed in practice, our sparse representation always performs significantly better. It is also apparent that linear interpolation performs better than kANN in almost all cases. As a dedicated compression algorithm, 3D-SPIHT naturally achieves better reconstruction quality than our approach. It is however necessary to decompress the data back to its full size before any analysis can be carried out, while our sparse representation can directly be used for flamelet analysis and visualization of feature surfaces without prior reconstruction.

To further illustrate the data reduction performance of our approach, we tested it on eight time steps of the data set Hydrogen. This data set is from a turbulent premixed spherical hydrogen flame and has a resolution of 400^3 voxels and 11 variables. This is a typical size for our cooperation partner. One time step amounts to about 5 GB of data. The whole simulation has tens of thousands of time steps. Even storing a fraction of them quickly results in terabytes of data. We selected eight time steps from a late (most complex) stage of the simulation. Choosing $q = 5$, which retains good accuracy for reconstruction, we reduced them to about 30 MB each. This means only 0.6 % of the original storage space is required.

For a scalability test, we generated a synthetic data set with 900^3 voxels and 11 variables. This is a typical size for a modern large-scale DNS run. We created noise based on an isotropic turbulence frequency spectrum [132]. This noise was added to a low frequency component to emulate larger flame structures with smaller surface perturbations. Thresholding produces a surface on which average profiles of the different variables were superimposed. Domain experts confirmed the similarity of the result to real simulations, making it suitable for scalability tests.

Since each profile line has to be processed separately, the run time of the algorithm is approximately linear in the number of seed points, and thus depends indirectly on the flame surface area and structure. Figure 4.10 shows run times and compression ratios for the synthetic data set and Hydrogen. Data set size and flame surface area of the synthetic data set are one order of magnitude higher than that of Hydrogen. This results in a run time which is also one order of magnitude higher, confirming the scalability of our approach.

4.4 Discussion

We introduced a sparse representation for DNS data of premixed combustion that is tailored to flamelet-related analysis tasks. The sparse representation enables storing the simulation results with far smaller space requirements. The space requirement is mainly dependent on the complexity of the flame shape, not on the size or resolution of the data, i.e., the less complex the flame shape, the less profile lines need to be seeded for an accurate representation. Via fitting of models using feature points, the sparse representation directly captures important characteristics of the scalar fields that can be analyzed in different ways without the need for data reconstruction. Feature surfaces derived from these models can directly be visualized and facilitate the visual analysis of the data.

Apart from the feature surfaces, further characteristics might be extracted from the sparse representation, such as gradient fields and their topology. Note that our approach is specifically tailored to DNS data, but can be used for other kinds of multi-field data where changes are only located in narrow-band regions.

Despite its many advantages, our approach has some limitations. Because we assume combustion in the flamelet regime, our approach has limited applicability in scenarios where the flamelet assumption is not or only partially fulfilled. However, there is still a large number of practical scenarios for premixed combustion where the flamelet assumption is valid.

Our sparse representation does not capture pressure or velocity information, as these do not only vary in narrow-band regions and they do not necessarily conform to the models we use to approximate variable profiles. Many flamelet-related analysis tasks can do without this data. If it is needed for flamelet analysis, one could store the raw unmodeled data for these variables along the profile lines. It would however not be possible to reconstruct this data on the original grid with sufficient accuracy. As an alternative, these variables could be stored in the original resolution. Since pressure and velocity only represent a small fraction of the data in combustion simulations, this will still result in a significant reduction in storage size.

Our approach also has some technical limitations. Seldom outliers lead to locally large distances between feature surface and flame surface, which is visible as spots in the visualization. These are however rare and indicate areas of unusual behavior on the flame surface, which also provides meaningful information. Finally, relying on a random process to seed the profile lines might produce insufficient numbers of samples in some regions. This might be avoided by using a deterministic seeding approach and is left for future work.

Four experts in combustion DNS examined our approach, two of which were also partly involved in its development. They stated that the extraction of feature surfaces especially for variables that have a maximum near the flame surface is a welcome addition to their set of analysis tools. Such surfaces are of particular interest in the comparison of combustion processes between laminar and turbulent flows. Established practice is to approximate them by isosurfaces of other variables, which were assumed to be close to the desired feature surface based on the conditions in laminar flow. The possibility of directly comparing feature surfaces with our approach opens new possibilities in the investigation of the effects of turbulence on combustion.

Another application proposed by the experts is the comparison of experimental and simulation research. In experiments, the flame surface is often determined by easily measurable quantities, while more precise definitions are used in simulation research. Our feature surfaces enable comparison of these definitions and deriving models to make experiments and simulations more comparable.

This comparison of different flame surface definitions is also important when comparing different simulations. Since there is no universally agreed-upon definition of the flame surface, different researchers often use different definitions, which could be quantitatively compared with our method.

The sparse data representation, apart from much-needed space savings, opens possibilities of statistical analysis of the relations of feature surfaces, which could be used to improve combustion models for RANS or LES methods.

Our approach is currently implemented as a post-processing step. By transforming the original data into the sparse representation, disk space usage is reduced considerably. At the same time, the data is also brought into a form better suited for flamelet analysis and feature surface visualization. The implementation of the approach as an in-situ process brings additional technical problems to be solved. One of these problems is tracking the flame surface over time, such that the correspondence between flamelets at different time steps can be maintained. An approach to this surface tracking problem is presented in the next chapter.

This chapter is based on the publication:
T. Oster, A. Abdelsamie, M. Motejat, T. Gerrits, C. Rössl, D. Thévenin, and H. Theisel. "On-The-Fly Tracking of Flame Surfaces for the Visual Analysis of Combustion Processes". In: *Computer Graphics Forum* 37.6 (2018), pp. 358–369

5

In-Situ Tracking of the Flame Surface

Unsteady effects and their incorporation into turbulent combustion models are still an area of active research in the combustion community. In order to observe and analyze such effects, researchers need access to data with high temporal resolution. Storing raw DNS data in this resolution is only practical for very short time intervals. To observe behavior that stretches over longer time spans, or whose occurrence cannot be easily predicted, in-situ processing becomes an absolute necessity.

The flame surface is central to many combustion models and -studies. Quantities such as surface speed, -stretch, -curvature, -area and their relation to the structure and behavior of the flame are frequently discussed in the literature. In order to observe the detailed unsteady behavior of the flame surface, we need an algorithm to track it during the simulation. This chapter presents an approach for such an in-situ surface tracking. Our goal is to capture the shape of the surface, the history of individual surface points, and the local tangential deformation of the surface over extended periods of time. This also allows studies of the history of single surface points over extended time periods, such as described by Yeung et al. [72] and Sripakagorn et al. [73], to be extended to the complete flame surface. Additionally, we extend the

notion of instantaneous surface stretch (such as described by Poinsot and Veynante [65]) to the tangential deformation of the surface over arbitrary time intervals.

Our algorithm must overcome the following challenges:

1. The massively parallel simulation, distributes its domain across a large number of processes. The surface tracking must be similarly parallelizable.
2. Performing analysis in-situ during a simulation means that at any point in time only data for the current time step is available in memory and there is no way of going back in time to retrieve previous information.
3. The surface is expected to undergo significant deformation over time, making an adaptive refinement and coarsening necessary.
4. The tangential deformation must be reconstructed for areas of the surface that have been refined and/or coarsened multiple times over an arbitrary time interval.

Item 1 precludes the use of a mesh or space partition data structure with neighborhood information, as the global nature of operations in such a structure is not well suited for a massively parallel algorithm. Instead, we represent the surface as a number of independent micro-patches consisting of a central point and four ghost particles measuring the local surface deformation. This is described in Section 5.3.1. Items 2 and 3 mean that we need to refine the micro-patches adaptively before they are significantly distorted. We therefore introduce a way of monitoring the distortion of a patch and split it before the distortion becomes too large in Section 5.3.2. This method of tracking and refining a surface without explicit neighbor information is a major contribution of this work. Based on the behavior of the micro-patches over time, we introduce the computation of the tangential deformation gradient (Item 4) for arbitrary time intervals in Section 5.3.3.

5.1 Related Work

The evolution of simulation variables on the path of single points on the flame surface has been used in multiple works in combustion literature [72–74]. In these works only a relatively small number of points on the surface are tracked and they do not consider the relative tangential movement of points. Stretching of a flow restricted to a surface has been investigated similarly to our approach by Garth et al. [134].

Tracking different kinds of features has been the subject of numerous works in the field of flow visualization. A lot of research deals with tracking volumetric features over time [83, 118, 135–138], while some methods focus on point-

[139, 140] or line-type features [80]. Most of these methods are designed for datasets that fit into the main memory of a consumer-grade computer, although some are explicitly designed to work in distributed-memory settings [81].

Surface extraction and tracking is another large field of research. In the context of this work, particle-based isosurface extraction methods, like the one proposed by Crossno et al. [141], as well as methods for tracking evolving surfaces over time [142–144] are relevant. Of particular interest for our application is the work by Camp et al. [145], which deals with stream surface integration in a distributed-memory environment.

To the best of our knowledge, there is no existing approach that solves the problem of tracking a time surface in a large distributed-memory simulation while maintaining temporal correspondence between surface points.

5.2 Mathematical Basis

In this section, we give a brief introduction into the mathematical basis of the problems we want to solve. First, we derive the equation for the movement of the flame surface over time. Then, we introduce the tangential deformation gradient, which describes the distortion an infinitesimally small section of the surface experiences in a certain time interval.

5.2.1 Tracking the Flame Surface

We view the flame surface as an implicit surface $s(\mathbf{x}, t) = \Gamma$. This scalar function is transported by the fluid velocity $\mathbf{v}(\mathbf{x}, t)$ and influenced by diffusion and chemical reaction processes. A point $\mathbf{x}$ with velocity $\mathbf{u}$ tracking the surface has a zero Lagrangian derivative, i.e., the value of s at the point does not change over time. Therefore

$$\frac{\mathrm{D}s}{\mathrm{D}t} = \frac{\partial s}{\partial t} + \nabla s \cdot \mathbf{u} = 0\,.$$

This constrains the component of $\mathbf{u}$ that is normal to the surface. The tangential component of $\mathbf{u}$ is constrained by the fluid velocity through

$$\|\mathbf{u} - \mathbf{v}\|^2 \to \min\,.$$

Combining both constraints using Lagrange multipliers, the solution for the velocity of a point on the implicit surface is given by

$$\mathbf{u} = \mathbf{v} - \frac{\frac{\partial s}{\partial t} + \nabla s \cdot \mathbf{v}}{\|\nabla s\|^2} \nabla s\,. \tag{5.1}$$

In combustion literature, this equation is expressed as

$$\begin{aligned} \mathbf{u} &= \mathbf{v} + s_\mathrm{d}\,\mathbf{n}\,, \\ \text{with} \quad s_\mathrm{d} &= \frac{\partial s}{\partial t}\frac{1}{\|\nabla s\|} + \frac{\nabla s}{\|\nabla s\|}\mathbf{v} \quad \text{and} \quad \mathbf{n} = -\frac{\nabla s}{\|\nabla s\|}\,. \end{aligned} \tag{5.2}$$

Here, s_d is the speed of flame propagation normal to the surface and relative to the fluid velocity and $\mathbf{n}$ is the unit surface normal.

5.2.2 Tangential Deformation of an Implicit Surface in a Flow

In the previous section, we showed how a point on the surface moves over time. We now derive the tangential deformation gradient, which encodes the relative linear movement of surface points in an infinitesimal neighborhood.

We consider the starting position of a point on the surface at some time t_0. W.l.o.g. we assume that this point is at the origin $\mathbf{0}$. Let $\chi(\mathbf{x}, t)$ be the mapping function that maps a point $\mathbf{x}$ at t_0 to its position at time $t > t_0$ after moving with the surface. W.l.o.g. we assume that $\chi(\mathbf{0}, t) = \mathbf{0}$. The spatial deformation gradient $\mathbf{F} = \nabla\chi$, which encodes the behavior of a point $\mathbf{x}$ in an infinitesimally small neighborhood around $\mathbf{0}$, can then be expressed as

$$\mathbf{F}\,\mathbf{x} = \chi(\mathbf{x}, t) \quad \text{for } \mathbf{x} \to \mathbf{0}\,.$$

We are only interested in the tangential part of this deformation, i.e., the behavior in a tangential coordinate system that moves and rotates with the surface. In order to isolate the tangential component, we first need to eliminate this rotation. A (right) polar decomposition $\mathbf{F} = \mathbf{U}\mathbf{P}$ separates the rotational part $\mathbf{U}$ from the rest of the deformation. The tangential component is now obtained by projecting $\mathbf{P} = \mathbf{U}^\mathrm{T}\mathbf{F}$ into the local tangent space at time t_0. Let $\mathbf{B}$ be a matrix with orthonormal basis vectors of this tangent space as its columns. Then the tangential deformation gradient $\hat{\mathbf{F}}$ is

$$\hat{\mathbf{F}} = \mathbf{B}^\mathrm{T}\mathbf{U}^\mathrm{T}\mathbf{F}\,\mathbf{B}\,. \tag{5.3}$$

Note that $\mathbf{B} \in \mathbb{R}^{3\times 2}$; and thus $\hat{\mathbf{F}} \in \mathbb{R}^{2\times 2}$. See Figure 5.1 for a visual interpretation of this transformation. $\hat{\mathbf{F}}$ now encodes the linear behavior of a point $\hat{\mathbf{x}}$ in an infinitesimally small neighborhood around $\hat{\mathbf{0}}$ in tangent space:

$$\hat{\mathbf{F}}\cdot\hat{\mathbf{x}} = \mathbf{B}^\mathrm{T}\,\chi(\mathbf{B}\,\hat{\mathbf{x}}, t) \quad \text{for } \hat{\mathbf{x}} \to \hat{\mathbf{0}}\,.$$

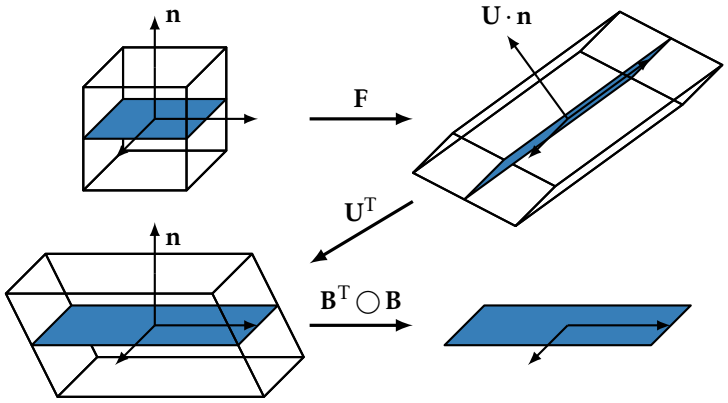

Figure 5.1: Extracting the tangential component of the deformation gradient. A local neighborhood is transformed by the deformation gradient **F**. In the process, the local coordinate system is rotated by **U**, which must be reverted before isolating the tangential component of **F** by multiplying with the tangential basis **B** from both sides.

5.3 Discretization

Our objective is an algorithm for tracking the flame surface in-situ during a simulation run. It must produce the paths of single points on the surface over time, as well as the tangential deformation of the surface for arbitrary time intervals. Additionally, the algorithm needs to be highly parallelizable in order to work well in the environment of a massively parallel simulation.

A naive approach might be performing an isosurface extraction in each time step of the simulation. However, subsequent isosurfaces do not contain information about surface point correspondence between time steps. This correspondence is necessary if we want to provide point paths and compute the surface deformation.

Alternatively, one could advect the vertices of an explicit surface mesh, adaptively remeshing it as the surface deforms over time. Due to the global nature of such remeshing operations, this is infeasible in a massively parallel environment, where irregular communication between neighboring processors is known to be a major source of bottlenecks.

We therefore choose to represent the surface as a cloud of micro-patches consisting of a central point and four *ghost particles* (see Figure 5.2, left). The ghost particles sample the shape and deformation of the micro-patch in the local neighborhood around the central point. Each group of points is tracked completely independently. We monitor the deviation of the ghost particles from the tangent plane at the central point and the deviation of their relative movement from linear behavior. Once this deviation exceeds a user-defined threshold, we split the patch into three independent new patches to ensure

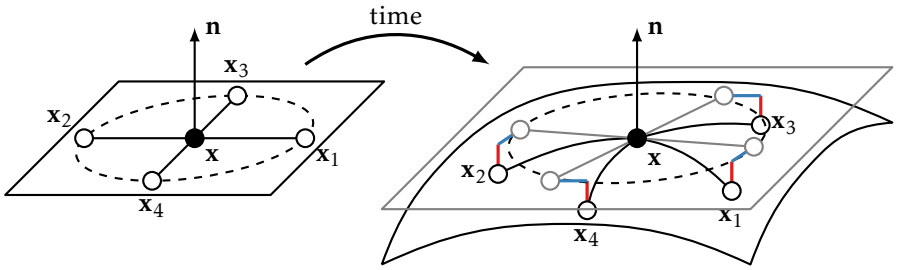

Figure 5.2: Ghost particles (○) are initialized in an orthogonal configuration around the central point (●) in the tangent plane. While integrating the points over time, they deviate from a planar, linear behavior. The difference between the real positions and the nearest linear configuration (○) in the tangent plane is measured by the normal error $e_\perp$ (—) and the tangential error $e_\parallel$ (—).

sufficient sampling. To avoid oversampling, we merge patches when they become too small, e.g., because the surface flattens over time or tangential movement bunches up many patches in a small area. The tangential surface deformation is then reconstructed from the relative behavior of the points in a group between split/merge events.

5.3.1 Micro-Patches for Surface Tracking

A micro-patch is represented by a group of five points: A center and four *ghost particles*. On initialization, the points are arranged in an orthogonal cross shape in the tangent plane defined by the surface normal $\mathbf{n}$ at the central point $\mathbf{x}$ (see Figure 5.2). In this way, they sample the shape and deformation in all directions as they follow the surface over time.

We describe the configuration (e.g., the state excluding the absolute position) of a micro-patch at time t as a matrix $\mathbf{C}(t) \in \mathbb{R}^{3\times 5}$ consisting of the surface normal $\mathbf{n}(t)$ at the central point as well as the relative positions of the ghost particles $\mathbf{x}_i(t)$, $i \in \{1,\dots,4\}$.

$$\mathbf{C}(t) = \begin{pmatrix} \mathbf{n}(t) & \mathbf{x}_1(t)-\mathbf{x}(t) & \cdots & \mathbf{x}_4(t)-\mathbf{x}(t) \end{pmatrix}. \tag{5.4}$$

In the following, we omit the dependence of $\mathbf{C}$, $\mathbf{n}$, and $\mathbf{x}_i$ on the time t and we assume that $\mathbf{x} = \mathbf{0}$ wherever the meaning is clear from context.

For the initial configuration at some time t_0, $\mathbf{C}$ can be exactly represented by a unit base configuration $\mathbf{C}_0$ being transformed by a linear transformation $\mathbf{T}$, i.e.,

$$\mathbf{C}(t_0) = \mathbf{T}\,\mathbf{C}_0, \quad \text{with} \quad \mathbf{C}_0 = \begin{pmatrix} \mathbf{e}_3 & \mathbf{e}_1 & -\mathbf{e}_1 & \mathbf{e}_2 & -\mathbf{e}_2 \end{pmatrix},$$

where $\mathbf{e}_i$ are the unit vectors in the i-th coordinate direction. As the points track the surface over time, their relative position will change. Their relation

to $\mathbf{C}_0$ might no longer be linear. Now, $\mathbf{T}$ is the linear transformation that best approximates the mapping between $\mathbf{C}_0$ and $\mathbf{C}$, i.e., the solution to the least squares problem

$$e^2 = \left\| \mathbf{C}_0^{\mathrm{T}} \mathbf{T}^{\mathrm{T}} - \mathbf{C}^{\mathrm{T}} \right\|_{\mathrm{F}}^2 \to \min .$$

Here, $\|\cdot\|_{\mathrm{F}}$ denotes the Frobenius norm.

The residual error e measures the deviation of the real mapping between $\mathbf{C}_0$ and $\mathbf{C}$ from linear behavior. However, its scale is dependent on the size of the micro-patch, i.e., the magnitude of the last four columns of $\mathbf{C}$, and it weights those columns differently from the normal in the first column. To get a more meaningful error, we normalize these columns by their average magnitude and obtain a modified transformation $\mathbf{T}_{\mathrm{n}}$ and error e_{n} by

$$\begin{aligned} e_{\mathrm{n}}^2 &= \left\| \mathbf{C}_0^{\mathrm{T}} \mathbf{T}_{\mathrm{n}}^{\mathrm{T}} - \mathbf{C}_{\mathrm{n}}^{\mathrm{T}} \right\|_{\mathrm{F}}^2 \to \min , \\ \text{with} \quad \mathbf{C}_n &= \begin{pmatrix} \mathbf{n} & \mathbf{x}_1/c & \cdots & \mathbf{x}_4/c \end{pmatrix} \quad \text{and} \quad c = \frac{1}{4} \sum_i^4 \|\mathbf{x}_i\| . \end{aligned} \tag{5.5}$$

We separate this error into normal and tangential components

$$e_\perp = \left\| \mathbf{n}\,\mathbf{n}^{\mathrm{T}} (\mathbf{T}_{\mathrm{n}} \mathbf{C}_0 - \mathbf{C}_{\mathrm{n}}) \right\|_{\mathrm{F}} \tag{5.6}$$

$$e_{\shortparallel} = \left\| (\mathbf{I} - \mathbf{n}\,\mathbf{n}^{\mathrm{T}}) (\mathbf{T}_{\mathrm{n}} \mathbf{C}_0 - \mathbf{C}_{\mathrm{n}}) \right\|_{\mathrm{F}} . \tag{5.7}$$

$e_\perp$ is a measure for the deviation of the ghost particles $\mathbf{x}_i$ from the tangent plane. As such, it measures the local surface curvature in relation to the patch size. $e_{\shortparallel}$ measures how much the tangential deformation of the ghost particles deviates from linear behavior. As the micro-patch changes over time, we use these errors to decide when to split or merge patches to ensure sufficient sampling of the surface geometry and tangential deformation.

5.3.2 Splitting and Merging Surface Patches

We split a micro-patch into three new patches when one of the following occurs:

- The error $e_\perp$ exceeds a threshold $r_\perp$
- The error $e_{\shortparallel}$ exceeds a threshold $r_{\shortparallel}$
- The major axis exceeds a threshold r_{size}

The first two are to ensure a sufficient sampling, the third is to ensure a maximum patch size even on flat parts of the surface. In a parallel distributed simulation, the size of a patch is limited by the size of the block of the simulation domain it is contained in.

When splitting a micro-patch, we need to decide the positions of the points forming the new patches. In this context, it helps to think of the micro-patch as an ellipse being defined by the positions of the ghost particles around the central point. On initialization, the ghost particles are always placed along the principal axes of the ellipse. When the patch is transformed over time, the principal axes will generally not stay aligned with the ghost particles. The major and minor axes of the current configuration can be reconstructed from the tangential component $\hat{\mathbf{T}}$ of $\mathbf{T}$, which is obtained similar to (5.3):

$$\hat{\mathbf{T}} = (\mathbf{e}_1 \quad \mathbf{e}_2)^{\mathrm{T}} \mathbf{U}^{\mathrm{T}} \mathbf{T} (\mathbf{e}_1 \quad \mathbf{e}_2) , \tag{5.8}$$

where $\mathbf{U}^{\mathrm{T}}$ is the rotation obtained from the polar decomposition $\mathbf{T} = \mathbf{UP}$. The directions and extents of the principal axes of the micro-patch are encoded in the singular value decomposition

$$\hat{\mathbf{T}} = \hat{\mathbf{V}} \hat{\Sigma} \hat{\mathbf{W}}^{\mathrm{T}} .$$

The singular values $\sigma_{1,2}$ on the diagonal of $\hat{\Sigma}$ are the extents of the ellipse, while the rows of $\hat{\mathbf{V}}$ are the directions of the major and minor axis in tangent space. The directions in 3D space are the columns of $\mathbf{U}(\mathbf{e}_1 \quad \mathbf{e}_2)\hat{\mathbf{V}}^{\mathrm{T}}$.

When splitting the micro-patch, we want the new patches to cover the whole area captured by the old patch without overlapping, to prevent over- or undersampling as patches are split multiple times. We also want to leave the central point in place, so we can track its path for as long as possible. Consequently, we split the patch along its major axis into three identical new ones (see Figure 5.3). The points of the new patches are again arranged in an orthogonal cross shape with the axes parallel to the axes of the old patch. The length of the new first axis is exactly 1/3 of the length of the old major axis. The new second axis is as long as the old minor axis. The new points are then projected onto the surface along the normal at the central point. From this point on, they are treated independently again.

The new central patch continues tracking the behavior of the neighborhood around the same point as the old patch. We therefore consider it to be the same entity, but with its ghost particles reset to a more numerically stable configuration. In contrast, we consider the new outer patches to be entirely new entities. Consequently, if we say that a patch has been split or merged multiple times, we mean that it has been the central patch in a number of these operations.

We consider a micro-patch for merging when

- both errors $e_\perp$ and $e_\parallel$ decrease below lower thresholds $l_\perp$ and $l_\parallel$, respectively.
- the minor axis decreases below a threshold l_{size}.

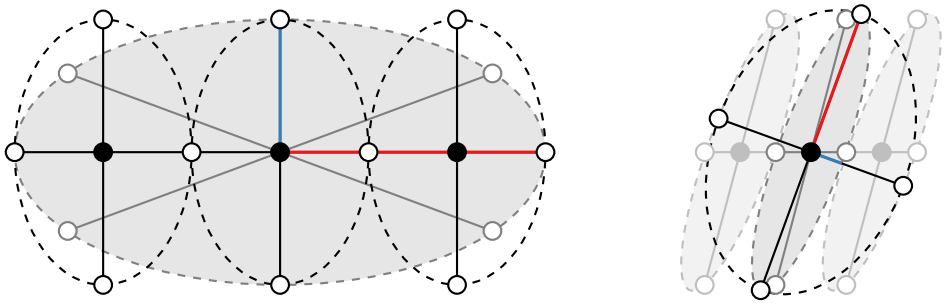

Figure 5.3: Splitting and merging micro-patches. Left: An old micro-patch (⬭) is split up along its major axis (—) into three new identical patches (⬭). Right: An old micro-patch (⬭) is extended to three times its size along its minor axis (—). Neighboring patches (⬭) that were split off from the central one at some earlier time are deleted instead.

Because the micro-patches are completely independent, we cannot explicitly merge multiple patches. We therefore give every patch a counter, which is set to 0 when it is first initialized. When a patch is split, the counter of the central patch is incremented while the new outer patches get a counter of 0. When the conditions for a merge are met, the behavior of a patch depends on its counter. If the counter is 0, the patch is simply deleted. Otherwise, the counter is decremented and the patch is extended by a factor of 3 along its minor axis. Assuming the behavior of neighboring patches is similar, this effectively means that the new extended patch now covers the area of the neighboring patches, which were deleted. This assumption does not generally hold for all parts of the surface. We therefore typically choose $l_\perp$, $l_\parallel$, and l_{size} to be very small. In this case, points will only be deleted in areas of extreme tangential compression, where removing very small patches will only result in small errors, and in areas of very simple deformation, where the assumption is unlikely to be violated.

The central point of the extended patch stays in place, while the ghost particles are again placed along the principal axes of the old patch, but with the minor axis extended. In this way, we ensure that the surface is not oversampled where it is not necessary and that no infeasibly small patches are tracked, e.g., in areas of compressive tangential behavior. Patches that were originally initialized at the start of the simulation are never deleted, in order to prevent holes from forming. Since the surface at the start of a simulation is generally very simple and can be represented by a relatively small number of patches, this is not an issue in practice.

5.3.3 Reconstructing Tangential Surface Deformation

Let $\hat{\mathbf{F}}_{t_\mathrm{s}}^{t_\mathrm{e}}$ be the tangential deformation gradient for a time interval $[t_\mathrm{s}, t_\mathrm{e}]$ at the central point $\mathbf{x}(t_\mathrm{e})$. To reconstruct it we need the corresponding deformation gradient $\mathbf{F}_{t_\mathrm{s}}^{t_\mathrm{e}}$, which describes the relative change in position of points in a small neighborhood between times t_s and t_e. The behavior of the ghost particles of a micro-patch over time contains exactly this information. However, because all ghost particles are located on the surface, they do not contain information about the behavior of $\mathbf{F}$ in normal direction. Fortunately, we only need the change in orientation of the surface normal to reconstruct $\hat{\mathbf{F}}$, as any other information is discarded when projecting into the tangent plane. It is therefore sufficient to determine a proxy transformation $\mathbf{E}_{t_\mathrm{s}}^{t_\mathrm{e}}$, which maps the micro-patch configuration $\mathbf{C}(t_\mathrm{s})$ to $\mathbf{C}(t_\mathrm{e})$, and reconstruct $\hat{\mathbf{F}}$ via

$$\hat{\mathbf{F}}_{t_\mathrm{s}}^{t_\mathrm{e}} = \mathbf{B}^\mathrm{T}\,\mathbf{U}^\mathrm{T}\,\mathbf{E}_{t_\mathrm{s}}^{t_\mathrm{e}}\,\mathbf{B}\,, \tag{5.9}$$

with $\mathbf{B}$ the basis vectors of the tangent plane at t_s and $\mathbf{U}$ the rotation matrix from the polar decomposition of $\mathbf{E}$.

Let us first assume the micro-patch was not split or merged in the time interval, i.e., its ghost particles were not reset. Because the real mapping between $\mathbf{C}(t_\mathrm{s})$ and $\mathbf{C}(t_\mathrm{e})$ will generally not be linear, $\mathbf{E}_{t_\mathrm{s}}^{t_\mathrm{e}}$ is the solution to a least squares problem very similar to (5.5). To make the solution independent of the scale of the micro-patch, we scale the last four columns of both $\mathbf{C}(t_\mathrm{s})$ and $\mathbf{C}(t_\mathrm{e})$ by the average norm of these columns in $\mathbf{C}(t_\mathrm{e})$. The system for reconstructing $\mathbf{E}_{t_\mathrm{s}}^{t_\mathrm{e}}$ is then

$$\left\| \mathbf{C}_\mathrm{n}^\mathrm{T}(t_\mathrm{s})\,{\mathbf{E}_{t_\mathrm{s}}^{t_\mathrm{e}}}^\mathrm{T} - \mathbf{C}_\mathrm{n}^\mathrm{T}(t_\mathrm{e}) \right\|_\mathrm{F}^2 \to \min\,, \tag{5.10}$$

with $\mathbf{C}_\mathrm{n}(t_\mathrm{s})$ and $\mathbf{C}_\mathrm{n}(t_\mathrm{e})$ being the configurations with their last columns scaled.

If the ghost particles were reset one or more times during the time interval, $\mathbf{E}_{t_\mathrm{s}}^{t_\mathrm{e}}$ is the concatenation of multiple transformations:

$$\mathbf{E}_{t_\mathrm{s}}^{t_\mathrm{e}} = \mathbf{E}_{t_n}^{t_\mathrm{e}}\,\mathbf{E}_{t_{n-1}}^{t_n}\,\cdots\,\mathbf{E}_{t_1}^{t_2}\,\mathbf{E}_{t_\mathrm{s}}^{t_1}\,, \tag{5.11}$$

where t_i are the discrete times between t_s and t_e when the ghost particles were reset in the course of a split or merge operation. Each sub-interval transformation is reconstructed from the new configuration of the micro-patch just after a split/merge and the old configuration just before the next one (see Figure 5.4).

If we want to compute the tangential deformation a micro-patch at time t_e has experienced since the start time t_s, it is possible that this patch has not existed for the complete time interval, i.e., it was created at some time $t_k > t_\mathrm{s}$

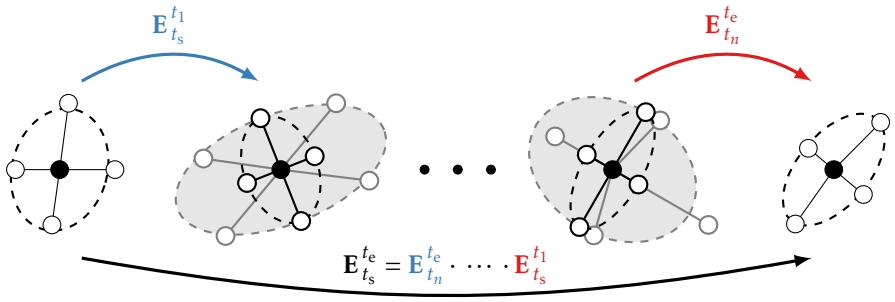

Figure 5.4: Reconstructing the deformation for an arbitrary time interval by concatenating the deformations between split and merge events of the patch.

during a split operation. In this case, we estimate the transformation between t_s and t_k from its parent patch by weighting the contributions of each ghost particle differently depending on their distance from the center of the new patch. The details of this algorithm are presented in Appendix A. With this, we can reconstruct the tangential deformation gradient of any micro-patch for an arbitrary time interval.

5.3.4 Initialization

At the start of the simulation, the initial surface has to be seeded with patches. Most simulations start with a very simple surface, such as a plane or sphere, for which this operation is trivial. If the starting surface is more complex, any existing isosurface meshing algorithm that produces near-equilateral triangles can be used, with patches initialized to cover the area of the 1-ring of each vertex. When initializing the micro-patches, care has to be taken not to leave any holes, which might grow larger over time if the surface expands. We ensure this by overlapping the initial patches, which will increase the number of necessary patches by a constant factor, but is easy to implement. More elaborate approaches which avoid covering the surface with more than one layer of patches are conceivable.

5.4 Implementation

Our algorithm is implemented as an extension to the DINO direct numerical simulation code [69]. The simulation domain is a rectangular box which is distributed to multiple processors by a block decomposition along two of its three axes (pencil decomposition). This unusual decomposition is required because of the pressure solver, which operates in Fourier space and needs the

complete domain in memory in one dimension. The communication between the processors uses MPI [146].

Each tracked point gets a unique ID at its creation. In each process, all points located in its domain block, as well as all points in a halo region around the block, are kept in memory. This halo region is large enough to contain all ghost particles of any patch whose central point is inside the processor's block.

After each simulation step, the fluid velocity $\mathbf{v}$ and derivatives of the scalar variable s defining the flame surface are interpolated from the simulation grid at all surface points to compute their velocity $\mathbf{u}$. Because each simulation block only holds data for the grid cells inside its boundaries, the values for the points in the halo region have to be obtained by communicating with all neighbor processes. The points are then advanced by one simulation time step. For additional stability, we then perform a few steps of a Newton scheme to move them back onto the isosurface we are tracking. As the surface points move during the simulation, they will often migrate across processor boundaries. This is automatically handled by our implementation as part of the information exchange with neighboring processors when updating the points in the halo region.

Since our algorithm is running in lockstep with the simulation, only data for the current time step is available in memory at any time and the time step is controlled by the simulation. This means that without generating additional memory overhead, only first order schemes can be used to integrate the surface velocity $\mathbf{u}$. In our implementation, we therefore use an Euler scheme to advance the positions of the points between simulation steps.

The simulation time step in DNS is generally dominated by the chemistry time scale, which is much smaller than the fluid velocity time scale. Since the surface velocity typically follows the fluid velocity closely, and an Euler scheme is adequate to track the flame surface almost everywhere, as points move only a fraction of the size of a grid cell in each time step. Exceptions to this rule only occur at very sharp creases in the surface, which can occur when two parts of the surface fold into each other, possibly leading to changes in topology. The high surface velocities occurring here can lead to points migrating far away from the surface in a single time step. Since the surface at these locations is contracting rapidly anyway, we simply delete any micro-patches that end up further away from the surface than one half of their previous advection step, provided the distance is at least $1/5$ the size of a grid cell. In our tests, this strategy did not impact the accuracy of the surface tracking in any significant way.

Depending on the values of the errors $e_\perp$ and $e_{\parallel}$, the micro-patches are now split or merged and any patches that partially crossed a non-periodic outflow boundary of the simulation domain are removed. At this point, control is

given back to the simulation, which performs the next iteration.

The deformation gradients can either be computed in-situ or as a post-processing step. In the first case, the time intervals for the deformation need to be specified before the simulation starts. Each patch then remembers its configuration **C** at the time it was last reset and the transformations **E** from the start of each time interval up until this last reset time. In the second case, the point positions and normals of all micro-patches are stored to disk for each time step. The deformation gradients can then be reconstructed for arbitrary time intervals, but at the cost of increased hard disk storage demand.

5.5 Results

We tested our algorithm on an analytic test function that is designed to resemble the behavior of a vortex, and on two real-world simulations. The simulations were carried out on Phase 1 of the SuperMUC Petascale System of the Leibnitz Supercomputing Centre in Garching, Germany. Each node of the system has two 8-core Intel Xeon E5 processors with a clock frequency of 2.7 GHz and 32 GB of shared memory. The nodes are connected via Infiniband FDR10.

In the following sections, we evaluate the accuracy of our method on the analytic test function, and show our results for the two simulation cases.

5.5.1 Analytical Test Function

To evaluate the accuracy of the tangential deformation gradient obtained by our method, we designed an analytic test function. It imitates the behavior of a vortex in the shear layer between two gas streams. The test function s is defined as

$$s(x,y,z,t) = \begin{cases} z\cos A - (x-\frac{1}{2})\sin A & \text{for } \sqrt{(x-\frac{1}{2})^2+z^2} < \frac{1}{2}, \\ z & \text{else}, \end{cases} \tag{5.12}$$

with

$$A(x,y,z,t) = 2\pi t \sin\left(\frac{\pi}{2}\left(1+\sqrt{4x^2-4x+4z^2+1}\right)\right)^2 \sin(\pi y)^2. \tag{5.13}$$

The isosurface $s = 0$ coincides with the xy plane at $t = 0$. As t increases, the surface is curled up around the center at $(x,y,z) = (1/2, 1/2, 0)$. The underlying velocity field of a vortex can not be easily expressed as an analytic function. We therefore assume a fluid velocity of $\mathbf{v} = \mathbf{0}$ for our tests.

We observe the function in the domain $x \in [0,1]$, $y \in [0,1]$, $z \in [-1/2, 1/2]$, $t \in [0,2]$ resolved on a $72\times72\times72$ regular grid and a time step of $\Delta t = 5\times10^{-3}$.

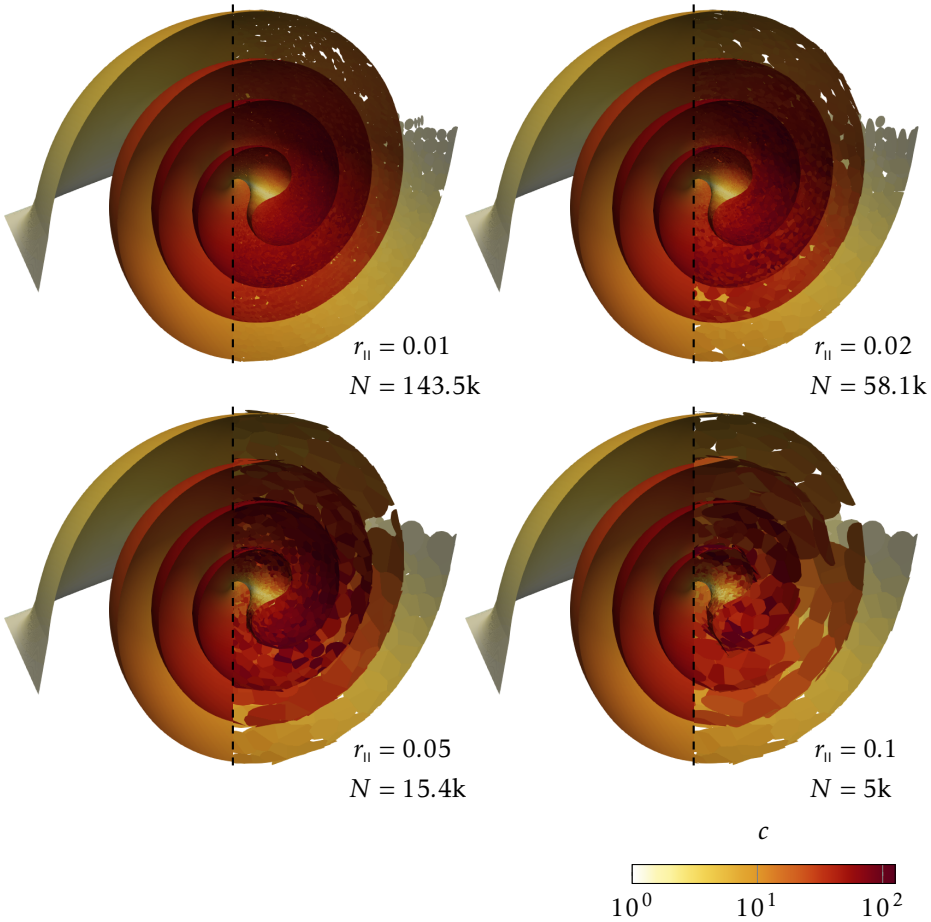

Figure 5.5: The number of patches N and the stretch coefficient c of the tangential deformation gradient $\hat{\mathbf{F}}_0^2$ for the analytic test function, using $r_{\perp} = 0.5$ and varying values for $r_{\parallel}$. Each micro-patch is represented by an ellipse scaled and aligned according to its principal axes and orthogonal to its surface normal. The left half of each image shows the ground truth. Note that c is displayed on a logarithmic color scale.

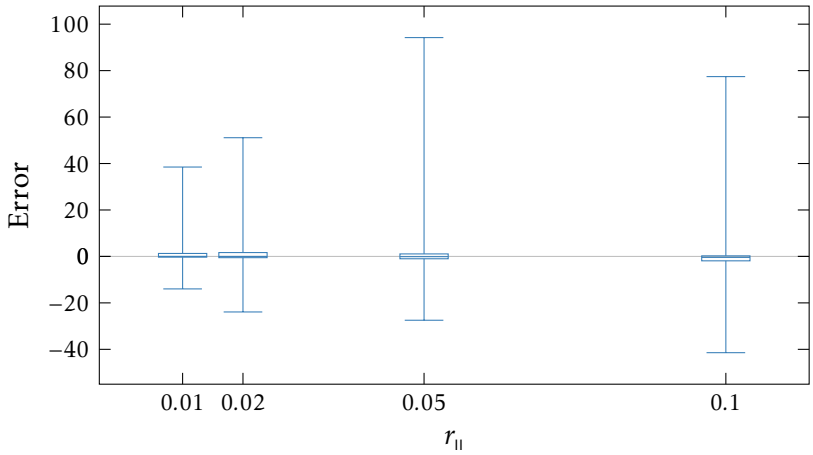

Figure 5.6: Box plot of error distributions for the analytical test function. We show the minimum, lower quartile, median, upper quartile and maximum errors for different values of $r_{\shortparallel}$. See Figure 5.5 for a visual comparison of the results. Please note that this plot was erroneous in the original publication [133] and has been corrected here.

This means that in the investigated time span the vortex makes exactly two full turns, resolved in 4000 time steps. This closely resembles the lifetime of a vortex and temporal resolution observed in a real simulation setting.

The velocity $\mathbf{u}$ of the surface $s = 0$ can be expressed as an analytic function. This enables us to obtain highly accurate ground truth data for the tangential deformation gradient $\hat{\mathbf{F}}$. The largest singular value of $\hat{\mathbf{F}}$ signifies the largest stretching in any tangential direction experienced by the local neighborhood of a point on the surface, which we call the stretching coefficient c with

$$c = \sigma_{\max}(\hat{\mathbf{F}}) = \sqrt{\lambda_{\max}(\hat{\mathbf{F}}^{\mathrm{T}}\hat{\mathbf{F}})}. \tag{5.14}$$

This is similar to the measure used when computing the FTLE for measuring separation in flow fields [147]. The tangential deformation gradient also contains the relative change in surface area $a = \det(\hat{\mathbf{F}})$, which is equivalent to the instantaneous surface stretch that is used in many combustion models, integrated over time.

In all tests, we use a normal error threshold of $r_{\perp} = 0.5$, which is rather coarse. By doing this, we limit its influence on the accuracy of the results for the tangential deformation. We show the distribution of differences to the ground truth solution for different tangential error thresholds $r_{\shortparallel}$ in Figure 5.6. For the sake of brevity, we only show the results for the complete interval $t \in [0, 2]$, which will naturally show the largest errors.

As shown in Figures 5.5 and 5.6, the accuracy of our method is strongly dependent on $r_{\shortparallel}$. For large values of $r_{\shortparallel}$, the resulting deformation does not

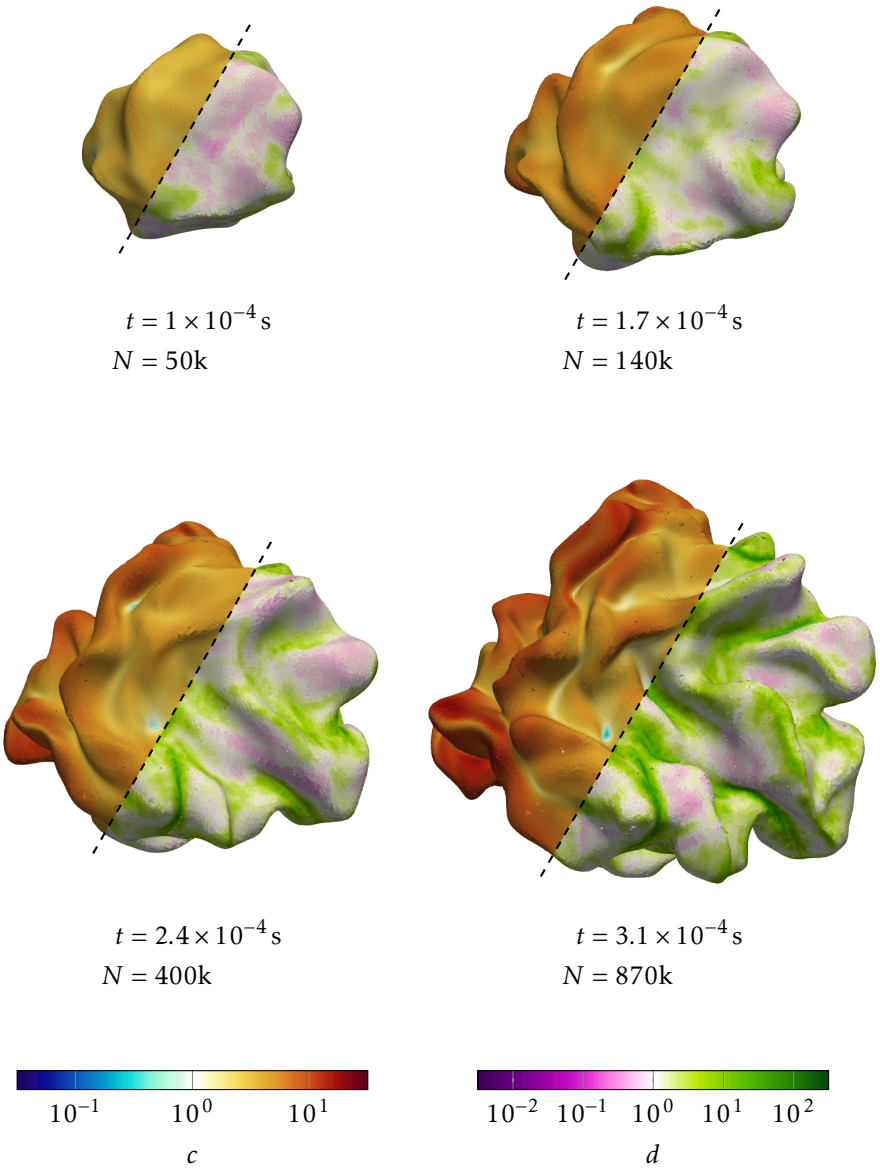

Figure 5.7: Results of our algorithm for the Premixed Flame case. We show the stretch coefficient c and density factor d on logarithmic scales. We computed c for an interval of $\Delta t = 1.7 \times 10^{-4}$ s and $\varepsilon_{\text{II}} = 0.02$. We also show the number of patches N in each time step.

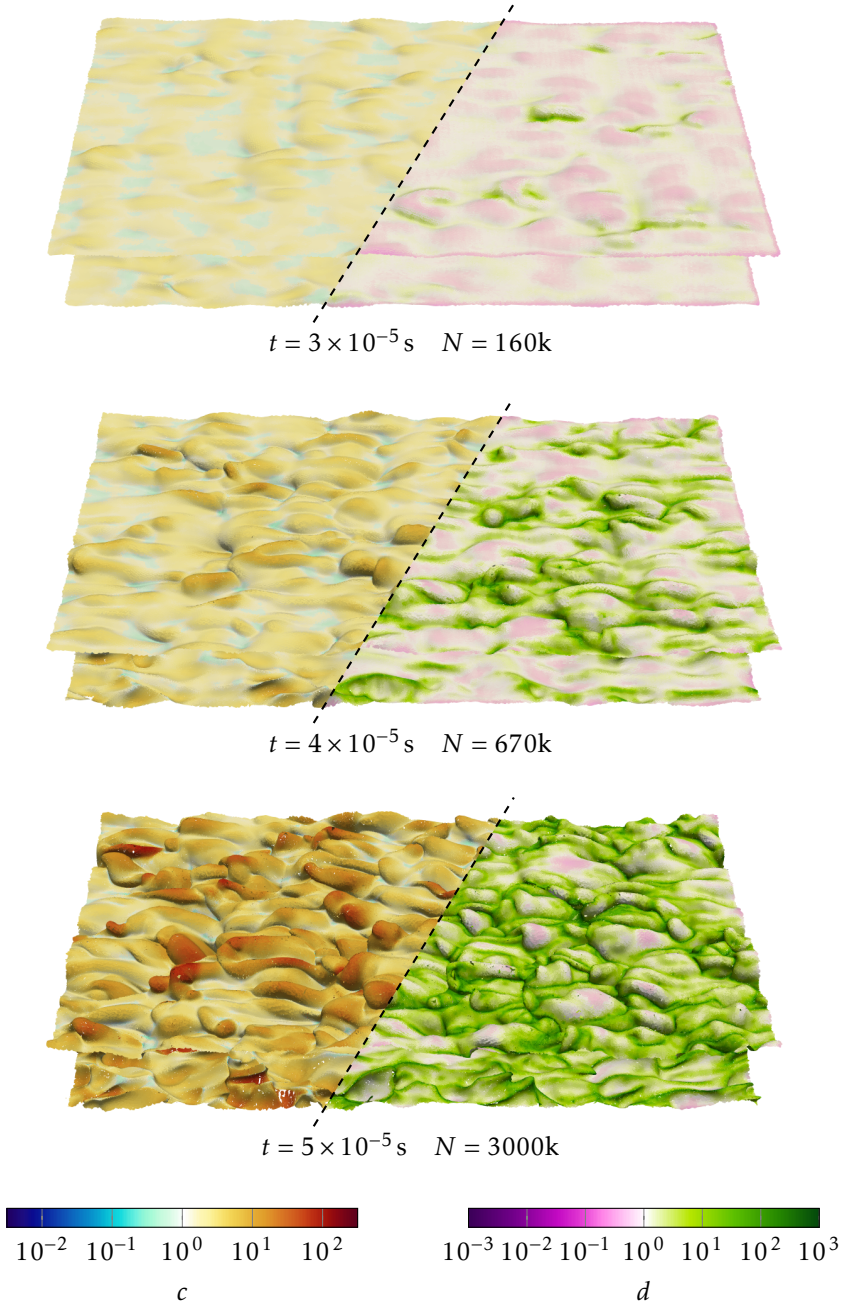

Figure 5.8: Results of our algorithm for the Temporal Diffusion Jet case. We show the stretch coefficient c and density factor d on logarithmic scales. We computed the stretch coefficient since the start of the simulation using $\varepsilon_{\text{II}} = 0.04$. The number of patches N is also shown for each time step.

only contain unreasonably large errors, but the surface also exhibits some holes. Splitting too late means that there is potentially a lot of non-uniform stretching across the patch that is not accounted for by the resulting child patches. This shows that observing only the normal error $e_\perp$ is not sufficient to guarantee a good sampling of the surface, even if one is not interested in an accurate result for the tangential deformation gradient. As the error threshold decreases, the holes close and the stretching value approaches the ground truth solution. For the lowest error threshold value $r_\parallel = 0.01$, the number of patches at the end is still an order of magnitude smaller than the number of grid cells the function is resolved in.

5.5.2 Premixed Flame in a Box

We applied our algorithm to the combustion of a premixed hydrogen-air mixture in a periodic box resolved on a $512 \times 512 \times 512$ regular grid. A high-temperature hot spot is placed in the middle of the domain, which is initialized with a flow field exhibiting isotropic turbulence. After the gas mixture is ignited, a flame front travels through the domain, consuming the fresh gas mixture and leaving burned products behind. As the flame expands, it is deformed by the turbulent flow, which is in turn influenced by the temperature and pressure changes induced by the chemical reaction. The flame surface of a premixed flame is often defined as an isosurface of the temperature between the unburnt and burnt gases, which is what we track here. Simulations of this type are relevant in safety research, where the influence of the turbulence intensity on the ignition probability of the flame is studied.

The simulation ran for about 90 h using 1024 parallel processors and performing about 21 000 iterations. We chose $r_\perp = 0.1$ and $r_\parallel = 0.02$ as well as $l_\perp = l_\parallel = 10^{-4}$. The thresholds r_{size} and l_{size} were chosen such that a surface patch is always smaller than the smallest extent of a block of the simulation domain, and larger than $1/16$ of the size of a grid cell.

Figure 5.7 shows the results of our algorithm for four snapshots of the simulation. The simulation starts with a spherical configuration represented by about 9000 micro-patches. In the first shown time step, the surface has started to expand and wrinkle from its initial configuration. At this point, the surface is represented by about 50 000 micro-patches. Up until the last time step at $t = 3.0 \times 10^{-4}$ s, the number of patches increases to about 870 000. Despite the change in surface area and complexity, we are able to accurately track the surface without any neighbor information between micro-patches. We obtain smooth results that show which regions of the surface have expanded or contracted significantly.

We also show the density factor d. This number is a local approximation of

the ratio of number of patches per surface area at a given time to the initial seeding density. A high concentration of patches occurs in areas with high surface curvature, as well as in areas where the surface deformation has a large nonlinear component.

The ratio of the total surface area of all micro-patches to the true area of the flame surface remains stable around 3.3 for the whole simulation. This shows that our strategy for splitting and merging micro-patches is successful in maintaining a consistent and stable coverage of the surface. The number of splits performed per iteration fluctuates around 0.015 % of the total number of micro-patches at all times. The number of merge events per iteration is almost zero up until the time between the second and third snapshot shown in Figure 5.7, at iteration 15 000, where it starts to increase, stabilizing at around half the number of splits.

5.5.3 Temporal Diffusion Jet Flame

The second simulation is a temporal diffusion syngas jet flame resolved on a $1024 \times 1025 \times 512$ regular grid. The domain is initialized with a turbulent fuel layer in the center, surrounded by a quiescent air co-flow. The fuel layer and co-flow move in opposite directions, resulting in strong shear forces that form vortices where the two gases meet. The flame surface we track here is the isosurface of the stoichiometric mixture fraction, which is the ratio between fuel and oxidizer that theoretically results in perfect consumption of the fuel with no excess of the oxidizer.

The simulation ran for about 23 h using 4096 parallel processors. In this time, the simulation performed about 3600 iterations. For this case, we chose $r_\perp = 0.2$ and $r_\parallel = 0.04$ as well as $l_\perp = l_\parallel = 10^{-4}$. The thresholds are chosen higher than in the premixed case, because here we are interested in the deformation of the surface over a smaller time interval of 4×10^{-5} s. The thresholds r_{size} and l_{size} were chosen with the same method employed for the premixed case.

We show our results for three different snapshots in Figure 5.8. The case starts with a planar surface on both sides of the fuel jet, which is moving from left to right in the images. It is initially seeded with about 50 000 micro-patches. This number stays almost constant for the first 2000 iterations, as the surface wrinkles only a very small amount. After this, the surface starts deforming rapidly. As a result, the number of micro-patches increases rapidly up to about 3 million at the end of the simulation.

The surface deformation is characterized by lots of small fuel pockets intruding into the opposing air flow. These pockets are round and smooth towards the outside, while they form a lot of sharp angles towards the inside. Both the significant expansion of surface area at the tip of these pockets as well

as the sharp angles towards the fuel side are handled well by our algorithm. The density factor d seen on the right side shows that a high patch density mainly occurs in the sharp creases between the pockets where the surface contracts, while the expanding tips are represented with a lower number of patches.

The ratio of the total area of all micro-patches to the true surface area remains constant here as well, but on a slightly lower level of 3.15. The number of splits is almost zero until the surface starts to deform significantly around iteration 2000. At this point, it starts to increase from 0.002 % to about 0.003 % of the total number of micro-patches until the end of the simulation. This number is much smaller than for the premixed case, because here, the surface area does not change so dramatically over time. The number of merges is initially much smaller than the number of splits, but increases steadily throughout the simulation until it is at about 75 % towards the end. This is due to the contracting behavior near the sharp angles in between fuel pockets, which causes lots of small patches to accumulate in a small area.

5.5.4 Performance

Figure 5.9 shows the performance over the course of the simulation, as well as the number of micro-patches existing at each time step. We conducted multiple experiments with different choices for the subdivision threshold $\varepsilon_{\parallel}$. For the sake of brevity, we only discuss the results for the lowest choice of $\varepsilon_{\parallel}$, which causes the highest overhead in computing time and shows the most accurate results.

At the start of the Premixed Flame case, our implementation causes an overhead in computing time of about 300 ms per iteration, or about 3 % of the base computing time. As the flame surface expands, the overhead gradually increases, until it is at about 12.5 s or 130 % at the end of the simulation. For the Temporal Diffusion Jet, the overhead starts out at about 1200 ms per iteration, or about 10 % of the base computing time. It remains fairly constant for the first 1500 iterations, as the surface does not change much during that time. The surface then starts to wrinkle significantly, resulting in a sharp rise of computing time, reaching up to 42 s, or about 350 % of the base time.

The overhead in computing time is proportional to the number of surface patches. This means that it is also strongly dependent on the shape, area and deformation of the flame surface, which will be different for each simulation case. For the Premixed Flame case, our implementation generates an overhead of about 30 h or 50 % for the whole simulation. The memory consumption behaves very similarly, as it is also directly dependent on the number of micro-patches. At the start of the simulation, we consume about 90 GB of additional memory, which is an overhead of about 30 % for this simulation

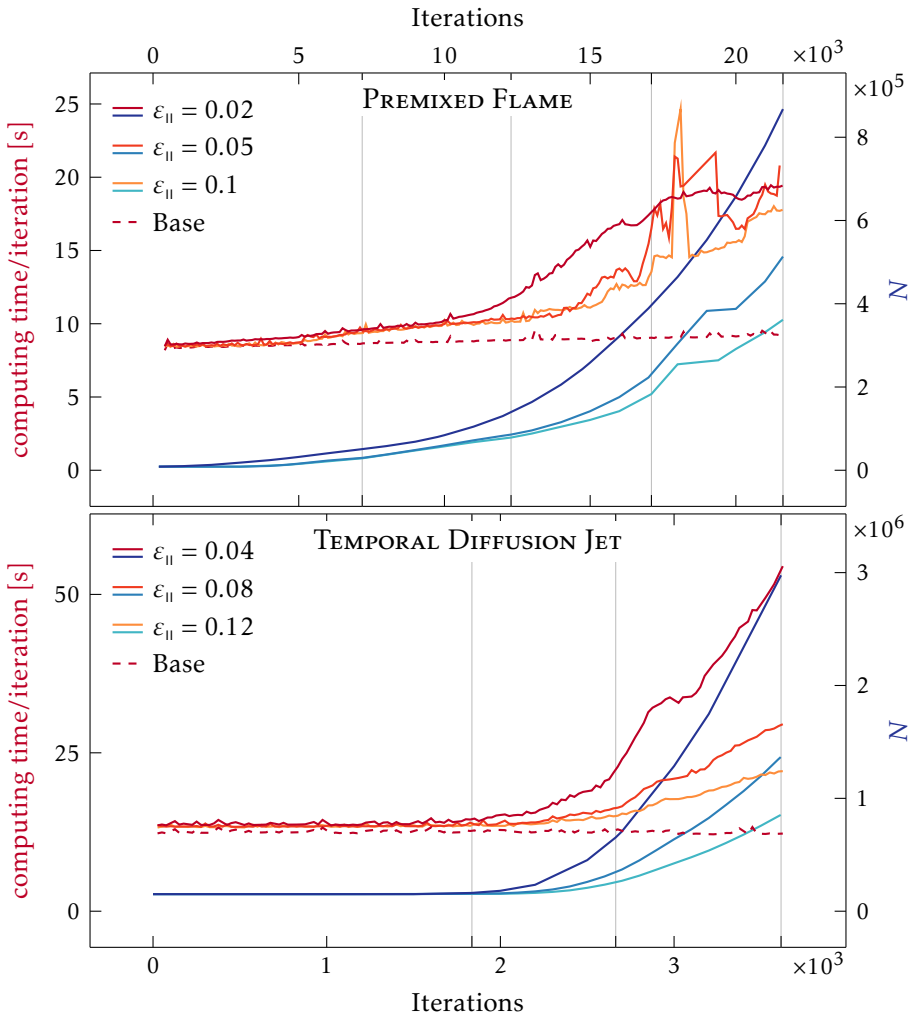

Figure 5.9: Computing times per iteration for different subdivision thresholds ε_{II}. We show the base computing time of the simulation without surface tracking (- -) as well as the total times with our method enabled (≡). The number N of micro-patches over time is shown as context information (≡). The vertical lines mark the snapshots shown in Figures 5.7 and 5.8.

case. This rises to about 400 GB at the end of the simulation, which represents an overhead of about 130 %. The total overhead in computing time in the Temporal Diffusion Jet case is 8.4 h or about 67 %. The additional memory consumption here reaches from 800 GB to 1800 GB, or 44 % to 150 %.

If we were to implement our approach as a post-processing step, we would need to store the raw simulation data to disk, possibly transfer it over a network, and read it again for each time step. If we only store the simulation variables that are directly needed for tracking the flame surface (flow velocity and one scalar variable), we would already need 100 TB of storage space for the whole Premixed Flame case, and 60 TB for the Temporal Diffusion Jet case. This is more than is typically available for a user or project on a current high performance computing cluster. Additionally, writing this data to disk for every simulation time step alone would incur about 21 h of run time on our computing cluster for the Premixed Flame case and about 13 h for the Temporal Diffusion Jet case. If we write all simulation variables, this increases to 420 TB (90 h) and 320 TB (70 h), respectively. Considering that this is just part of the overhead inherent to a post-processing approach makes it clear that an in-situ solution is the only way of obtaining results with reasonable cost.

5.6 Discussion

Due to its nature as an algorithm designed for in-situ execution in a highly parallel environment, our method has some inherent limitations. The most significant problem arises from the exponential nature of stretch in a flow field. If a constant flow field stretches a time surface by a factor of two in a certain time interval, the surface will quadruple its area in double the time and so on. As a consequence, errors in estimating this stretch will accumulate exponentially over time.

Because we are operating in-situ and can not go back in time to fix errors after the fact, we must split micro-patches early enough to limit the amount of error accumulation we get. This is especially critical for keeping the surface sufficiently covered in patches. Even a tiny hole left at some point in time may eventually grow very large. Splitting micro-patches early to limit error accumulation leads to very large numbers of patches over time. Therefore, the error thresholds $r_\perp$ and $r_\parallel$ have to be carefully chosen, taking into account the acceptable error for the maximum investigated time interval, and the overhead in terms of memory consumption and computing time added to the simulation. This is not a trivial task and requires some experience of the user, and it is not clear how it could be simplified. However, the tangential deformation will generally be investigated in statistics with other simulated

quantities, where the error can be analyzed and taken into account. In future work, we want to investigate a strategy for better controlling the number of micro-patches over time. This could be handled by periodically merging or redistributing patches on the surface. This is a global process that would pose challenges for parallelization.

Because we track independent micro-patches, we do not produce a closed, manifold mesh of the flame surface. This is due to our strict parallelization requirements, which are not met by the global nature of subdivision and join operations in meshes. Direct rendering of the data produced by our method can be done via splatting of the micro-patches. If a closed mesh surface is required, it can be reconstructed using any available meshing algorithm for point clouds.

Because our algorithm only tracks micro-patches initialized on the starting surface, it does not handle the case of new disconnected surface parts appearing during the course of the simulation. This can be easily addressed by periodically checking for new parts of the surface that are not yet covered, and initializing new patches. For new surface parts streaming in from a non-periodic boundary, a more sophisticated approach might be necessary, which is a subject for future work.

If we wanted to measure the tangential deformation of the surface only at single points, we could simply compute the product integral of the instantaneous Jacobian $\mathbf{J}(\mathbf{u})$ along the path of each point. This would not require the tracking of ghost particles and would be cheaper in terms of communication and memory overhead. However, our goal is to track the whole surface. By measuring deformation based on the relative movement of multiple points, we get the average of a finite part of the surface. This introduces a filtering effect and better represents the behavior of the surface as a whole. More importantly, we need the information gained from tracking a group of points to compute the error measures $e_{\parallel}$ and $e_{\perp}$, which are based on the deviation from linear behavior. Without this information, which is not included in $\mathbf{J}(\mathbf{u})$, we could only use more inaccurate criteria for splitting and merging patches. This would lead to larger errors in the measurement of deformation and larger holes in the surface.

Our algorithm is the first to provide a viable way of tracking the whole flame surface in-situ over the complete simulation time. This opens new pathways to the investigation of flame behavior. Single snapshots, which are still often the basis for the analysis of a simulation, do not show detailed changes in surface shape over time, and the correspondence between points on the surface in two different snapshots can not be reconstructed. Because we explicitly track single points on the surface over extended periods of time, the evolution of simulation variables at the point positions over time also becomes easily observable. Obtaining this data for the complete flame surface enables

visual and statistical evaluation of direct numerical combustion simulations on a new scale. Our novel method of measuring tangential deformation provides combustion researchers with a new quantity to study the effects of flame-turbulence interactions. This integration-based quantity is only made possible by the in-situ nature of the algorithm. Accurate path line integration is simply not possible as a post process, if the simulation data can only be stored to disk in a massively reduced temporal or spatial resolution.

The overhead in memory and computing time caused by our method is fairly large compared to existing in-situ visualization approaches, which are generally designed to require only a small fraction of the computing time of the simulation [85]. In this context, it is important to note that our approach is not meant to be a visualization method only, and it is not meant to be a general-purpose tool that is activated for every simulation. It is a specialized analysis tool that can be used in situations where combustion researchers are specifically interested in the detailed behavior of the flame surface over long time periods.

Such tools are necessary when a detailed analysis of the low-level behavior of the flame is required. For example, Scholtissek et al. [74] report a four-fold increase in computing time for their gradient trajectory tracking, which enabled them to develop a more accurate flamelet model. In this case, the analysis was employed in the context of a research project that required accurate low-level information which can only be achieved with high computational overhead. We expect our algorithm to be used in a similar context. It is of particular interest for the building of combustion models and the investigation of local flame extinction and re-ignition mechanics. Existing combustion literature, such as works by Sripakagorn et al. [73], observe the history of temperature and heat release at single points on the surface that are tracked over time. By providing such histories for a great number of points covering the whole flame surface, we enable a statistical evaluation that could be the basis for new models of unsteady flame behavior.

6

Conclusion

We have presented two approaches for the visualization and analysis of different aspects of the flame front in turbulent combustion DNS. Both approaches are designed to deal with the huge amount of raw data that has become the bottleneck of large-scale simulations.

The sparse representation for premixed flames presented in Chapter 4 focuses on the analysis of flamelets, i.e., the behavior orthogonal to the surface. We sample the profiles of simulation variables at many locations distributed over the surface and approximate them with simple models to get a space-saving representation of the flame. This representation can directly be used for flamelet-related analysis and visualization, or the full scalar fields can be reconstructed for regular post-processing.

The flame surface tracking algorithm presented in Chapter 5 focuses on the tangential behavior of the flame. We track the surface using independent micro-patches that refine and coarsen without using any neighbor information. This gives us a complete picture of the behavior of the surface over time, particularly about the history and relative movement of points attached to the surface. This information is crucial for understanding and modeling the unsteady behavior of the flame.

Both approaches are contributions towards a visualization/analysis toolbox for in-depth quantitative analysis of DNS for the purpose of combustion modeling. In contrast to the many important contributions towards fast and effective general-purpose visualization for large-scale visualizations that have been developed in recent years, these approaches represent a more targeted class of visualization tools that may be afforded more computational resources and time in order to answer specific research questions. More work is necessary to continue bridging the gap between the two and develop a range of tools from general to specific that support the full analysis process of combustion researchers. An important area of research in this regard are visualization techniques that support the analysis of unsteady behavior, such as transport and mixing. The flame surface tracking algorithm we propose belongs to this category. Compared to the numerous methods for visualizing single snapshots of the data, these techniques are more challenging technically and conceptually. As the frameworks for in-situ processing are improving and taking care of some of the technical challenges, we will hopefully see increased activity in this area.

III

Line Features in 3D Second-Order Tensor Fields

7

This chapter is based on the publication: T. Oster, C. Rössl, and H. Theisel. "The Parallel Eigenvectors Operator". In: *International Symposium on Vision, Modeling and Visualization (VMV)*. The Eurographics Association, 2018

The Parallel Eigenvectors Operator

Feature extraction is one of the most successful types of techniques for scientific visualization. When we talk about features, we mean geometric structures where the data fulfills certain interesting criteria. Chapter 2 introduced several generic features, such as ridges in scalar fields, critical points and vortex core lines in vector fields, and degenerate structures in tensor fields. Extracting and representing such features is an effective approach to understanding even complex scientific datasets.

A number of line-type features for scalar- and vector fields can be expressed in terms of a common operation: the PV operator [19]. This operator yields all locations where two vector fields defined on the same domain are parallel. It delivers structurally stable lines, i.e., stable under the influence of noise, which we call PV lines. Depending on the concrete vector fields it is applied to, the PV operator can be used to extract ridge and valley lines, extremum lines, vortex core lines and separation- and attachment lines from scalar or vector data.

Some of these features are originally defined as the locations where a vector is parallel to an eigenvector of a tensor field. Such cases can be broken down to an application of the regular PV operator. However, this is not possible

if a feature is defined by the locations where two tensor fields have parallel eigenvectors.

In this chapter, we extend the concept of the PV operator to tensor fields. We define the *parallel eigenvectors* (PEV) operator on two (not necessarily symmetric) 3D second-order tensor fields. Let $\mathbf{S}(\mathbf{x})$ and $\mathbf{T}(\mathbf{x})$ be two such tensor fields. Then the PEV operator yields all locations $\mathbf{x}$ where $\mathbf{S}$ and $\mathbf{T}$ have parallel real eigenvectors. This can be concisely expressed as

$$\mathrm{PEV}(\mathbf{S},\mathbf{T}) = \{\mathbf{x} \mid \exists\, \mathbf{e} \in \mathbb{R}^3, \mathbf{e} \parallel \mathbf{S}(\mathbf{x})\mathbf{e} \parallel \mathbf{T}(\mathbf{x})\mathbf{e} \wedge \mathbf{e} \neq \mathbf{0}\}. \tag{7.1}$$

In this chapter, we establish this operator by ...

- ... studying its properties. In particular, we show that the PEV operator produces structurally stable line structures.
- ... presenting a numerical algorithm to extract PEV lines in piecewise linear tensor fields. The main idea is to do a recursive search not only in 3D space but simultaneously in 3D space and the space of all possible eigenvectors.
- ... applying it to compare pairs of stress tensor fields defined on the same domain.

At first glance, the extension of the PV operator to eigenvectors seems trivial: given a tensor field, consider all eigenvector fields as vector fields and apply the PV operator to them. However, this naive approach cannot give well-defined and stable results for the following reasons:

- Undefined length and orientation of eigenvectors:
  Eigenvectors of a matrix are not unique but span linear subspaces. To express the field of eigenvectors as a vector field, heuristic choices about the length and orientation of the vectors are necessary. Applying such choices globally can not always give results that are free of discontinuities
- Existence of multiple eigenvectors:
  Regions with three real eigenvectors require a decision on which of them to use for the PV operator – a decision that is particularly non-unique in near-isotropic regions (i.e., where the difference between two real eigenvalues is small).
- Discontinuities in eigenvectors:
  A small change of a tensor does not necessarily result in a small change of the eigenvectors. In fact, in near-isotropic regions, a small change of the tensor may result in a large change of the eigenvector. Moreover, in regions of transition between real and imaginary eigenvalues (i.e., in neighborhoods containing both tensors with all real eigenvalues and

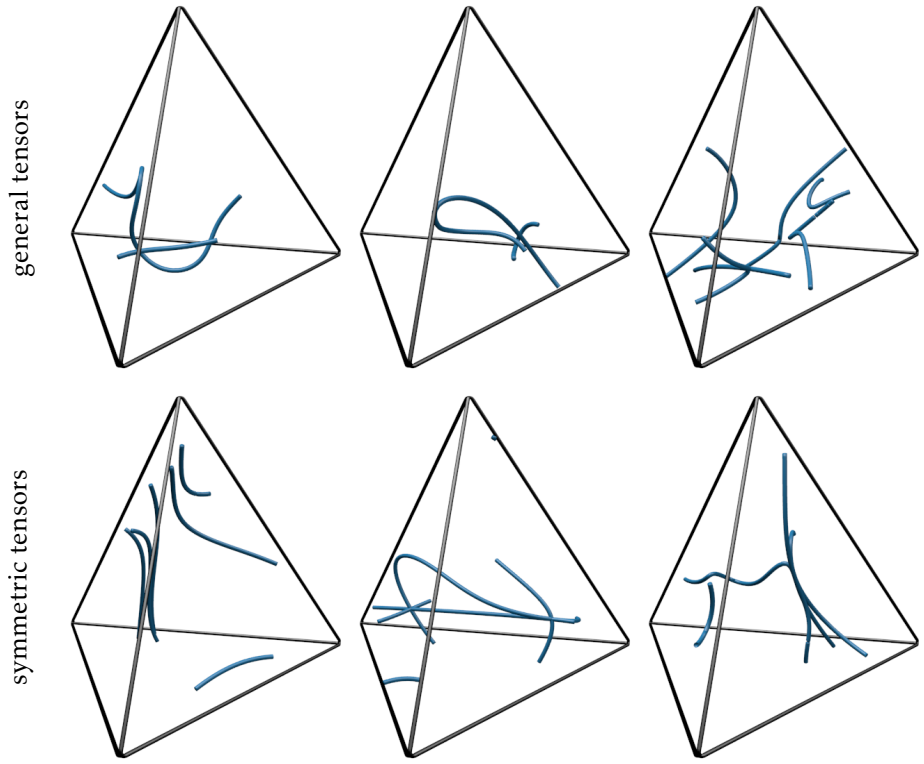

Figure 7.1: PEV lines in pairs of random linear tensor fields.

tensors with complex eigenvalues), a small change of the tensor can result in a sudden appearance or disappearance of real eigenvectors.

All of these problems show that eigenvector fields are fundamentally different from vector fields, for which the PV operator is designed. Extracting PEV lines requires new algorithms that are explicitly designed for tensor fields.

In the following, we first give an overview of related work. We then explore the theoretical properties of the PEV operator in Section 7.2, before detailing our algorithm for finding PEV lines in piecewise linear tensor fields in Section 7.3. In Section 7.4, we apply our algorithm to mechanical stress tensor data. We close with a discussion and future work in Section 7.5 and Section 7.6.

7.1 Related Work

The PEV operator is related to the PV operator as well as tensor field visualization in general. We have already given an overview of tensor field visualization in Section 2.3, so we will focus on literature regarding the PV operator here.

The PV operator was introduced by Peikert and Roth [19] as a generalization of a concept that had been used with slight variations in a lot of different contexts. Among these are ridge detection in scalar fields [149], extraction of attachment/separation lines in flows [150], and the identification of vortex core lines [18, 151].

In his PhD thesis, Martin Roth [152] gives an overview of several numerical algorithms for the PV operator. Most of them are based on first finding intersections of PV lines with the surface of cells of a dataset. The resulting intersection points are then connected to lines using different kinds of heuristics.

An alternative approach is to trace PV lines starting from a seed point. Algorithms using this general approach have been proposed by Banks and Singer [151], Miura and Kida [153], Sukharev et al. [154] and Theisel et al. [140]. Methods for avoiding the accumulation of errors when tracing PV lines were introduced by van Gelder and Pang [155], as well as Weinkauf et al. [156].

While most PV algorithms operate on piecewise linear data that is not time-dependent, there are some publications that deal with higher-order data or use higher-order methods. This includes approaches for finding curved vortex core lines [157], scale-space techniques [158], and computing the PV operator on time-dependent [159, 160] or piecewise analytic vector fields [161]. Recently, Gerrits et al. [13] proposed an approximate parallel vectors operator for ensembles of more than two vector fields. The PEV operator we introduce here deals with higher-order data of a different kind: It operates on tensor instead of vector data.

7.2 Theoretical Considerations

Having defined the PEV operator in Equation (7.1), we use this section to study its properties. We show that like the PV operator, the PEV operator yields structurally stable lines, i.e., they do not disappear when adding noise. However, unlike the PV operator, multiple PEV lines may stably intersect in a single point if the two tensor fields are symmetric.

Given the similarity to the PV operator, one would already expect curves as PEV solutions. The case, however, is slightly more complicated because eigenvectors can transition from real to imaginary, and they are not uniquely defined in isotropic regions. Even considering these cases we can formulate the main theorem

Theorem 1. *The* PEV *operator yields structurally stable curves that are either closed or end at the boundaries of the domain.*

The proof for this theorem, which was provided by Holger Theisel, can be found in Appendix B.

We now study the possibility of PEV lines intersecting in a single point. We call such points, where more than one pair of eigenvectors is parallel, *bifurcation points*.

Theorem 2. *For general (asymmetric) tensor fields, bifurcation points are structurally unstable, i.e., they disappear under small perturbations of the tensor fields.*

To show this, we consider a PEV line l and observe the other eigenvectors (the ones that do not define l) along its path. Since they are not constrained by each other, more than one condition must be fulfilled along l for the other eigenvectors to become parallel. This can be interpreted as having at least two independent scalar values that must vanish at the same point along l. If this happens, adding noise will split up the points on l of common zero crossings and the bifurcation point will disappear.

This situation is different if the tensor fields are symmetric.

Theorem 3. *For symmetric tensor fields, structurally stable bifurcation points exist where both fields have three pairs of parallel eigenvectors.*

This can be shown as follows: If two symmetric tensor fields $\mathbf{S}$, $\mathbf{T}$ have two pairs of parallel eigenvectors, the third pair must be parallel as well, due to the orthogonality of the eigenvectors. Further, we consider again a PEV line l that is defined by the vector $\mathbf{e}$ along l that is eigenvector of both $\mathbf{S}$ and $\mathbf{T}$. All other eigenvectors of $\mathbf{S}$ and $\mathbf{T}$ are perpendicular to $\mathbf{e}$ and can therefore be expressed by one number: the rotation angle around $\mathbf{e}$. The conditions of further pairs of common eigenvectors can then be described as the roots of one scalar function: the difference in rotation angles. Adding noise will slightly change the location of l and slightly change the location of zero crossings on l, but does not make them disappear. Consequently, bifurcation points in symmetric tensor fields are structurally stable.

Figure 7.1 shows some examples of PEV lines in random linear tensor fields. The examples for symmetric tensor fields at the bottom show clear examples of bifurcation points.

7.3 Extracting PEV Lines from Piecewise Linear Data

We will now detail our algorithm for finding PEV lines in piecewise linear tensor fields. We assume that both tensor fields are defined on the vertices of the same tetrahedral mesh. The general approach is to first find all intersections of PEV lines with the faces of the mesh, and then to connect those points to lines.

We showed that PEV structures are lines in the structurally stable case. It follows that their intersections with the triangular faces of a tetrahedral mesh are isolated points. Finding an analytic solution to the parallel eigenvectors problem is impossible, as it involves the intersection of cubic polynomials. Instead, we opt for a numerical approach that is based on recursive subdivision both on the triangle and in the space of possible eigenvector directions.

Our algorithm can be summarized as follows: We first find a direction $\mathbf{r}$ which becomes an eigenvector of both $\mathbf{S}$ and $\mathbf{T}$ at some (possibly different) points inside the triangle. If such a direction is found, we subdivide the triangle and check the parts for possible eigenvector directions again. We do this until we converge on a single point where both $\mathbf{S}$ and $\mathbf{T}$ have parallel eigenvectors. In order to find a valid direction $\mathbf{r}$, we perform another recursive search in the space of possible eigenvector directions, which we represent as some triangulation of a hemisphere centered at the origin. In the following, we describe the details of this algorithm.

7.3.1 Mathematical Basis

A linear tensor field on a triangle is defined by the tensors at its three corners. We denote the set of corner points as $\Delta_{\mathbf{x}} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$, and the set of corner tensors as $\Delta_{\mathbf{S}} = \{\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3\}$ and $\Delta_{\mathbf{T}} = \{\mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3\}$. We express the tensor fields in barycentric coordinates $\mathbf{w} = (w_1, w_2, w_3)^{\mathrm{T}}$:

$$\mathbf{S}(\mathbf{w}) = \sum_i w_i \mathbf{S}_i\,, \quad \mathbf{T}(\mathbf{w}) = \sum_i w_i \mathbf{T}_i\,, \quad \text{with} \quad \sum_i w_i = 1\,.$$

The position (in barycentric coordinates $\mathbf{w}$) at which an arbitrary direction $\mathbf{r}$ becomes an eigenvector in $\mathbf{S}$ is given by the solution to

$$\mathbf{S}(\mathbf{w})\mathbf{r} = \sum_i w_i \mathbf{S}_i \mathbf{r} = \lambda \mathbf{r}\,.$$

Rather than needing the exact position, we want to know if the position is inside the triangle, i.e., if $\mathbf{r}$ is a valid eigenvector direction for $\mathbf{S}$. In barycentric coordinates, a point is inside the triangle if all $w_i > 0$. Since the scaling factor λ is arbitrary, we eliminate it:

$$\sum_i \tilde{w}_i \mathbf{S}_i \mathbf{r} = \mathbf{A}(\mathbf{r})\tilde{\mathbf{w}} = \mathbf{r}\,,$$

$$\text{with} \quad \mathbf{A}(\mathbf{r}) = \begin{pmatrix} \mathbf{S}_1\mathbf{r} & \mathbf{S}_2\mathbf{r} & \mathbf{S}_3\mathbf{r} \end{pmatrix}, \quad \tilde{\mathbf{w}} = \mathbf{w}/\lambda\,,$$

and only require that all $\tilde{w}_i$ have the same sign. Using Cramer's rule, the components of $\tilde{\mathbf{w}}$ are

$$\tilde{w}_i = \frac{\det \mathbf{A}_i(\mathbf{r})}{\det \mathbf{A}(\mathbf{r})}\,.$$

Here, $\mathbf{A}_i$ denotes the matrix $\mathbf{A}$ with its i-th column replaced by $\mathbf{r}$. Note that all $\tilde{w}_i$ are divided by the same factor $\det \mathbf{A}(\mathbf{r})$. Since this influences all signs of $\tilde{w}_i$ equally it can be ignored, leading to

$$\hat{w}_i(\mathbf{r}) = \det \mathbf{A}_i(\mathbf{r}). \tag{7.2}$$

The equations for $\mathbf{T}$ are analogous. In the following, we show all equations for $\mathbf{S}$ only. The equivalent equations for $\mathbf{T}$ can be obtained trivially by substituting $\mathbf{T}$ for $\mathbf{S}$. We denote the solutions for $\mathbf{S}$ and $\mathbf{T}$ by $\hat{\mathbf{w}}_\mathbf{S}$ and $\hat{\mathbf{w}}_\mathbf{T}$ respectively, whenever it is necessary to discriminate them.

7.3.2 Subdivision in Direction Space

The core of the algorithm is to find a direction $\mathbf{r}$ for which all components of $\hat{\mathbf{w}}_\mathbf{S}(\mathbf{r})$ have a common sign, and all components of $\hat{\mathbf{w}}_\mathbf{T}(\mathbf{r})$ also have a common sign (that possibly differs from $\hat{\mathbf{w}}_\mathbf{S}$). If this is fulfilled, $\mathbf{r}$ becomes an eigenvector somewhere inside the triangle for both $\mathbf{S}$ and $\mathbf{T}$.

Note that the $\hat{w}_i(\mathbf{r})$ are cubic in $\mathbf{r}$. Finding an analytic solution for $\mathbf{r}$ means analytically finding the intersections of the roots of $\hat{w}_i(\mathbf{r})$, which is impossible. Instead, we solve the problem numerically by applying another recursive search in the space of all possible eigenvector directions. Since we are looking for eigenvectors, the magnitude and orientation of $\mathbf{r}$ are not significant. We can therefore represent this space by some triangulation of a hemisphere centered at the origin (Figure 7.2, right). We again express a direction in a triangle $\Delta_\mathbf{r} = \{\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3\}$ on this hemisphere in barycentric coordinates u_j of its corner vectors:

$$\mathbf{r}(\mathbf{u}) = \sum_j u_j \mathbf{r}_j .$$

Substituting this in Equation (7.2), the barycentric coordinate functions now become

$$\hat{w}_i(\mathbf{u}) = \det\left(\sum_j u_j \mathbf{A}_i(\mathbf{r}_j)\right). \tag{7.3}$$

We can now express the polynomials $\hat{w}_i$ in Bernstein-Bézier basis as

$$\hat{w}_i(\mathbf{u}) = \sum_{\substack{j,k,l\geqslant 0,\\ j+k+l=3}} \frac{3!}{j!\,k!\,l!}\, u_1^j\, u_2^k\, u_3^l \cdot b_{jkl}. \tag{7.4}$$

Here, b_{jkl} are the 10 coefficients needed to express a trivariate polynomial of degree 3. Note that because the barycentric coordinates $\mathbf{u}$ are restricted to the triangle $\Delta_\mathbf{r}$, they really have only two degrees of freedom.

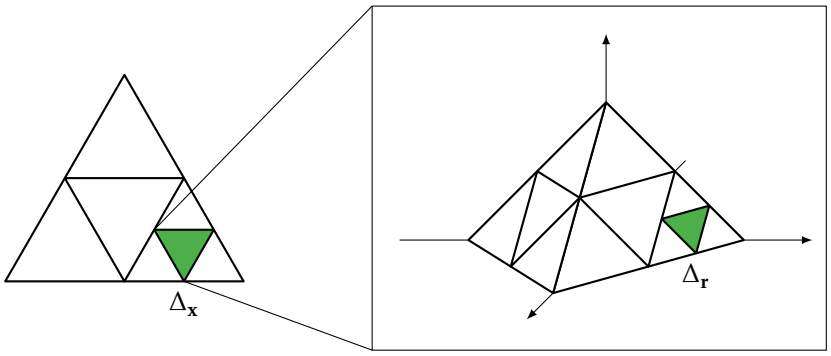

Figure 7.2: Two-level recursion scheme for finding intersections of PEV lines with the faces of piecewise linear tensor fields. For each sub-triangle $\Delta_{\mathbf{x}}$, a recursive search in the space of possible eigenvector directions is performed to find a direction $\mathbf{r}$ that becomes an eigenvector of both $\mathbf{S}$ and $\mathbf{T}$ within $\Delta_{\mathbf{x}}$.

We use the property that a polynomial in Bernstein-Bézier form is bounded in its domain by the convex hull of its coefficients [162]. This means that $\hat{w}_i$ is positive over the whole triangle if all $b_{jkl} > 0$, and negative over the whole triangle if all $b_{jkl} < 0$. If the b_{jkl} have different signs, $\hat{w}_i$ might become 0 somewhere inside the triangle.

We use this when recursively subdividing the triangle $\Delta_{\mathbf{r}}$. If any $\hat{w}_i$ might have roots within the triangle according to the Bernstein-Bézier coefficients, then we can not make a decision. We need to subdivide the triangle and check the different parts again. If no $\hat{w}_i$ can have roots within the triangle as indicated by the coefficients, then there are two possibilities:

1. All $\hat{w}_i$ have the same sign everywhere on the triangle
2. The $\hat{w}_i$ have different signs everywhere on the triangle

In case 1, all directions within the triangle become eigenvector directions somewhere in $\Delta_{\mathbf{x}}$ for both $\mathbf{S}$ and $\mathbf{T}$. If this happens, we can accept any direction within the current triangle as a possible solution. In case 2, no direction within the triangle can become an eigenvector of both $\mathbf{S}$ and $\mathbf{T}$, and the triangle is discarded. When the triangle becomes smaller than some subdivision threshold ε_{r}, and we still can not say for sure that there are no possible eigenvector directions inside, we accept the central direction as a candidate.

7.3.3 Final Numerical Algorithm

The complete algorithm for finding intersections of PEV lines with a triangle of the dataset now works as follows: Start with the complete triangle as $\Delta_{\mathbf{x}}$. Then, search for a direction that becomes an eigenvector of both $\mathbf{S}$ and $\mathbf{T}$ somewhere inside the triangle by using the algorithm described in Section 7.3.2. If such a

Algorithm 1: Find intersections of PEV lines with a triangle

Function FindPEV($\Delta_\mathbf{S}$, $\Delta_\mathbf{T}$, $\Delta_\mathbf{x}$)
  $\mathbf{r} \leftarrow$ FindEigenDir($\Delta_\mathbf{S}$, $\Delta_\mathbf{T}$);
  if $\mathbf{r}$ is **null then**
    return {} ; *// Discard triangle*
  else if size of $\Delta_\mathbf{x} < \varepsilon_s$ **then**
    return $\{(1/3 \sum \mathbf{x}_i,\ \mathbf{r})\}$; *// Accept solution*
  end
  $l = \{\}$;
  foreach $(\Delta'_\mathbf{S}, \Delta'_\mathbf{T}, \Delta'_\mathbf{x}) \in$ (Split($\Delta_\mathbf{S}$), Split($\Delta_\mathbf{T}$), Split($\Delta_\mathbf{x}$)) **do**
    $l \leftarrow l \cup$ FindPEV($\Delta'_\mathbf{S}$, $\Delta'_\mathbf{T}$, $\Delta'_\mathbf{x}$) ; *// Recursive subdivision*
  end
  return l;
end
Function FindEigenDir($\Delta_\mathbf{S}$, $\Delta_\mathbf{T}$)
  $R \leftarrow$ set of triangles covering a hemisphere;
  foreach $\Delta_\mathbf{r} \in R$ **do**
    $\mathbf{r} \leftarrow$ FindEigenDirRecursive($\Delta_\mathbf{S}$, $\Delta_\mathbf{T}$, $\Delta_\mathbf{r}$);
    if $\mathbf{r}$ is not **null then**
      return $\mathbf{r}$;
    end
  end
  return null;
end
Function FindEigenDirRecursive($\Delta_\mathbf{S}$, $\Delta_\mathbf{T}$, $\Delta_\mathbf{r}$)
  Compute Bernstein-Bézier coefficients for $\hat{w}_i$;
  if any $\hat{w}_i$ might have roots within $\Delta_\mathbf{r}$ **then**
    foreach $\Delta'_\mathbf{r} \in$ Split($\Delta_\mathbf{r}$) **do**
      $\mathbf{r} \leftarrow$ FindEigenDirRecursive($\Delta_\mathbf{S}$, $\Delta_\mathbf{T}$, $\Delta'_\mathbf{r}$);
      if $\mathbf{r}$ is not **null then**
        return $\mathbf{r}$;
      end
    end
  else if all $\hat{w}_i$ have the same sign **or** size of $\Delta_\mathbf{r} < \varepsilon_r$ **then**
    return $1/3 \sum \mathbf{r}_i$; *// Accept solution*
  end
  return null ; *// Discard triangle*
end

direction is found, subdivide the triangle and process the parts recursively. If no direction is found, discard the triangle. When a spatial sub-triangle becomes smaller than a subdivision threshold ε_s, we accept the center of the triangle and the accompanying direction $\mathbf{r}$ as a solution candidate. Algorithm 1 shows the procedure in pseudo code.

The result of the algorithm is a list of points on $\Delta_\mathbf{x}$ with corresponding eigenvector directions $\mathbf{r}$. This list of points has to be post-processed for two reasons:

1. For each intersection of the PEV line with the triangle, multiple adjacent candidate points may be found. This happens if eigenvectors of $\mathbf{S}$ and $\mathbf{T}$ are closer than ε_r in a region larger than ε_s, e.g., because the gradient of the tensor fields is very small, or because the PEV line intersects the face at a very steep angle. Choosing ε_r very small helps with this, but it can not be avoided in the presence of limited numerical precision on a computer.
2. A candidate point might not be a PEV point at all. These false positives occur if there are directions $\mathbf{r}$ that become eigenvectors of one of the tensor fields inside $\Delta_\mathbf{x}$, while $\mathbf{A}(\mathbf{r})$ has rank 1 for the other tensor field. For this case, $\hat{\mathbf{w}} = \mathbf{0}$, which means that a consistent sign of all components can never be determined, and subdivision can not be terminated early, even if the tensor field does not have any valid eigenvector directions inside $\Delta_\mathbf{x}$.

We deal with Item 1 by clustering nearby solution candidates. We employ a simple single-linkage hierarchical clustering algorithm [163]. Given two candidate position-direction pairs $(\mathbf{x}, \mathbf{r})$, we define the distance as the maximum of their distances in position- and in direction space. We start with each parameter region as a single cluster. Two clusters are merged if the distance between any two elements from both clusters is smaller than a clustering threshold ε_c. We repeat this process until the number of clusters no longer changes.

We then select the point in each cluster where the corresponding eigenvectors are most parallel as the representative and discard the others. Since we already have eigenvector directions for each point, we do not need to explicitly compute them again. Instead, we use the parallelism error

$$e_p = \left\| \frac{\mathbf{S}(\mathbf{w})\mathbf{r}}{\|\mathbf{S}(\mathbf{w})\mathbf{r}\|} \times \frac{\mathbf{r}}{\|\mathbf{r}\|} \right\| + \left\| \frac{\mathbf{T}(\mathbf{w})\mathbf{r}}{\|\mathbf{T}(\mathbf{w})\mathbf{r}\|} \times \frac{\mathbf{r}}{\|\mathbf{r}\|} \right\|, \tag{7.5}$$

which measures the deviation of $\mathbf{r}$ from the true eigenvectors of both $\mathbf{S}(\mathbf{w})$ and $\mathbf{T}(\mathbf{w})$.

This algorithm has a complexity of $O(n^3)$ in the number of solution candidates. Typically n is small: less than 200 candidates are found in the vast

majority of cases. At this scale, the performance impact of the clustering algorithm is negligible.

In order to address Item 2, we discard all candidate points for which e_p is greater than some parallelism threshold ε_p. This threshold can be chosen quite coarse (e.g. 0.01), as e_p is typically quite large for false positive candidate points.

In certain cases, the PEV line might not intersect the triangle at a single point. This happens in the structurally unstable cases where eigenvectors are parallel on a structure with a dimension larger than 1, or where the PEV line is completely in the plane of the triangle. In these cases, the recursive subdivision will not converge on isolated points and slow down the algorithm considerably. To mitigate this, we terminate the recursion if the number of triangle subdivision operations exceeds a reasonable threshold.

Once we have clustered the candidate solutions and removed false positives, we have a number of final PEV points for each face of the mesh. These PEV points are now connected to lines on a cell-by-cell basis. This problem is also faced when computing the PV operator, where it has been solved in a variety of ways using different heuristics, which can be employed here as well. In our implementation, we simply connect two points if they are the only two intersections of a PEV line with a grid cell. In case of more than one intersection, we greedily connect pairs of points that have the most similar parallel eigenvector directions, assuming that PEV lines are generally smooth relative to the grid resolution.

7.4 Results

We applied our method to different stress tensor fields from solid mechanics simulations. The Cauchy stress tensor (often referred to as σ in mechanics literature) is a symmetric tensor that describes the local stress state of an object experiencing small elastic deformations. Its eigenvectors are real and orthogonal and point in the directions of the principal stresses. The sign of the eigenvalues indicate if the stress is compressive or tensile in the corresponding direction. When comparing two different stress tensor fields, PEV lines occur where two principal stress directions align. We show PEV lines for three different stress tensor datasets: Two different point loads applied to a uniform material, two different traction forces applied to the end of a clamped beam, and two different load scenarios applied to a flange.

We used the same parameters for all datasets: $\varepsilon_s = 10^{-3}$, $\varepsilon_c = 5 \times \varepsilon_s$, $\varepsilon_d = 10^{-9}$, $\varepsilon_p = 10^{-3}$. Our results were computed on a 4-core Intel Core i7 CPU at 3.4 GHz.

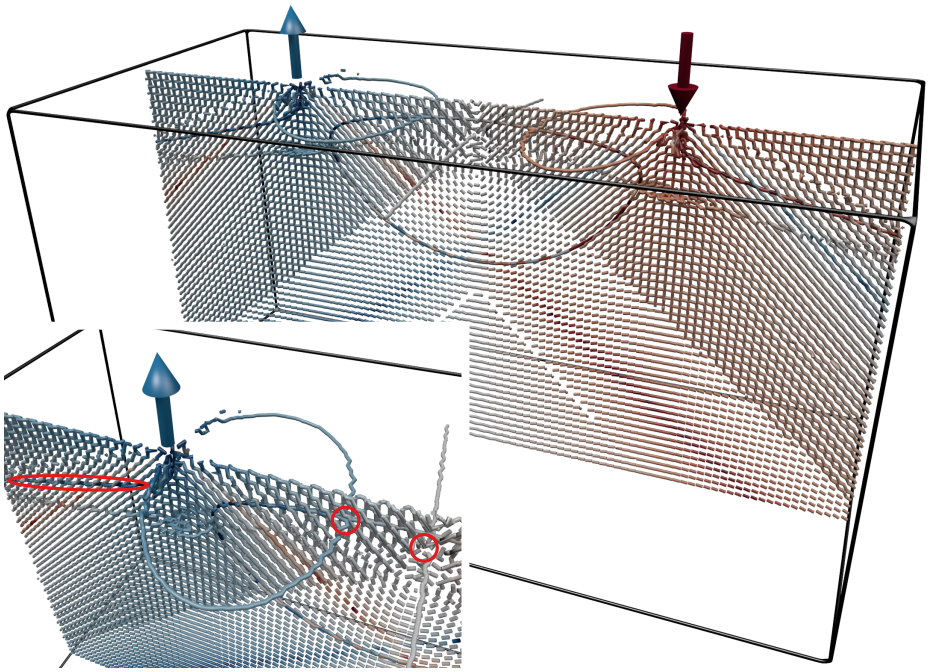

Figure 7.3: PEV lines for the POINT LOADS dataset. Lines are colored by absolute eigenvalue ratio.

7.4.1 Point Loads

In this example, we compare two different point loads applied to a uniform material with infinite extents. We show our results in Figure 7.3. The first load (red arrow) is a compressive force, the second load (blue arrow) is a tensile force of equal magnitude. We compute the PEV operator for the two resulting stress tensor fields. The POINT LOADS case has a closed analytic solution [164], which we sampled on a regular grid with $100 \times 50 \times 50$ points using the `vtkPointLoad` source from the Visualization Toolkit [42]. We then tetrahedralized the data, resulting in 1.1 million cells and 4.8 million faces. The computing time for this dataset was 4.2 h, which means that PEV intersections on each face were found in 3.2 ms on average.

Since the point loads were applied in the same plane, this synthetic dataset shows the rare case where eigenvectors are parallel on a plane instead of a line. This degenerate case also accounts for the long computing time, as for each face intersected by the PEV plane, the recursive subdivision can not be terminated early. Even though this structurally unstable case produces visual artifacts when using our method, interesting PEV line structures are still visible. There is a bifurcation point exactly at both load points, extending

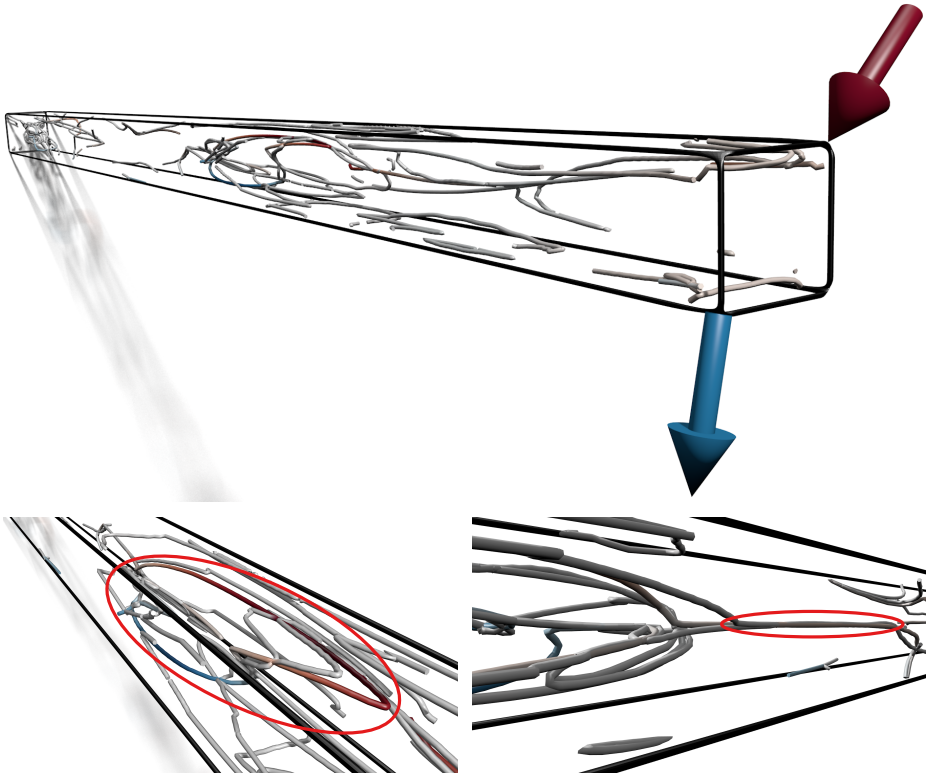

Figure 7.4: PEV lines for the CLAMPED BEAM dataset. Lines are colored by the eigenvalue of the stress tensor corresponding to the red arrow (red is positive, blue is negative). Interesting structures mentioned in the text are highlighted.

into a curved ring slightly below the surface. At the second intersection of this ring with the central plane, another closed PEV structure embedded into the plane becomes visible. Within a PEV plane, structures where all three eigenvectors are parallel become lines, instead of points. A similar structure can be observed starting at the load points and leading outwards. In the center of the dataset, there is another PEV line, orthogonal to the plane and slightly curved downwards, separating the two load points. For this dataset, we colored the PEV lines by the absolute eigenvalue ratio of the parallel eigenvectors. Since both tensor fields result from forces of equal magnitude, this ratio makes visible the directions in which forces propagate outwards from the load points.

7.4.2 Clamped Beam

Next, we extracted PEV lines for a beam that is fixed on one side. We applied two different traction forces on the free end of the beam, whose directions

are indicated by the blue and red arrows in Figure 7.4. The Clamped Beam dataset consists of 150k cells and 600k faces. The computing time was 26 min, which means 2.6 ms per face on average.

There are two regions of particular interest in the Clamped Beam. The first is near the middle of the beam, where a curved structure has high eigenvalues in both tensor fields (visible in red and blue in Figure 7.4, bottom left). This is where the beam experiences a lot of stress, and therefore the tensor fields have a high magnitude. The second is somewhat further towards the wall. Here, all three eigenvector directions are parallel along a line structure near the center with considerable length (bottom right). This area seems to be the most similar between the two scenarios in terms of stress directions.

7.4.3 Flange

Our final stress tensor dataset is a flange from an OpenFoam [165] tutorial. We subjected the flange to two different loads, applied on the back wall and the flange ring (see red and blue arrows in Figure 7.5). The original mesh uses polygonal cells, which is why we resampled the data on a regular grid, resulting in 1.2 million cells and 5 million faces. The computing time was 36 min, i.e., 0.5 ms per face.

The Flange dataset exhibits a lot of PEV lines, which can be seen in Figure 7.5 (top). Most of the PEV lines correspond to eigenvectors with small eigenvalues in both tensor fields. We therefore filtered out all PEV lines where both eigenvalues are very small in the bottom images in Figure 7.5. Especially prominent are two bifurcation points with high eigenvalues between the two outer screw holes and the central tube (bottom left). There are also PEV lines leading outwards both above and below the screw holes (bottom right). In general, the most similar directions of significant stress are near the screw holes and in the area where the large flange ring meets the central block.

7.5 Discussion

The PEV operator was introduced as a generic operator, its interpretation is dependent on the application scenario.

For stress tensors in mechanical engineering, the PEV operator gives insight into the alignment of the tensors under different acting forces. Areas with PEV lines can be wanted or unwanted. In areas with present PEV lines, the stress tensor is similarly oriented for different external forces. This could be used e.g., for deciding the placement of structural reinforcements or to guide the selection of materials. In regions without PEV lines, there is no stress in a preferred direction when applying different outer forces and material with a more isotropic behavior could be used.

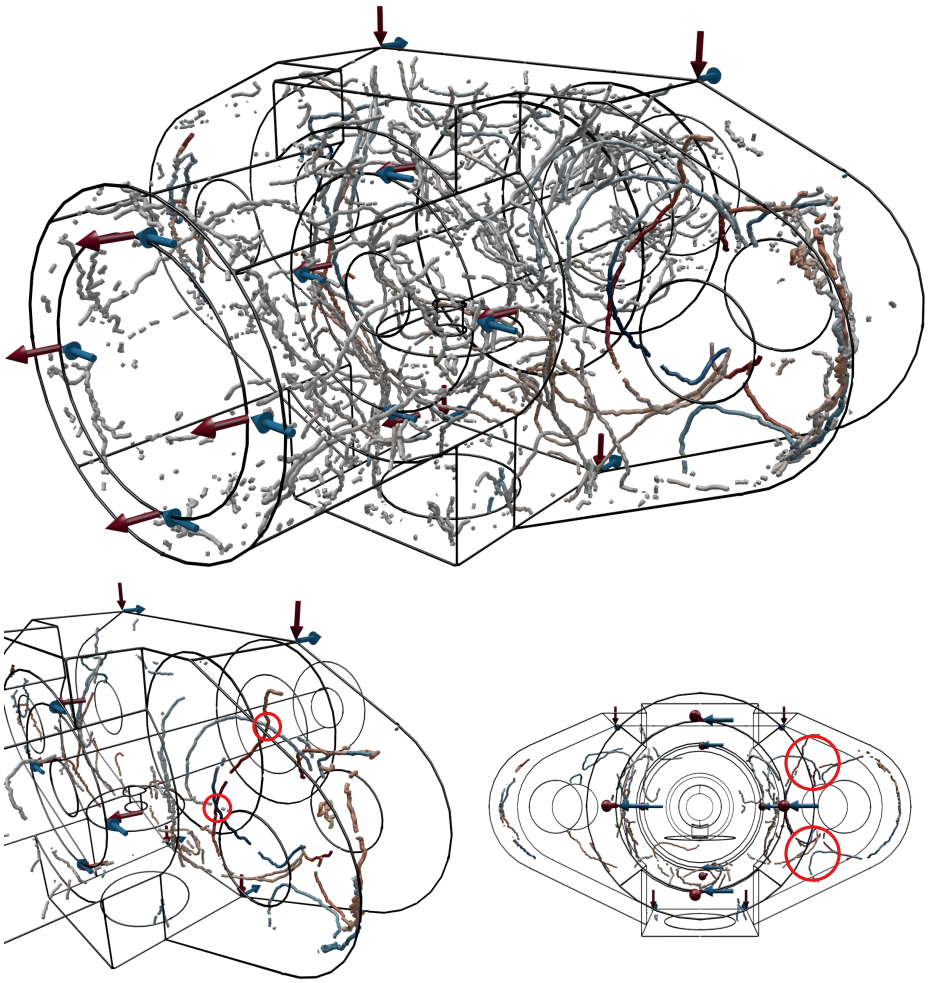

Figure 7.5: PEV lines for the FLANGE dataset. Lines are colored by the eigenvalue corresponding to the red arrows. The top image shows all lines while the bottom shows only lines where the magnitudes of both eigenvalues are above a threshold. Interesting structures mentioned in the text are highlighted.

Besides this particular interpretation, there are general interpretations that are common to all applications of the PEV operator. The PEV operator is agnostic to isotropic scaling of the tensors. It gives information about the orientation of the tensors only. In this way, the PEV operator can be seen as an addition to many standard measures for comparing tensors like norm, trace, or eigenvalues.

The presented algorithm for piecewise linear tensor fields does not use any derivatives of the data. It depends on a number of thresholds to guide subdivision levels and filtering. The spatial subdivision threshold ε_d influences the accuracy of the resulting PEV lines. A small threshold means more subdivisions and is one of the main factors influencing performance. Since subdivision converges to single points, the computing time increases logarithmically when decreasing ε_d.

The directional subdivision threshold ε_r guides the accuracy of the obtained eigenvector direction. Typically, the smaller the current spatial triangle $\Delta_{\mathbf{x}}$ becomes, the smaller the region of valid eigenvector directions. For increasing subdivision level in space, the valid eigenvector directions will converge on a point. This means that for small ε_d, the recursion in the space of directions will generally proceed to the highest subdivision level. The influence of ε_r on computation time is about the same as for ε_d. However, an accurate determination of eigenvector direction is essential to decrease the number of candidate PEV solutions that have to be clustered. This means that ε_r should be chosen very small. We find $\varepsilon_r = 10^{-9}$ to be a choice that provides consistently good results.

Because our algorithm can produce multiple candidate points for an intersection of a PEV line with a tetrahedral face, we need to cluster the results. Theoretically, all candidate points should be in adjacent triangles, as (unoriented) eigenvector directions in linear tensor fields do not oscillate on small scales. However, due to numerical noise and rounding errors on a computer, some candidate triangles might not be exactly adjacent to each other. To bridge this gap, the clustering threshold ε_c defines the radius in which two candidate solutions are considered to belong to the same cluster. Because the numerical noise influencing the size of gaps between candidate solutions is random, we do not expect candidates to be more than two or three lengths of ε_d from each other. We recommend to set ε_c to some fixed multiple of ε_d. In our experiments, $\varepsilon_c = 5\varepsilon_d$ proved sufficient for all datasets.

The parallelism threshold ε_p is used to weed out false positive candidates that are a byproduct of our algorithm. It must be chosen carefully to separate false positive solutions from numerical errors. Because of this threshold, the spatial subdivision threshold ε_d can not be chosen arbitrarily large. The larger ε_d, the larger the possible difference in eigenvector direction between the real PEV point and the tensor at the center of the triangle, which is chosen as a

representative. In general, the choice of ε_p is dependent on ε_d. More spatial subdivision levels enable a smaller choice of the parallelism threshold. In our experiments, a choice of $\varepsilon_p = 10^{-2}$ for $\varepsilon_d = 10^{-3}$, and $\varepsilon_p = 10^{-3}$ for $\varepsilon_d = 10^{-6}$ worked very well.

7.6 Limitations and Future Research

Limitations can be discussed from two points of view: the operator itself and the presented numerical extraction algorithm. A limitation of the PEV operator is that it can only be applied to problems where the norm of the tensors does not matter. This limits the applicability but on the other hand focuses on features of the tensor fields that are less covered by other methods.

The presented extractor works for piecewise linear tensor fields only. An extension to hexahedral grids as well as higher order interpolations is subject of future research. The performance of the algorithm can be improved by parallelization. In principle, the algorithm is parallelizable (each cell can be treated independently). However, even if this is carefully carried out, interactive frame rates (for instance for comparing time-dependent tensor fields) are hardly achievable because we still have to do a search in a 5D space. Due to the possibility of many candidate solutions for each intersection of a PEV line with a face, we can not give an upper limit on the error of the PEV line position. This might limit its applicability in cases where a highly accurate PEV line is required.

We stated in Theorem 1 that PEV lines are generally closed. However, the results obtained from our algorithm sometimes exhibit gaps. Because the extractor is a numerical algorithm, and not a combinatorial one, we will sometimes not find solutions on faces where the presence of an intersection point is numerically unstable. This happens for example if the PEV line is parallel and very close to a face, or when the tensor field is almost zero.

In this chapter, we only show examples of the PEV operator for (symmetric) stress tensor fields. Further possible scenarios that are left to future research are the comparative visualization of DTI data or a comparative visualization of Jacobian fields for flow visualization, which are not necessarily symmetric.

8

This chapter is based on the publication:
T. Oster, C. Rössl, and H. Theisel. "Core Lines in 3D Second-Order Tensor Fields". In: *Computer Graphics Forum* 37.3 (2018), pp. 327–337

Core Lines in 3D Second-Order Tensor Fields

Vortex core lines describe the centers of swirling behavior in vector fields. They are a useful tool in vector field visualization because they provide an explicit geometrical representation of an important flow feature. Different definitions and strategies for extracting vortex core lines have been presented in Section 2.2.5. One of the simplest and most popular methods is the one proposed by Sujudi and Haimes [18], which can be computed using the PV operator [19].

Tensor field lines, i.e., lines that are everywhere tangential to an eigenvector of a tensor field (see Section 2.3.4), can also exhibit "swirling" behavior similar to vortices in vector fields. For example, stress tensor fields show stress trajectories winding around a common core in regions of twist. Various visualization methods for tensor fields exist, but to date no approach has been proposed to extract core lines of such vortex-like structures.

As we have already explained in Chapter 7, simply using the Sujudi/Haimes criterion and applying the PV operator to the "eigenvector fields" of the data cannot give consistent results. We therefore introduce *tensor core lines* as an equivalent to vortex core lines in vector fields. Their definition is a direct extension of the Sujudi/Haimes criterion to tensor fields and their extraction is

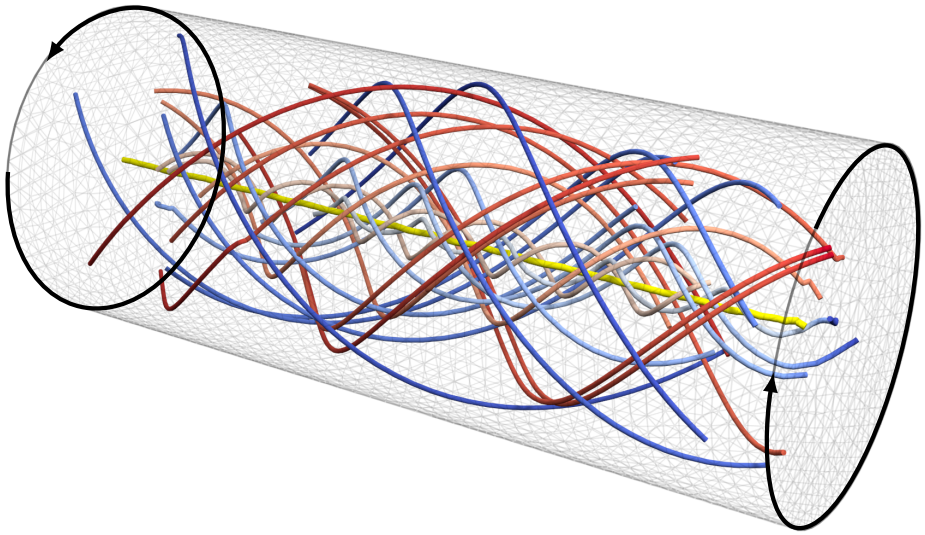

Figure 8.1: Tensor field lines in a stress tensor field induced by applying a torque to a cylindrical shaft. Field lines of both major (blue) and minor (red) eigenvectors show a swirling behavior around a common core line (yellow).

based on the PEV operator. In particular, we make the following contributions:

- We give a rigorous definition of tensor core lines and show that indeed the definition gives structurally stable line structures.
- We provide a numerical algorithm for the extraction of tensor core lines from piecewise linear tensor fields.
- We introduce a filter criterion based on numerical stability to separate significant and insignificant tensor core lines.
- We show tensor core lines in mechanical stress tensor fields, interpret them and compare them with degenerate lines where two eigenvalues are equal.

We introduce tensor core lines in Section 8.1 and study their properties. We then present our algorithm for extracting tensor core lines in Section 8.2. We show our results on several stress tensor fields in Section 8.3 and study the performance, robustness and relation to degenerate lines. As it turns out, our algorithm is sufficiently generic to be applied to the extraction of PEV lines and degenerate lines as well. We show how this can be done in Section 8.4 and close with a discussion in Section 8.5.

8.1 Tensor Core Lines

The Sujudi/Haimes criterion defines a vortex core line as a structure on which streamlines have a locally vanishing curvature. The intuition is that in a region where streamlines show a swirling behavior, there must be a center of rotation where the swirl vanishes. This is fulfilled where the acceleration vector $\mathbf{J}\,\mathbf{v}$ is parallel to the velocity $\mathbf{v}$, i.e., the flow is accelerated on a straight line. Using this formulation, the criterion can be expressed in terms of the PV operator. To only extract vortex core lines, the Sujudi/Haimes criterion requires an additional filter criterion. Vortices only occur in regions with swirling flow behavior, which is indicated by the presence of complex eigenvalues of the Jacobian. Zero-curvature lines also occur in regions where the Jacobian has three real eigenvalues. These hyperbolic trajectories are the centers of simultaneous converging and diverging behavior of the vector field and can be used to extract Lagrangian coherent structures [167, 168]

We apply the idea of Sujudi/Haimes to tensor fields by looking for locations where the curvature of tensor field lines locally vanishes. We define tensor core lines as the locations where tensor field lines have a locally vanishing curvature. Note that like vortex core lines, tensor core lines are not generally field lines of the tensor field. In this section, we provide a formal definition of tensor core lines and examine their mathematical properties.

8.1.1 Definition

Let $\mathbf{T}(\mathbf{x})$ be a 3D second-order tensor field that may or may not be symmetric. We want to find locations where the direction of a real eigenvector of $\mathbf{T}$ does not change when moving along the eigenvector direction, i.e., where the curvature of a tensor field line vanishes. A vector $\mathbf{r} \neq \mathbf{0}$ is a real eigenvector of $\mathbf{T}$ if $\mathbf{T}\,\mathbf{r} = \lambda\mathbf{r}$ for eigenvalue $\lambda \in \mathbb{R}$. To observe the change of eigenvector direction when moving along $\mathbf{r}$, we need to consider the derivative of $\mathbf{T}$ in this direction. The directional derivative of $\mathbf{T}$ along $\mathbf{r}$, which we write as $\nabla_{\mathbf{r}}\mathbf{T}$, is the linear combination of three component-wise derivatives

$$\nabla_{\mathbf{r}}\mathbf{T}(\mathbf{x}) = \nabla\mathbf{T}(\mathbf{x})\,\mathbf{r} = \frac{\partial\mathbf{T}(\mathbf{x})}{\partial x_1}r_1 + \frac{\partial\mathbf{T}(\mathbf{x})}{\partial x_2}r_2 + \frac{\partial\mathbf{T}(\mathbf{x})}{\partial x_3}r_3\,,$$

for $\mathbf{x} = (x_1, x_2, x_3)^{\mathrm{T}}$ and $\mathbf{r} = (r_1, r_2, r_3)^{\mathrm{T}}$.

Given $\nabla_{\mathbf{r}}\mathbf{T}$ we can approximate the behavior of $\mathbf{T}$ along $\mathbf{r}$ as

$$\mathbf{T}(\mathbf{x} + h\mathbf{r}) = \mathbf{T}(\mathbf{x}) + h\nabla_{\mathbf{r}}\mathbf{T}(\mathbf{x})\,, \quad \text{with} \quad h \in \mathbb{R}\,. \tag{8.1}$$

For our zero-curvature requirement to be fulfilled, the eigenvector direction must not change when moving along $\mathbf{r}$, i.e.,

$$\mathbf{T}(\mathbf{x} + h\mathbf{r})\,\mathbf{r} = \mathbf{T}(\mathbf{x})\,\mathbf{r} + h\nabla_{\mathbf{r}}\mathbf{T}(\mathbf{x})\,\mathbf{r} = \mu\mathbf{r}\,, \quad \text{with} \quad \mu \in \mathbb{R}\,. \tag{8.2}$$

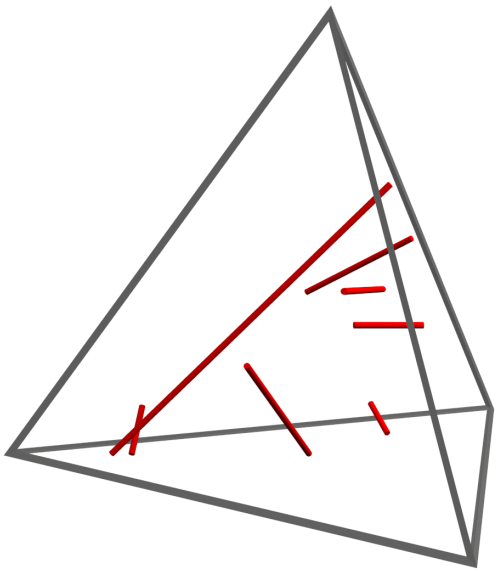

Figure 8.2: Example of seven distinct core lines in a random linear tensor field.

This means that $\mathbf{r}$ is still an eigenvector of $\mathbf{T}$ (with eigenvalue μ) at the slightly offset position $\mathbf{x} + h\mathbf{r}$. If we substitute $\mathbf{T}(\mathbf{x})\,\mathbf{r} = \lambda\mathbf{r}$, we get

$$\begin{aligned} \lambda\mathbf{r} + h\nabla_{\mathbf{r}}\mathbf{T}(\mathbf{x})\,\mathbf{r} &= \mu\,\mathbf{r} \\ \nabla_{\mathbf{r}}\mathbf{T}(\mathbf{x})\,\mathbf{r} &= \frac{\mu-\lambda}{h}\,\mathbf{r}\,, \end{aligned}$$

i.e., $\mathbf{r}$ is also an eigenvector of $\nabla_{\mathbf{r}}\mathbf{T}(\mathbf{x})$. With this, a tensor core line is an isolated line of positions $\mathbf{x}$ where

$$\lambda\mathbf{T}(\mathbf{x})\,\mathbf{r} = \mu\nabla_{\mathbf{r}}\mathbf{T}(\mathbf{x})\,\mathbf{r} = \mathbf{r},$$

for $\mathbf{r} \neq \mathbf{0}$ and $\lambda, \mu \in \mathbb{R}$. In terms of the PV operator, tensor core lines can then be expressed as

$$\mathrm{TCL}(\mathbf{T}) = \{\mathbf{x} \,|\, \exists\, \mathbf{r} \in \mathbb{R}, \mathbf{r} \parallel \mathbf{T}(\mathbf{x})\,\mathbf{r} \parallel \nabla_{\mathbf{r}}\mathbf{T}(\mathbf{x})\,\mathbf{r} \wedge \mathbf{r} \neq \mathbf{0}\}\,. \tag{8.3}$$

In words: a tensor core line is located where an eigenvector $\mathbf{r}$ of $\mathbf{T}$ is parallel to an eigenvector of the directional derivative of $\mathbf{T}$ along $\mathbf{r}$. Note the similarity of this criterion to Sujudi/Haimes, which states that a vortex core lines is located where a vector $\mathbf{v}$ of the vector field is parallel to the directional derivative of the field along $\mathbf{v}$, which is defined by the acceleration $\mathbf{J}(\mathbf{v})\,\mathbf{v}$. In fact, the criterion is almost a straightforward application of the PEV operator we presented in Chapter 7. However, the second tensor field $\nabla_{\mathbf{r}}\mathbf{T}$ is dependent on the unknown $\mathbf{r}$, which requires a slightly different solution strategy.

8.1.2 Mathematical Properties

We now examine the mathematical properties of tensor core lines. We show that they are indeed structurally stable lines, and that in linear tensor fields, these lines are always straight.

Theorem 4. *In a C^1-continuous tensor field, tensor core lines are structurally stable line structures that are either closed or end at the domain boundary.*

To show Theorem 4, we consider a local representation of a real eigenvector field in the neighborhood of a point as a normalized vector field. Then the fact that the PV operator gives such line structures [19] shows the theorem. Note that although such a representation of an eigenvector field as a normalized vector field is locally possible, it does not apply globally in a consistent way, and therefore does not provide a strategy to extract tensor core lines. We also mention that in real datasets, tensor cores may build surfaces or even volumetric structures. This is due to shape symmetries often observed in artificial data produced by humans. Even though these structures are unstable (adding noise destroys the surfaces to many lines), our extraction algorithm has to deal with them.

Theorem 5. *In linear tensor fields, tensor core lines are straight lines.*

This follows from the fact that for linear tensor fields, the linear approximation in (8.1) describes the whole data set exactly: if (8.2) holds for a small h, it holds for all h (i.e., on a whole straight line) as well. Figure 8.2 shows an example of a random linear tensor field containing seven isolated tensor core lines.

8.2 Extracting Tensor Core Lines from Piecewise Linear Data

As we have mentioned in Section 8.1.1, the definition of tensor core lines is almost a straightforward application of the PEV operator, except for the unknown $\mathbf{r}$ in the second tensor field. We can therefore not directly use the algorithm for extracting PEV lines we presented in Section 7.3. Instead we derive a different, more generic, algorithm that is based on the same principles. In fact, this algorithm can be used to extract PEV lines and degenerate lines as well, as we point out in Section 8.4.

Analogous to our PEV extractor, we assume tetrahedral partition of the three-dimensional domain and restrict our search to the boundaries between tetrahedral cells. The intersections of tensor core lines with these triangles are again isolated points, which we extract using a root finding algorithm.

8.2.1 General Algorithm

A tensor core line consists of all locations $\mathbf{x}$ where $\mathbf{T}(\mathbf{x})\mathbf{r} \parallel \mathbf{r}$ and $\nabla_{\mathbf{r}}\mathbf{T}(\mathbf{x})\mathbf{r} \parallel \mathbf{r}$, see (8.3). If two vectors are parallel, their cross product is zero. We can therefore express the criterion as solutions to the system of equations

$$\begin{aligned}(\mathbf{T}(\mathbf{x})\mathbf{r}) \times \mathbf{r} &= \mathbf{0} \\ (\nabla_{\mathbf{r}}\mathbf{T}(\mathbf{x})\mathbf{r}) \times \mathbf{r} &= \mathbf{0}\,.\end{aligned} \tag{8.4}$$

This system consists of six polynomial equations in the unknown variables $\mathbf{x}$ and $\mathbf{r} \neq \mathbf{0}$.

The polynomials are of maximum degree 1 in $\mathbf{x}$ and 3 in $\mathbf{r}$. Note that for a linear tensor field, $\nabla_{\mathbf{r}}\mathbf{T}(\mathbf{x}) = \nabla_{\mathbf{r}}\mathbf{T}$ is constant and thus independent of $\mathbf{x}$. Like for the PEV algorithm, we parameterize the space of possible eigenvector directions $\mathbf{r}$ such that they are defined w.r.t. triangles covering a hemisphere. With the restriction of the local search space to triangles that bound tetrahedra, the solution of the polynomial system is equivalent to finding isolated (real) roots, i.e., points where all six polynomials simultaneously become zero.

We again construct a recursive subdivision algorithm that uses the Bernstein-Bézier form (Section 8.2.2) of the polynomials to exclude the presence of roots within sub-triangles. Within the search space (Section 8.2.3) a recursive subdivision (Section 8.2.4) approximates the locations of roots in $\mathbf{x}$-$\mathbf{r}$-space with an arbitrary user-defined precision. The local feature extraction is followed by a stage that clusters solutions and then connects feature points to lines within cells (Section 8.2.5) Finally, we use a filtering criterion for removing numerically unstable solutions (Section 8.2.6).

8.2.2 Polynomial System in Bernstein-Bézier Form

We consider bivariate polynomials $p : \mathbb{R}^2 \to \mathbb{R}$ of degree n, which are evaluated in a triangular domain $\Delta \subset \mathbb{R}^2$. Let indices $i, j, k \geq 0$. We write $p(\mathbf{u})$ in Bernstein-Bézier form as

$$p(\mathbf{u}) = \sum_{i+j+k=n} B^n_{ijk}(\mathbf{u})\, b_{ijk}\,, \quad B^n_{ijk}(\mathbf{u}) = \frac{n!}{i!j!k!}\, a_1^i\, a_2^j\, a_3^k \tag{8.5}$$

with the bivariate Bernstein polynomials B^n_{ijk} as basis and coefficients (or Bézier control points) b_{ijk}. The basis is defined w.r.t. the barycentric coordinates $a_\ell := a_\ell(\mathbf{u})$, the linear polynomials that satisfy

$$a_1\mathbf{u}_1 + a_2\mathbf{u}_2 + a_3\mathbf{u}_3 = \mathbf{u} \quad \text{and} \quad a_1 + a_2 + a_3 = 1$$

w.r.t. triangle Δ spanned by vertices $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3$ [169].

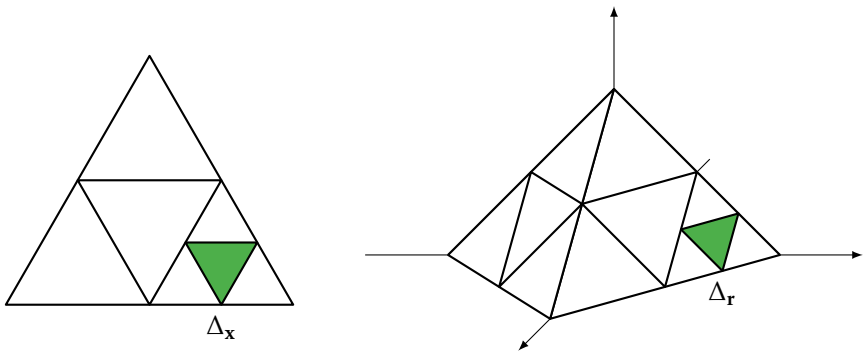

Figure 8.3: The search space is parameterized on triangles $\Delta_{\mathbf{x}}$ in position space and $\Delta_{\mathbf{r}}$ on the hemisphere of possible eigenvector directions. The figure shows a pair of triangles after several subdivision steps.

We again use the *convex hull property* of Bernstein-Bézier polynomials: For any $\mathbf{u} \in \Delta$ – or equivalently all barycentric coordinates are nonnegative – the value $p(\mathbf{u})$ is bounded by the convex hull of the control points b_{ijk}. For scalar coefficients $b_{ijk} \in \mathbb{R}$ this means that if either all $b_{ijk} > 0$ or all $b_{ijk} < 0$ for $i + j + k = n$ then p *cannot* have a zero crossing (or real root) on Δ. Similar to our PEV algorithm, we use this criterion for an iterative subdivision of a triangle Δ into smaller and smaller triangles that either may contain or cannot contain a root.

8.2.3 Parameterization of the Search Space

The equations in the system (8.4) are polynomials in $\mathbf{x}$ and $\mathbf{r}$. This means the search space consists of two independent domains: position and direction.

As pointed out before, positions $\mathbf{x}$ are restricted to points on triangles bounding tetrahedral cells. For each triangle of a tetrahedron we represent positions $\mathbf{x}$ in barycentric coordinates w.r.t. this triangle. Barycentric coordinates are defined in *local coordinates* of the triangle and therefore have two degrees of freedom. After switching to barycentric coordinates the further steps, polynomial evaluation and subdivision, are independent of the supporting triangle.

We again represent the space of possible eigenvector directions $\mathbf{r}$ as a triangulation of a hemisphere. Just like for the PEV algorithm, the orientation and magnitude of $\mathbf{r}$ is irrelevant for the solution and can be approximated by any triangulation that covers all relevant directions. We simply use a four-sided open pyramid (see Figure 8.3). This way, we use the same parameterization and the same representation for $\mathbf{x}$ and $\mathbf{r}$.

For a given triangle, we have to consider a tensor field $\mathbf{T}$ that is linear

in $\mathbf{x}$ and a direction vector $\mathbf{r}$ that is linearly interpolated on a triangle of the "hemisphere". Then the left-hand-sides of the system (8.4) give three polynomials that are linear in $\mathbf{x}$ and quadratic in $\mathbf{r}$ and three polynomials that are cubic in $\mathbf{r}$ as they don't depend on $\mathbf{x}$ because $\nabla_{\mathbf{r}}\mathbf{T}$ is constant.

Each of these six polynomials can be written in the form

$$p(\mathbf{x},\mathbf{r}) = \sum_{\substack{i+j+k=1 \\ \alpha+\beta+\gamma=3}} B^1_{ijk}(\mathbf{x})\, B^3_{\alpha\beta\gamma}(\mathbf{r})\, b_{ijk\alpha\beta\gamma} \,. \tag{8.6}$$

This is the tensor product of the interpolation in $\mathbf{x}$ and $\mathbf{r}$. It gives $3 \times 10 = 30$ coefficients $b_{ijk\alpha\beta\gamma}$. (All indices are nonnegative. Latin indices indicate position space, Greek indices denote direction space.) This form has degree 4 and can represent all polynomials in system (8.4). We use this unified representation for didactic purposes. Using the real number of degrees in $\mathbf{x}$ and $\mathbf{r}$ for each polynomial gives a smaller number of coefficients (18 or 10). It is advisable to use these real degrees in an implementation to avoid superfluous computations. Note that position and direction are expressed in *local coordinates* (or barycentric coordinates) of the triangles, such that p has only four degrees of freedom in total. For the sake of a concise notation we write $p(\mathbf{x},\mathbf{r})$, $B^1_{ijk}(\mathbf{x})$ and $B^3_{\alpha\beta\gamma}(\mathbf{r})$.

8.2.4 Root Finding by Subdivision

Algorithms for finding roots of Bézier curves and surfaces are typically based on the convex hull property and use a recursive bisection [169, 170]. We adopt this technique. The main differences in our setting are the fact that all six equations in (8.4) must be satisfied simultaneously and that this is checked in two different two-dimensional domains: position and space.

Roots of One Single Polynomial

The system (8.4) defines six polynomials. For the initialization of the algorithm, we need to determine the Bernstein-Bézier form, i.e., the coefficients $b_{ijk\alpha\beta\gamma}$ in (8.6), for each of these polynomials. This can be done easily by sampling and interpolation: There are 3×10 coefficients and two triangular parameter domains. As (8.6) lives in a tensor-product space, interpolation of position and direction can be separated. Chose 3 or 10 distinct sampling positions in $\Delta_{\mathbf{x}}$ or $\Delta_{\mathbf{r}}$, respectively, and evaluate the values for the given polynomial. Then interpolate these values using the Bernstein-Bézier basis. The interpolation conditions define a linear system that has a unique solution. We remark that the choice of sampling positions can be arbitrary as long as they are distinct. For polynomial degree n, we use the domain points $\frac{1}{3}(i,j,k)$ with

$i+j+k=n$ in barycentric coordinates. This ensures a well-conditioned system matrix. As the sampling positions are fixed, the system matrix is constant, i.e., interpolation requires only inversion or factoring. So after sampling, the conversion to Bernstein-Bézier form reduces to a linear transform than can be expressed as a matrix-multiplication.

The outline of the subdivision algorithm is as follows. We are given a pair $(\Delta_{\mathbf{x}}, \Delta_{\mathbf{r}})$ of position-direction parameter triangles and a polynomial in Bernstein-Bézier form. The coefficients $b_{ijk\alpha\beta\gamma}$ indicate the absence of zero crossing if they are either all positive or all negative. In this case, no root is found and processing of $(\Delta_{\mathbf{x}}, \Delta_{\mathbf{r}})$ stops. If there is any sign change or zeros in the coefficients, there *may* exist roots within the parameter space $(\Delta_{\mathbf{x}}, \Delta_{\mathbf{r}})$. In this case we *subdivide* one of the parameter triangles. We alternate between subdividing $\Delta_{\mathbf{x}}$ in position-space and $\Delta_{\mathbf{r}}$ in direction-space. For both, we use a regular 1:4-split (see Figure 8.3). Each of the four new sub-triangles is processed recursively in the same way.

Similar to the initial interpolation, the Bernstein-Bézier form of the subdivided polynomial can be obtained by a linear transformation: Evaluate the polynomial at domain points in the new, smaller triangle and apply interpolation. Evaluation and interpolation can be combined to one linear transformation of the Bernstein-Bézier coefficients for each of the four new triangles.

System of Polynomials

We explained the basic algorithm for a single polynomial. Solving system (8.4) means finding solutions $\mathbf{x}, \mathbf{r}$ where all six polynomials become zero simultaneously. We test each polynomial *individually*. Only if there is a sign change in the coefficients for *all* polynomials, a simultaneous root can exist.

Every level of subdivision restricts the parameter domain and thus puts tighter bounds on the region that (potentially) contains a root. The subdivision stops either if there cannot be any root in $(\Delta_{\mathbf{x}}, \Delta_{\mathbf{r}})$ or when the magnitude of all polynomials drops below a threshold (see below). In the latter case, we have found a root, and the barycenters of the triangles define its location in parameter space.

Solution and Stopping Criterion

All computations involving Bernstein-Bézier polynomials are done in barycentric coordinates, which yields a concise implementation of the algorithm. However, the barycentric coordinates are relative to the current triangle, and we still need to keep track of the absolute positions of its vertices in parameter space for bounding the regions of roots. This can be done with a small amount

of bookkeeping by tracking the subdivision steps such that any "child" triangle can be reconstructed from its "parent" and ultimately from the initial parameter triangle.

We stop the subdivision when we are close enough to a root. We require the magnitude of all polynomials to drop below a threshold simultaneously. For each polynomial its magnitude is bounded in $(\Delta_{\mathbf{x}}, \Delta_{\mathbf{r}})$, and we test

$$|p| \; < \; \varepsilon_{\mathrm{t}} \cdot \|\mathbf{T}\|_{\infty} \; , \tag{8.7}$$

where $|p| := \max\{|b_{ijk\alpha\beta\gamma}|\}$ is an upper bound for the magnitude of p in $(\Delta_{\mathbf{x}}, \Delta_{\mathbf{r}})$. The magnitude of the tensor field in $\Delta_{\mathbf{x}}$ is given as

$$\|\mathbf{T}\|_{\infty} = \sup_{\mathbf{x}\in\Delta_{\mathbf{x}}}\{\|\mathbf{T}(\mathbf{x})\|\} = \max_{i}\{\|\mathbf{T}_i\|\} \; , \tag{8.8}$$

where $\mathbf{T}_i$ denote the constant tensors at the triangle vertices $i = 1, 2, 3$, and $\|\mathbf{T}_i\|$ denotes their spectral norm.

The parameter ε_{t} defines a *relative* threshold, which is independent of the local magnitude $\|\mathbf{T}\|_{\infty}$ of the tensor field within $\Delta_{\mathbf{x}}$.

Breadth-First Search Modification

As we hinted at in Section 8.1, the symmetries and regular shapes inherent to data for man-made objects such as stress simulations of mechanical parts often lead to (8.3) being fulfilled or almost fulfilled on surface- or volume-type structures. Also there may be tensor core lines which do not intersect but which are part of the domain triangle. In these cases the roots are no longer isolated points but algebraic structures. As a consequence, the presented algorithm would not be efficient, as it would do an exhaustive search for all "points" on the structure. In cases where the higher-order structures are disturbed by noise and break down to lines, the algorithm still has to do a large number of subdivisions before reaching a termination criterion.

A simple modification of the algorithm enables detecting such cases: we apply a breadth-first search when looking for roots. In the implementation, we use a queue of pairs $(\Delta_{\mathbf{x}}, \Delta_{\mathbf{r}})$ of parameter regions with potential roots. If the number of elements in the queue exceeds a threshold M, we assume that the solution to (8.4) forms an algebraic structure and terminate the local search. If the search is terminated for one of the initial triangles $\Delta_{\mathbf{r}}$ that tessellate the hemisphere, we still need to consider the other triangles, because they may still contain isolated solutions.

8.2.5 Clustering and Line Connection

The root finding returns a list of small parameter regions $(\Delta_{\mathbf{x}}, \Delta_{\mathbf{r}})$, which are assumed to contain a solution to (8.4). The size of these triangles is steered

by the threshold ε_t. Like for the PEV algorithm described in Section 7.3.3, the algorithm typically returns multiple regions that all refer to the same solution due to numerical noise. We apply the same post-process here for clustering such regions and selecting a unique representative $(\mathbf{x}, \mathbf{r})$ for each solution. For each cluster, we select the triangle pair as a representative where $\max\{|p_i|\}$ is smallest for all polynomials $i = 1, \ldots, 6$. We select the center points of both triangles in this pair as the solution represented by this cluster.

For each tetrahedral cell of the dataset, we now connect the root points found on its faces by a line segment. Since in piecewise linear fields, tensor lines are always straight within a cell (see Section 8.1), we greedily connect the two points with the smallest difference in eigenvector direction until no more pairs are left. Similar to the vortex core lines by Sujudi and Haimes [18], this gives a set of discontinuous line segments.

8.2.6 Filtering

If the dataset contains not only lines, but also surface- or volume-type structures where tensor field lines are locally straight, our algorithm might still find isolated solutions on these structures. These occur if the unstable structures are disturbed by noise. To filter these spurious points, we measure the numeric stability of a solution given its representative $(\mathbf{x}, \mathbf{r})$ with

$$s(\mathbf{x}, \mathbf{r}) = \left| \det \begin{pmatrix} \frac{\nabla_{\mathbf{r}_2} \mathbf{T}(\mathbf{x})\mathbf{r}}{\|\mathbf{T}(\mathbf{x})\|_\infty} & \frac{\nabla_{\mathbf{r}_1} \mathbf{T}(\mathbf{x})\mathbf{r}}{\|\mathbf{T}(\mathbf{x})\|_\infty} & \mathbf{r} \end{pmatrix} \right|, \tag{8.9}$$

where $\mathbf{r}, \mathbf{r}_1, \mathbf{r}_2$ are orthonormal.

The stability $s(\mathbf{x}, \mathbf{r})$ measures the directional change that the eigenvector $\mathbf{r}$ of the tensor $\mathbf{T}(\mathbf{x})$ experiences when moving orthogonal to the tensor core line. If this change is small, the magnitude of the determinant in s will be small. This is an indicator that the line is numerically unstable. The normalization by $\|\mathbf{T}(\mathbf{x})\|_\infty$ ensures the independence from the scale of the tensor field.

Filtering out lines with small numeric stability s is a post-process that must be done manually by a user. In practice, the distribution of s over all found solutions shows an exponential behavior. In order to facilitate choosing a threshold, we visualize s on a logarithmic scale.

8.3 Results

We applied our algorithm to stress tensor fields obtained from structural mechanics simulations of varying complexity. Swirling structures in stress tensor fields can result from torque induced in part of a structure. As we will show, it is not always intuitive where this will happen in a complex structure

subject to a load or deformation. Computing tensor core lines allows an easy identification of these phenomena. In this section, we present the results of our algorithm on several datasets, we analyze its performance and parameter sensitivity, and we compare our results to the topological skeleton formed by degenerate lines.

8.3.1 Cylinder

In Figure 8.1 we show the stress trajectories resulting from applying a torque around the long axis of a cylinder. The yellow line visible in the center is the result of our algorithm applied to this case after filtering out numerically unstable solutions. These solutions occur because the third eigenvector, which is orthogonal to the other two, points outwards from the center line everywhere in the domain. This means that the trajectories of this eigenvector are straight lines everywhere inside the cylinder. Situations like this are common in stress tensor fields, and are handled in our algorithm by the threshold M. Nevertheless, single line segments with low numeric stability s may occur due to noise (see Figure 8.9). After filtering them out, the clear central line visible in Figure 8.1 remains.

8.3.2 Handle

This case shows a handle-like structure with a right angle being deformed in two different ways. One end is fixed, while the other end experiences different displacements. The first is a rotation around the shaft, which applies a torque to it. The second includes an additional downward shift. Figure 8.4 shows a tensor core line in the center of the shaft for both cases. Interestingly, a line is also visible in the "handle"-part of the structure, even though no direct torque was applied here. The line in the handle shifts away from the plane of symmetry in the second deformation case. A look at the tensor field around the core lines confirms that they are indeed the center of a swirling behavior of the tensor field.

8.3.3 Truck Bumper

This case shows a load applied to the extreme end of the bumper of a cargo truck. Applying our algorithm to the dataset results in a large number of lines being found all over the domain. This may in part be explained by the low resolution of the simulation. After applying a filter on the numeric stability s, two lines with high stability stand out. Somewhat counterintuitively, these are found on the side opposite to the end experiencing the load. In Figure 8.5, we can clearly see the radial behavior of the tensor field around both lines.

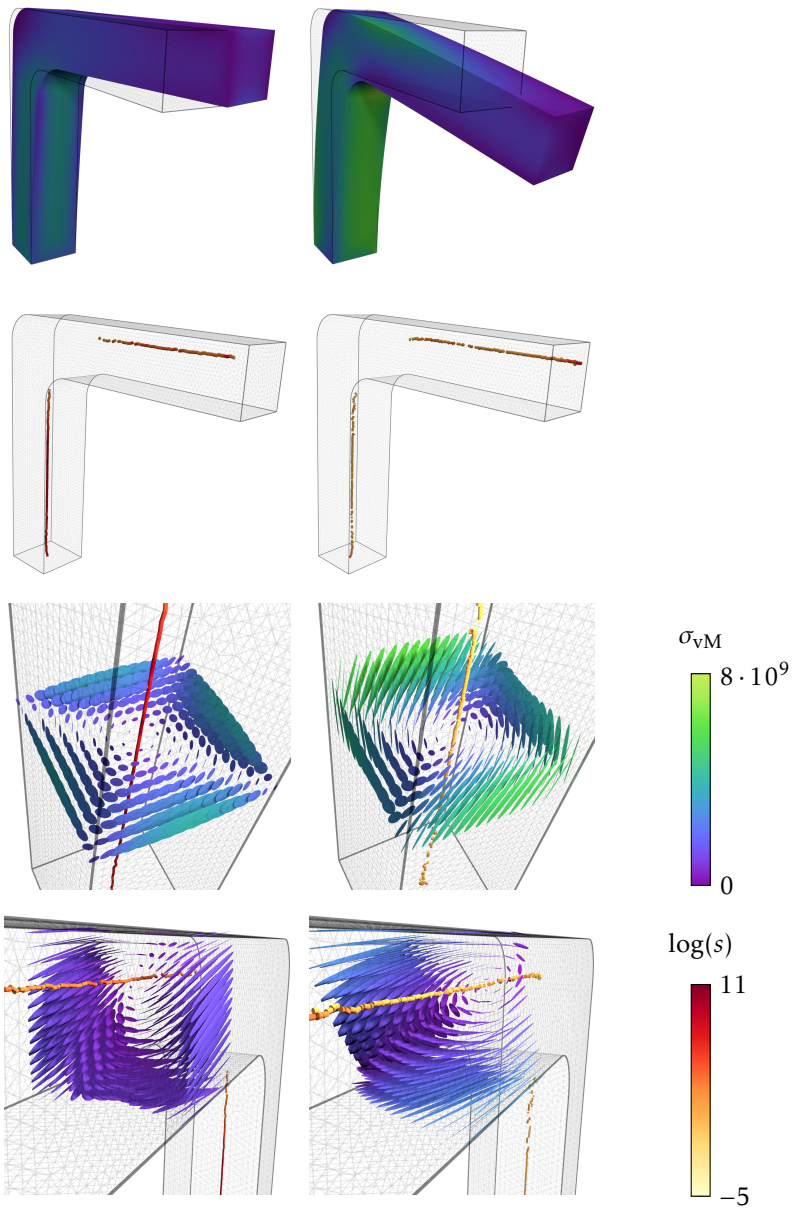

Figure 8.4: Tensor core lines for two different deformations in the HANDLE dataset. On the top, we show the resulting deformations. The von Mises stress σ_{vM} is color-coded on the surface. We represent the tensor core line as tubes in the undeformed coordinate system. Their color indicates the numerical stability s. The tensor field is shown for context using elliptical glyphs.

Finding these locations by manually inspecting the tensor field in detail would be a tedious task. Using the tensor core line extractor, they can be identified at a glance.

8.3.4 Crane

In this dataset, the arm of a crane is exposed to a downward pull applied to the lower side of a cube at the end (see Figure 8.6). Similar to the truck bumper, it is not intuitively clear in which parts of the structure a swirling behavior of the tensors will occur, just from looking at the setup of the case. Almost all stable solutions we find are located in the diagonal rods on the lower side. Again, looking closely at the tensors around the core lines, we can see the radial behavior.

8.3.5 Spring

A simulation of a coil spring being compressed and slightly bent between two plates is shown in Figure 8.7. Apart from numerical noise in the poorly resolved plates, we find significant tensor core lines at the center of the coil's cross-section. A look at the tensor field visualized by glyphs reveals that in this case, we do not have a simple swirling behavior of the tensors. Instead, the tensor field shows something similar to a hyperbolic behavior in vector fields. In the rightmost picture in Figure 8.7, we can see that eigenvector trajectories start at the wall on both sides and curve into the same direction. This direction is reversed on the top and bottom side of the spring. In the middle, there is a surface where these curves become straight lines along the diameter of the cross-section. This is exactly where we find a tensor core line.

8.3.6 Performance and Parameter Study

We tested our algorithm using a consumer PC with a 4-core Intel Core i7 CPU at 3.4 GHz. Our implementation is parallelized over the faces of the dataset using OpenMP [171]. Performance numbers for the different datasets shown in this paper are presented in Table 8.1.

The performance of our algorithm is dependent on the dataset. If we find a large number of tensor core lines in the dataset, computation will be slower as fewer cells can be discarded early. To examine the dependence of the performance and results of our algorithm on the parameters M, ε_t and ε_c, we conducted a parameter study. We selected baseline parameters $M = 10^3$, $\varepsilon_t = 10^{-6}$ and $\varepsilon_c = 10^{-3}$. We then varied each parameter separately and applied our algorithm to the cylinder dataset. The results can be seen in Figure 8.8. We can see that the performance is controlled by the threshold

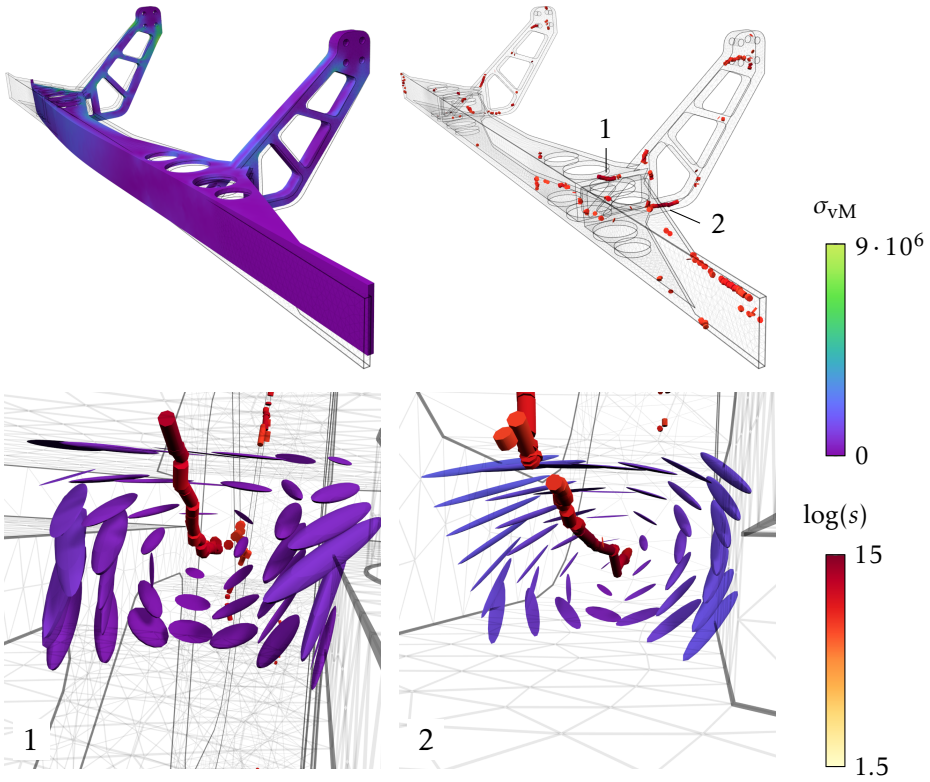

Figure 8.5: Tensor core lines in the TRUCK BUMPER dataset. The deformation shown on the top left is scaled by a factor of 500 for illustrative purposes. The bottom shows detail views of two interesting lines with tensor glyphs for context.

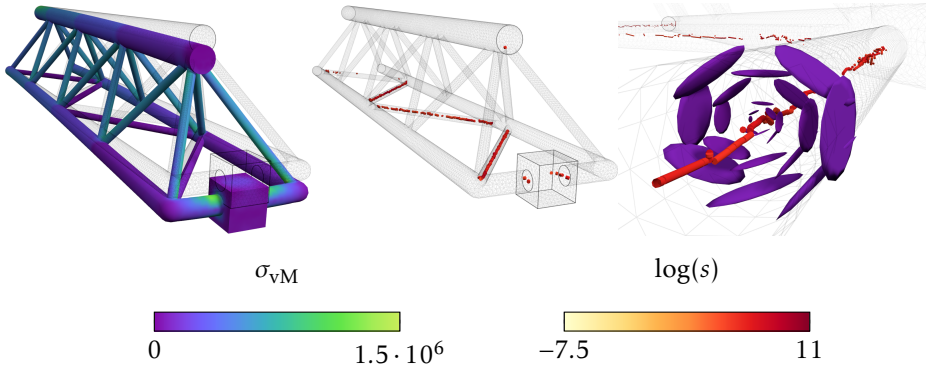

Figure 8.6: Tensor core lines in the CRANE dataset. The resulting deformation on the left is scaled by a factor of 1500.

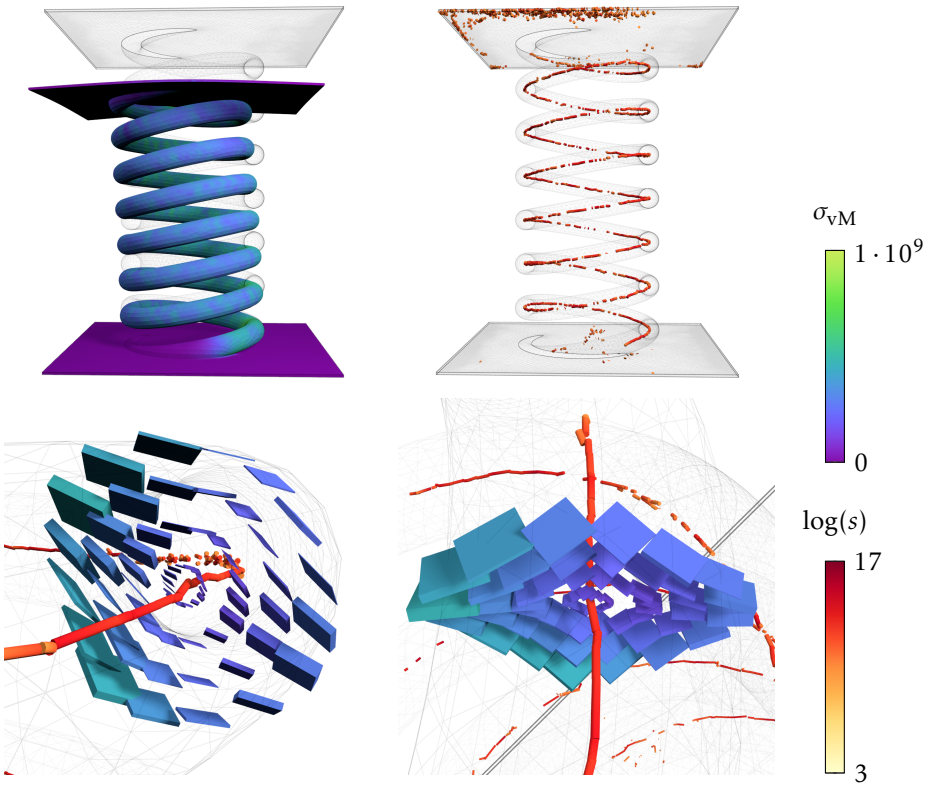

Figure 8.7: Tensor core lines in the Spring dataset. We visualize the tensors near the core line with box glyphs in this case. They make it easier to see the hyperbolic behavior of the eigenvectors that occurs in the coil's cross-section.

Table 8.1: Performance of the algorithm for the datasets presented in this paper.

Dataset	# of cells/10^3	time/s	avg. time per face/ms
Cylinder	65	8	0.034
Handle	235	36	0.038
Truck Bumper	97	32	0.081
Crane	108	63	0.146
Spring	181	82	0.114

Figure 8.8: Run times ( ) and number of found lines in the Cylinder dataset for various different parameters. We show the total number of lines found ( ) and the number of lines remaining after filtering out numerically unstable lines ( ).

M, which controls at which point we assume we are not converging onto an isolated solution. Increasing M also increases the number of solutions we find. However, if we look at the number of solutions that remain after filtering based on numeric stability, it becomes clear that these solutions are only caused by noise. Increasing M did not result in any additional numerically stable lines. The parameters ε_t and ε_c have almost no noticeable impact on runtime or solutions, unless we choose unreasonable numbers. In case of ε_t, choosing a value that is larger than 10^{-3} causes an explosion of the number of found solutions, as the tolerance is not tight enough. Choosing ε_c smaller than ε_t means that candidate solutions belonging to the same cluster often can not be clustered, because the search radius is smaller than the distance between the triangle centers. Otherwise, ε_c is very stable. This is because for solutions which are isolated points and belong to different eigenvectors, the separation between clusters in direction space is rather large. This means the choice of ε_c is not critical as long as it is not chosen extremely small, or so large that solutions which belong to different eigenvectors are clustered together.

Stability tests on our other datasets all produced very similar results. We recommend choosing $M = 10^2$, $\varepsilon_t = 10^{-3}$ and $\varepsilon_c = 10^{-2}$ if performance is important. If accuracy is important, we found that choosing stricter tolerances than $M = 10^3$, $\varepsilon_t = 10^{-6}$ and $\varepsilon_c = 10^{-3}$ does not produce noticeably better results.

8.3.7 Comparison with Degenerate Lines

Tensor core lines are mathematically distinct from degenerate lines where two or more eigenvalues are equal. The criterion for finding tensor core lines is completely independent of the eigenvalues of the tensor field. However, when looking at our results in stress tensor fields, one might wonder if tensor core lines coincide with degenerate lines in practice. To investigate this, we extracted degenerate lines from our datasets using the method presented by Zheng and Pang [58]. In stress tensor fields, degenerate lines mark locations where no unique principal directions of stress can be established. We found that tensor core lines and degenerate lines sometimes coincide, but neither is a subset of the other. In the Crane dataset, degenerate lines are found near the center of the lower diagonal rods, where we also find tensor core lines. However, a lot of degenerate lines are also found in regions where no tensor core lines are located. In the Truck Bumper dataset, a degenerate line coincides with one of the two most significant tensor core lines we find, but not the other one. Closeups of both datasets are shown in Figure 8.10. In several datasets, such as the Cylinder and Handle, Zheng and Pang's method fails to locate any distinct degenerate lines at all.

8.4 Computing PEV and Degenerate Lines

The algorithm we have presented for extracting tensor core lines is a generic root finding algorithm adapted to find simultaneous roots of multiple polynomials. Because of its genericity, we can also use it to extract PEV lines and degenerate lines, which can be expressed as root finding problems of the same type. This makes it possible to extract several different feature lines from tensor fields using the same algorithmic framework.

We remember the criterion for PEV lines of two tensor fields $\mathbf{S}$ and $\mathbf{T}$. A PEV line passes through every point $\mathbf{x}$ that satisfies

$$\mathbf{r} \parallel \mathbf{S}(\mathbf{x})\,\mathbf{r} \parallel \mathbf{T}(\mathbf{x})\,\mathbf{r}\,.$$

Again using the fact that the cross product of two parallel vectors is zero, we translate this to the system of polynomials

$$\begin{aligned} (\mathbf{S}(\mathbf{x})\,\mathbf{r}) \times \mathbf{r} &= \mathbf{0} \\ (\mathbf{T}(\mathbf{x})\,\mathbf{r}) \times \mathbf{r} &= \mathbf{0}\,. \end{aligned} \tag{8.10}$$

This system has six polynomials that are quadratic in $\mathbf{r}$ and linear in $\mathbf{x}$, which need to become zero at the same time. We can use the same recursive subdivision algorithm based on the Bernstein-Bézier forms of these polynomials to find solutions.

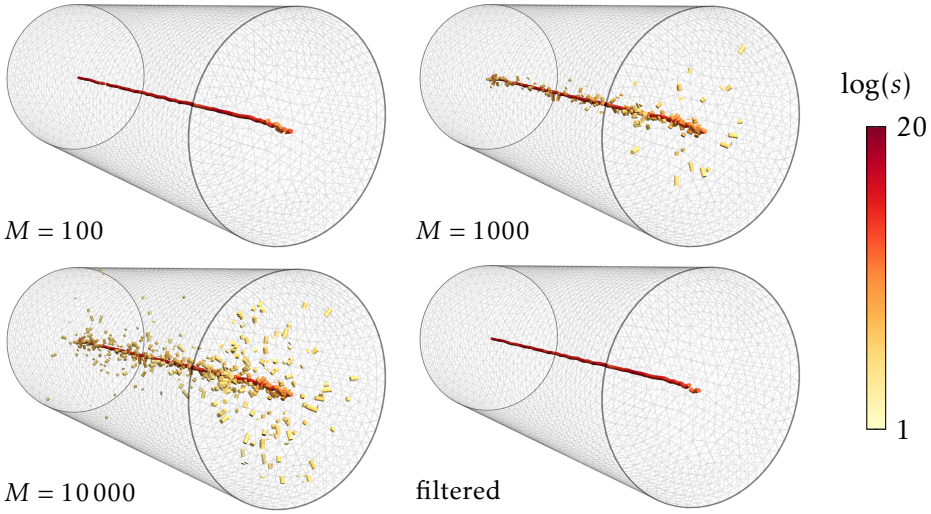

Figure 8.9: Results of our algorithm on the Cylinder dataset for different choices of M. Increasing M results in more numerically unstable lines being found. If we filter them out, the result is virtually identical.

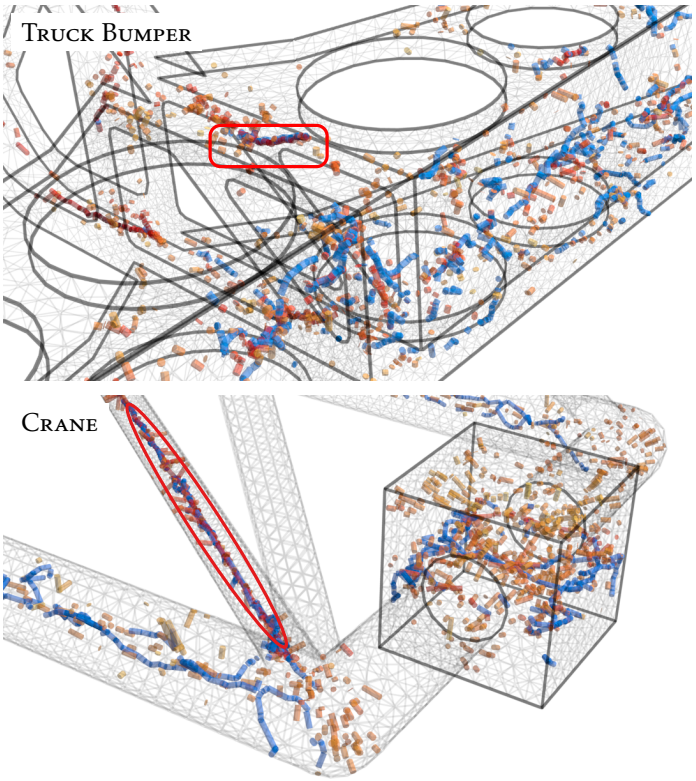

Figure 8.10: Comparison of unfiltered tensor core lines (red/yellow) and degenerate tensor lines (blue) for the Truck Bumper and Crane dataset. The red highlights mark the coincidence of numerically stable tensor core lines with degenerate tensor lines.

Computing PEV lines using this scheme has several advantages. The algorithm is somewhat simpler than the one we presented in Section 7.3, as no two separate nested recursive processes are required. This means that is is simpler to explain and implement. More importantly, using this algorithm does not result in false-positive solution candidates that need to be filtered out. Computing times using this algorithm are however somewhat longer. This is because the separate recursion in $\mathbf{r}$-space that is used in the original PEV algorithm can frequently be terminated early while the subdivision level in $\mathbf{x}$-space is still low. This is not possible in the root finding algorithm we presented here, as it considers both spaces simultaneously.

The extraction of degenerate lines in tensor fields can be expressed as a root finding problem as well. A degenerate line is located where two eigenvalues of a tensor field are equal. Zheng et al. [58] showed that these locations can be expressed as the simultaneous roots of the seven *discriminant constraint functions*

$$
\begin{aligned}
f_x(\mathbf{T}) &= T_{00}(T_{11}^2 - T_{22}^2) + T_{00}(T_{01}^2 - T_{02}^2) + T_{11}(T_{22}^2 - T_{00}^2) + T_{11}(T_{12}^2 - T_{01}^2) \\
&\quad + T_{22}(T_{00}^2 - T_{11}^2) + T_{22}(T_{02}^2 - T_{12}^2) \\
f_{y1}(\mathbf{T}) &= T_{12}\Big(2(T_{12}^2 - T_{00}^2) - (T_{02}^2 + T_{01}^2) + 2(T_{11}T_{00} + T_{22}T_{00} - T_{11}T_{22})\Big) \\
&\quad + T_{01}T_{02}(2T_{00} - T_{22} - T_{11}) \\
f_{y2}(\mathbf{T}) &= T_{02}\Big(2(T_{02}^2 - T_{11}^2) - (T_{01}^2 + T_{12}^2) + 2(T_{22}T_{11} + T_{00}T_{11} - T_{22}T_{00})\Big) \\
&\quad + T_{12}T_{01}(2T_{11} - T_{00} - T_{22}) \\
f_{y3}(\mathbf{T}) &= T_{01}\Big(2(T_{01}^2 - T_{22}^2) - (T_{12}^2 + T_{02}^2) + 2(T_{00}T_{22} + T_{11}T_{22} - T_{00}T_{11})\Big) \\
&\quad + T_{02}T_{12}(2T_{22} - T_{11} - T_{00}) \\
f_{z1}(\mathbf{T}) &= T_{12}(T_{02}^2 - T_{01}^2) + T_{01}T_{02}(T_{11} - T_{22}) \\
f_{z2}(\mathbf{T}) &= T_{02}(T_{01}^2 - T_{12}^2) + T_{12}T_{01}(T_{22} - T_{00}) \\
f_{z3}(\mathbf{T}) &= T_{01}(T_{12}^2 - T_{02}^2) + T_{02}T_{12}(T_{00} - T_{11}),
\end{aligned}
\tag{8.11}
$$

where $\mathbf{T} = \mathbf{T}(\mathbf{x})$ is the tensor field and T_{ij} are the components of the (symmetric) tensor. The advantage of using these functions is that it does not require the explicit computation of eigenvalues. We can find the simultaneous roots of these seven polynomials that are maximum cubic in $\mathbf{x}$ using the same scheme we used to extract tensor core lines. Since these equations do not depend on a direction $\mathbf{r}$, we only need to subdivide in $\mathbf{x}$-space. The degenerate lines shown in Figure 8.10 were computed in this way.

8.5 Discussion

We introduced tensor core lines as a new feature of second-order tensor fields. It enables the quick detection of swirling behavior in tensor field lines. Such behavior might not have a distinct physical meaning in all applications. However, finding core lines helps to understand the structure of the tensor field by breaking down a complex feature into a simple line structure that can be easily visualized. In this regard, our method fits in well with other feature-based visualization methods.

Our method is a direct extension of the Sujudi/Haimes method for the extraction of vortex core lines in vector fields. As such, it shares many of its advantages and drawbacks. The criterion is completely local and does not require integration. As such, it is well parallelizable and not vulnerable to accumulating numerical errors. Still, we are hardly able to reach interactive run times, as we need to perform an exhaustive search in a 5D space. Like Sujudi/Haimes, we perform a search on piecewise linear data, which results in straight lines within cells and discontinuities of the tensor core lines at cell boundaries. Using higher-order interpolation of the tensor field would help finding continuous lines.

We have chosen to focus on piecewise linear tensor fields where each tensor component is interpolated independently. While alternative interpolation schemes have been proposed [172], component-wise interpolation is still widely used as a standard approach for both tensor- and vector fields.

Unlike Sujudi/Haimes, we have no way of explicitly ensuring our solutions show only swirling behavior by restricting them to regions where the derivative has complex eigenvalues. The derivative of the tensor field $\nabla\mathbf{T}$ is a third-order tensor, for which the definition of eigenvalues and eigenvectors is non-trivial [173]. This means that we also find structures similar to hyperbolic trajectories in vector fields [167, 168]. Further research is necessary in order to distinguish these different types of features.

We introduced a measure for the numeric stability of tensor core lines. Unfortunately, filtering out numerically unstable solutions must be done as an interactive post-processing step, as the threshold is different for each dataset. It is worth investigating if this process can be automated. Nevertheless, the measure enables us to distinguish significant and insignificant solutions, which is a very useful tool for assessing the result of our algorithm.

Our algorithm is numerically very stable. We have three free parameters, two of which can be chosen in a wide range without significant influence on the results, as we show in Section 8.3.6. The parameter M, which influences run time the most, can be chosen the same for most datasets and as such does not require fine-tuning either.

Our algorithm is only designed for extracting structurally stable line fea-

tures, but surfaces or regions where the zero curvature criterion is almost fulfilled seem to be common in real-world stress tensor data. This might be due to the common occurrence of symmetries and regular shapes in human-made objects, which are most frequently the focus of structural analysis. It would therefore be interesting to investigate if these structures can explicitly be extracted, possibly by restricting the search space to the edges of tetrahedral cells.

Finally, it is worth noting that neither the formal definition of tensor core lines nor the extraction algorithm poses any restrictions on the tensor field, except that it be differentiable. As such, it might also be used on indefinite tensor data, such as the Jacobian of a vector field. Finding applications outside of stress tensor analysis is a subject for further research.

9

Conclusion

In the previous two chapters, we extend the idea of a class of feature-based visualization techniques for vector- and scalar fields to the realm of tensor fields. In Chapter 7, we establish the PEV operator as a direct extension of the generic PV operator. It finds all locations where two tensor fields have parallel real eigenvectors. Using this, we translate the concept of vortex core lines to their counterpart in tensor fields in Chapter 8. These tensor core lines mark the centers of "swirling" behavior of tensor field lines. Feature lines of this type can be extracted from piecewise linear data by determining their intersections with the boundaries of tetrahedral cells. The search for such intersections is a search for roots of higher-order polynomials, which we solve using a recursive subdivision algorithm based on their Bernstein-Bézier form. The intersections are then connected to lines afterwards.

The work presented in this part of the thesis is basic research into higher-order features in tensor fields. As an application area, we focus on the visualization of stress tensor fields from solid mechanics simulations. In-detail analysis of the topology and structure of stress tensor fields still is not well established within the solid mechanics community. One reason for this might be that tensor fields are even more complex than vector fields, which are already

challenging to visualize and understand. The PEV operator and tensor core lines are additions to the visualization toolbox that help to better understand the complex ways in which forces act in solid materials. Such feature-based techniques break down complex behavior into simple geometric primitives that are more easily parsed and understood. Maybe the development of more such techniques can help to better establish tensor field visualization with solid mechanics researchers.

Appendix

A

Interpolating the Transformation for New Surface Patches

In Section 5.3.3, we explain how to reconstruct the tangential deformation experienced by a micro-patch over a finite time interval $[t_s, t_e]$, if that patch has existed for this whole interval. If instead the patch was created at time $t_k > t_s$ as one of the outer patches during a split operation, we have to estimate the deformation it has experienced until that time from its parent patch. If the deformation across the parent patch was completely uniform, we could just pass this deformation on to the child patches after a split. However, in general there will be slight differences in the transformations which the neighborhoods of each of the four ghost particles have experienced. At the time of the split, we still have this information for the time interval since the last split or merge operation the parent patch was a part of.

Let $\mathbf{x}_i(t_{k-1})$ be the positions of the ghost particles (relative to the central point) at the last reset of the parent patch, or at the start time t_s if the patch was not reset since that time. Let $\mathbf{x}_i(t_k)$ be the ghost particle positions at the time of the split. Then $\mathbf{E}_{t_{k-1}}^{t_k}$ obtained via (5.10) can be thought of as the

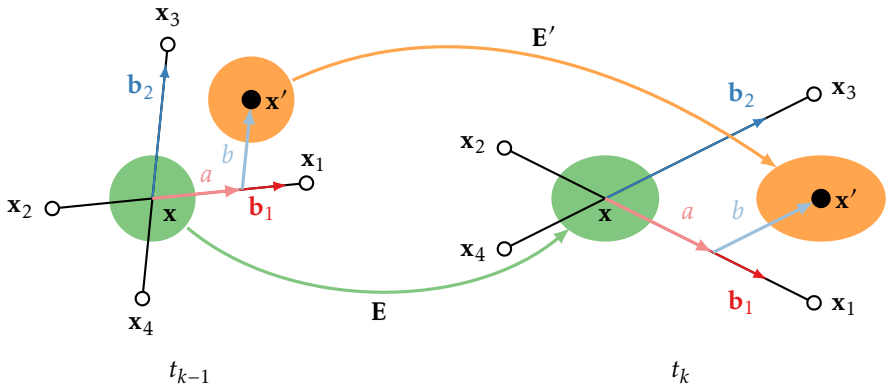

Figure A.1: Interpolating the transformation at an offset position $\mathbf{x}'$. The ghost particles in the direction of $\mathbf{x}'$ have been stretched more than their counterparts during the time interval. Therefore the interpolated $\mathbf{E}'$ has a stronger stretching effect on the local neighborhood of $\mathbf{x}'$ (●) than $\mathbf{E}$ stretches the local neighborhood of $\mathbf{x}$ (●).

solution of the system

$$\begin{pmatrix} \mathbf{b}_1(t_{k-1})^{\mathrm{T}} \\ \mathbf{b}_2(t_{k-1})^{\mathrm{T}} \\ \mathbf{n}(t_{k-1})^{\mathrm{T}} \end{pmatrix} \left(\mathbf{E}_{t_{k-1}}^{t_k}\right)^{\mathrm{T}} = \begin{pmatrix} \mathbf{b}_1(t_k)^{\mathrm{T}} \\ \mathbf{b}_2(t_k)^{\mathrm{T}} \\ \mathbf{n}(t_k)^{\mathrm{T}} \end{pmatrix} \tag{A.1}$$
$$\mathbf{b}_1(t) = (\mathbf{x}_1(t) - \mathbf{x}_2(t))/2$$
$$\mathbf{b}_2(t) = (\mathbf{x}_3(t) - \mathbf{x}_4(t))/2\,.$$

In other words, $\mathbf{E}_{t_{k-1}}^{t_k}$ is the average of the transformations that map the corresponding ghost particles to each other exactly.

For the central point, it makes sense to weight those transformations equally, but if we want to initialize a new patch whose center is slightly offset, we get a more accurate result if we adjust the weights depending on where the new center is located. For this purpose, we parameterize the tangent space of the micro-patch by expressing it as a linear combination of the two basis vectors $\mathbf{b}_1(t)$ and $\mathbf{b}_2(t)$. We can now express the location of the new center $\mathbf{x}'$ in this new basis:

$$\mathbf{x}' = \lambda_1 \mathbf{b}_1(t_k) + \lambda_2 \mathbf{b}_2(t_k)\,.$$

These coordinates correspond directly to the coordinates of $(\mathbf{E}_{t_{k-1}}^{t_k})^{-1}\mathbf{x}'$ at time t_{k-1}:

$$(\mathbf{E}_{t_{k-1}}^{t_k})^{-1}\,\mathbf{x}' = \lambda_1 \mathbf{b}_1(t_{k-1}) + \lambda_2 \mathbf{b}_2(t_{k-1})\,.$$

If λ_1 is positive, i.e., if $\mathbf{x}'$ is located more towards $\mathbf{x}_1(t_k)$ than towards $\mathbf{x}_2(t_k)$, we want $\mathbf{x}_1(t_k)$ to have a stronger influence on the result. The same applies

to the direction of $\mathbf{b}_2$. We therefore compute new interpolated basis vectors $\mathbf{b}'_{1,2}(t)$ by weighting the ghost particle positions with λ_1 and λ_2:

$$\begin{aligned}\mathbf{b}'_1(t) &= (1+2\lambda_1)\mathbf{x}_1(t) - (1-2\lambda_1)\mathbf{x}_2(t)\\ \mathbf{b}'_2(t) &= (1+2\lambda_2)\mathbf{x}_3(t) - (1-2\lambda_2)\mathbf{x}_4(t).\end{aligned} \tag{A.2}$$

The adjusted transformation $\mathbf{E}_{t_{k-1}}^{t_k\,\prime}$ is then the solution to the system

$$\begin{pmatrix}\mathbf{b}'_1(t_{k-1})^{\mathrm{T}}\\ \mathbf{b}'_2(t_{k-1})^{\mathrm{T}}\\ \mathbf{n}(t_{k-1})^{\mathrm{T}}\end{pmatrix}\left(\mathbf{E}_{t_{k-1}}^{t_k\,\prime}\right)^{\mathrm{T}} = \begin{pmatrix}\mathbf{b}'_1(t_k)^{\mathrm{T}}\\ \mathbf{b}'_2(t_k)^{\mathrm{T}}\\ \mathbf{n}(t_k)^{\mathrm{T}}\end{pmatrix}. \tag{A.3}$$

The complete transformation of a micro-patch that has been split off from a parent at some intermediate time t_k is then obtained by

$$\mathbf{E}_{t_\mathrm{s}}^{t_\mathrm{e}} = \mathbf{E}_{t_n}^{t_\mathrm{e}} \cdot \mathbf{E}_{t_{n-1}}^{t_n} \cdot \cdots \cdot \mathbf{E}_{t_k}^{t_{k+1}} \cdot \mathbf{E}_{t_{k-1}}^{t_k\,\prime} \cdot \mathbf{E}_{t_\mathrm{s}}^{t_{k-1}}. \tag{A.4}$$

Here, $\mathbf{E}_{t_\mathrm{s}}^{t_{k-1}}$ is the transformation of the parent patch from the start of the interval up to the time when its ghost particles were last reset before the split operation at t_k. If $t_\mathrm{s} > t_{k-1}$, it is omitted. $\mathbf{E}_{t_{k-1}}^{t_k\,\prime}$ is the estimated transformation at a point with a slight offset from the center of the parent patch in the interval before the split. Earlier transformations are inherited as-is from the parent patch.

B

Proof that the Parallel Eigenvectors Operator Yields Structurally Stable Curves

In Section 7.2, we formulated Theorem 1, which states that the PEV operator yields structurally stable curves that are either closed or end at the domain boundary. We give the proof for this theorem here. This proof was derived and written by Holger Theisel.

The main idea to prove Theorem 1 is to search for PEV lines not in 3D (x,y,z) space but in a 6D (x,y,z,u,v,w) space: at every point $\mathbf{x} = (x,y,z)^T$, all vector directions $\mathbf{r} = (u,v,w)^T$ are checked for being an eigenvector of $\mathbf{S}$ and $\mathbf{T}$. This means that we search for all 6D points $(\mathbf{x},\mathbf{r})^T$ fulfilling $\mathbf{S}(\mathbf{x})\mathbf{r} \times \mathbf{r} = \mathbf{0}$ and $\mathbf{T}(\mathbf{x})\mathbf{r} \times \mathbf{r} = \mathbf{0}$. We formulate this as a search for all 6D points $(\mathbf{x},\mathbf{r})^T$ where a 6D vector field $\overline{\mathbf{h}}$ vanishes:

$$\overline{\mathbf{h}}(\mathbf{x},\mathbf{r}) = \begin{pmatrix} \mathbf{S}(\mathbf{x})\mathbf{r} \times \mathbf{r} \\ \mathbf{T}(\mathbf{x})\mathbf{r} \times \mathbf{r} \end{pmatrix} = \overline{\mathbf{0}}, \tag{B.1}$$

where $\overline{\mathbf{0}}$ is the zero vector in 6D.

Suppose a point $(\mathbf{x}_0,\mathbf{r}_0)^T$ is on a PEV structure, i.e., fulfills Equation (B.1). In order to study the PEV structures in a linear neighborhood of $(\mathbf{x}_0,\mathbf{r}_0)^T$, we

search for all directions $(d\mathbf{x}, d\mathbf{r})^{\mathrm{T}}$ in which $\overline{\mathbf{h}}$ remains zero: $\nabla\overline{\mathbf{h}} \cdot (d\mathbf{x}, d\mathbf{r})^{\mathrm{T}} = \overline{\mathbf{0}}$. In other words: we have to explore the null space of $\nabla\overline{\mathbf{h}}$. Applying elementary differentiation rules gives

$$\nabla\overline{\mathbf{h}} = \begin{pmatrix} \mathbf{G}_1 & \mathbf{G}_3 \\ \mathbf{G}_2 & \mathbf{G}_4 \end{pmatrix}, \tag{B.2}$$

with

$$\begin{aligned} \mathbf{G}_1 &= \begin{pmatrix} \mathbf{S}_x\mathbf{r}\times\mathbf{r} & \mathbf{S}_y\mathbf{r}\times\mathbf{r} & \mathbf{S}_z\mathbf{r}\times\mathbf{r} \end{pmatrix}, \\ \mathbf{G}_2 &= \begin{pmatrix} \mathbf{T}_x\mathbf{r}\times\mathbf{r} & \mathbf{T}_y\mathbf{r}\times\mathbf{r} & \mathbf{T}_z\mathbf{r}\times\mathbf{r} \end{pmatrix}, \\ \mathbf{G}_3 &= \begin{pmatrix} \mathbf{S}\mathbf{e}_1\times\mathbf{r}+\mathbf{S}\mathbf{r}\times\mathbf{e}_1 & \mathbf{S}\mathbf{e}_2\times\mathbf{r}+\mathbf{S}\mathbf{r}\times\mathbf{e}_2 & \mathbf{S}\mathbf{e}_3\times\mathbf{r}+\mathbf{S}\mathbf{r}\times\mathbf{e}_3 \end{pmatrix}, \\ \mathbf{G}_4 &= \begin{pmatrix} \mathbf{T}\mathbf{e}_1\times\mathbf{r}+\mathbf{T}\mathbf{r}\times\mathbf{e}_1 & \mathbf{T}\mathbf{e}_2\times\mathbf{r}+\mathbf{T}\mathbf{r}\times\mathbf{e}_2 & \mathbf{T}\mathbf{e}_3\times\mathbf{r}+\mathbf{T}\mathbf{r}\times\mathbf{e}_3 \end{pmatrix}, \end{aligned} \tag{B.3}$$

where $\mathbf{S}_{x,y,z}$ and $\mathbf{T}_{x,y,z}$ are the partial derivatives of the tensor fields and $\mathbf{e}_i$ are unit vectors along the coordinates. Then

$$\mathbf{G}_1{}^{\mathrm{T}}\mathbf{r} = \mathbf{G}_2{}^{\mathrm{T}}\mathbf{r} = 0, \tag{B.4}$$

and from Equation (B.1) follows

$$\mathbf{G}_3{}^{\mathrm{T}}\mathbf{r} = \mathbf{G}_4{}^{\mathrm{T}}\mathbf{r} = 0, \tag{B.5}$$

and

$$\mathbf{G}_3\mathbf{r} = \mathbf{G}_4\mathbf{r} = 0. \tag{B.6}$$

Equations (B.4) and (B.5) give that

$$\operatorname{rank}\left(\nabla\overline{\mathbf{h}}\right) = 4 \tag{B.7}$$

in the structurally stable case. This means that for $\operatorname{rank}\left(\nabla\overline{\mathbf{h}}\right) < 4$, adding noise to $\mathbf{S}, \mathbf{T}$ brings $\operatorname{rank}\left(\nabla\overline{\mathbf{h}}\right)$ to 4. Equation (B.7) means that the PEV structure around $(\mathbf{x}_0, \mathbf{r}_0)^{\mathrm{T}}$ is a 2-manifold in 6D. To see Equation (B.7), we consider a rotation of the underlying coordinate system such that $\mathbf{r} = (0, 0, r_z)$. Then Equations (B.4) and (B.5) give that the rotated tensors $\mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_3, \mathbf{G}_4$ have vanishing third columns. This and Equation (B.1) gives that $\nabla\overline{\mathbf{h}}$ has two columns, which proves Equation (B.7).

One vector in the null space of $\nabla\overline{\mathbf{h}}$ is trivial and denotes a simple scaling of $\mathbf{r}$: Equations (B.4) to (B.6) give $\nabla\overline{\mathbf{h}} \cdot (\mathbf{0}, \mathbf{r})^{\mathrm{T}} = \overline{\mathbf{0}}$. This means that the projection of the null space of $\nabla\overline{\mathbf{h}}$ into the spatial subspace $\mathbf{x}$ gives a one-manifold in 3D. This shows that PEV gives line structures in 3D. To show that they are closed, we consider the 6 components of $\nabla\overline{\mathbf{h}}$ as scalar fields and interpret the

PEV structure as intersection of their 5D iso-hypersurfaces. Iso-hypersurfaces are always closed, which means their intersections are also closed.

Note that the proof did not make any assumptions on the behavior of $\mathbf{S}, \mathbf{T}$ around $(\mathbf{x}_0, \mathbf{r}_0)^{\mathrm{T}}$. This means that it holds also in case of a transition from real to imaginary eigenvectors of $\mathbf{S}$ or $\mathbf{T}$ as well as in regions of isotropic tensors.

Bibliography

[1] A. C. Telea. *Data Visualization*. 2nd ed. A K Peters/CRC Press, 2014 (cit. on p. 10).

[2] M. Levoy. "Display of Surfaces from Volume Data". In: *Computer Graphics and Applications* 8.3 (1988), pp. 29–37 (cit. on p. 11).

[3] R. A. Drebin, L. Carpenter, and P. Hanrahan. "Volume Rendering". In: *ACM SIGGRAPH Computer Graphics* 22.4 (1988), pp. 65–74 (cit. on p. 11).

[4] G. Kindlmann and J. W. Durkin. "Semi-Automatic Generation of Transfer Functions for Direct Volume Rendering". In: *IEEE Symposium on Volume Visualization*. IEEE Computer Society, 1998, pp. 79–86 (cit. on p. 11).

[5] R. Peikert and F. Sadlo. "Height Ridge Computation and Filtering for Visualization". In: *IEEE Pacific Visualization Symposium (PacificVis)*. IEEE Computer Society, 2008, pp. 119–126 (cit. on pp. 12, 13).

[6] D. Eberly. *Ridges in Image and Data Analysis*. Vol. 7. Springer Science & Business Media, 2012 (cit. on p. 12).

[7] T. Weinkauf. "Extraction of Topological Structures in 2D and 3D Vector Fields". PhD thesis. Otto-von-Guericke Universität Magdeburg, 2008 (cit. on p. 14).

[8] B. Cabral and L. C. Leedom. "Imaging Vector Fields Using Line Integral Convolution". In: *Proc. of SIGGRAPH*. ACM, 1993, pp. 263–270 (cit. on p. 15).

[9] C. Rezk-Salama, P. Hastreiter, C. Teitzel, and T. Ertl. "Interactive Exploration of Volume Line Integral Convolution Based on 3D-Texture Mapping". In: *IEEE Conference on Visualization (VIS)*. IEEE Computer Society, 1999, pp. 233–528 (cit. on p. 15).

[10] H.-W. Shen and D. L. Kao. "UFLIC: A Line Integral Convolution Algorithm for Visualizing Unsteady Flows". In: *IEEE Conference on Visualization (VIS)*. IEEE Computer Society, 1997, pp. 317–322 (cit. on p. 15).

[11] J. J. van Wijk. "Spot Noise Texture Synthesis for Data Visualization". In: *ACM SIGGRAPH Computer Graphics* 25.4 (1991), pp. 309–318 (cit. on p. 15).

[12] W. C. de Leeuw and J. J. van Wijk. "Enhanced Spot Noise for Vector Field Visualization". In: *IEEE Conference on Visualization (VIS)*. IEEE Computer Society, 1995, pp. 233–239 (cit. on p. 15).

[13] T. Gerrits, C. Rössl, and H. Theisel. "An Approximate Parallel Vectors Operator for Multiple Vector Fields". In: *Computer Graphics Forum* 37.3 (2018), pp. 315–326 (cit. on pp. 17, 116).

[14] J. L. Helman and L. Hesselink. "Visualizing Vector Field Topology in Fluid Flows". In: *Computer Graphics and Applications* 11.3 (1991), pp. 36–46 (cit. on pp. 19, 20).

[15] A. Surana, O. Grunberg, and G. Haller. "Exact theory of three-dimensional flow separation. Part 1. Steady separation". In: *Journal of Fluid Mechanics* 564 (2006), pp. 57–103 (cit. on p. 20).

[16] J. Helman and L. Hesselink. "Representation and Display of Vector Field Topology in Fluid Flow Data Sets". In: *Computer* 22.8 (1989), pp. 27–36 (cit. on p. 20).

[17] T. Günther and H. Theisel. "The State of the Art in Vortex Extraction". In: *Computer Graphics Forum* 37.6 (2018), pp. 149–173 (cit. on p. 20).

[18] D. Sujudi and R. Haimes. "Identification of Swirling Flow in 3-D Vector Fields". In: *12th Computational Fluid Dynamics Conference*. AIAA, 1995, p. 1715 (cit. on pp. 21, 22, 116, 131, 141).

[19] R. Peikert and M. Roth. "The "Parallel Vectors" Operator – A Vector Field Visualization Primitive". In: *IEEE Conference on Visualization (VIS)*. IEEE Computer Society, 1999, pp. 263–270 (cit. on pp. 21, 23, 113, 116, 131, 135).

[20] T. Weinkauf, J. Sahner, H. Theisel, and H.-C. Hege. "Cores of Swirling Particle Motion in Unsteady Flows". In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (2007), pp. 1759–1766 (cit. on pp. 21, 23).

[21] T. Günther, M. Gross, and H. Theisel. "Generic Objective Vortices for Flow Visualization". In: *ACM Transactions on Graphics* 36.4 (2017), 141:1–141:11 (cit. on pp. 21, 23).

[22] J. Jeong and F. Hussain. "On the Identification of a Vortex". In: *Journal of Fluid Mechanics* 285 (1995), pp. 69–94 (cit. on p. 22).

[23] G. Haller, A. Hadjighasem, M. Farazmand, and F. Huhn. "Defining Coherent Vortices Objectively from the Vorticity". In: *Journal of Fluid Mechanics* 795 (2016), pp. 136–173 (cit. on p. 22).

[24] S. K. Robinson. "Coherent Motions in the Turbulent Boundary Layer". In: *Annual Review of Fluid Mechanics* 23.1 (1991), pp. 601–639 (cit. on p. 23).

[25] A. Hadjighasem, M. Farazmand, D. Blazevski, G. Froyland, and G. Haller. "A Critical Comparison of Lagrangian Methods for Coherent Structure Detection". In: *Chaos* 27.5 (2017), p. 053104 (cit. on p. 24).

[26] E. Ott. *Chaos in Dynamical Systems*. Cambridge University Press, 2002 (cit. on p. 24).

[27] G. Haller. "Distinguished Material Surfaces and Coherent Structures in Three-Dimensional Fluid Flows". In: *Physica D: Nonlinear Phenomena* 149.4 (2001), pp. 248–277 (cit. on p. 24).

[28] E. Aurell, G. Boffetta, A. Crisanti, G. Paladin, and A. Vulpiani. "Predictability in the Large: An Extension of the Concept of Lyapunov Exponent". In: *Journal of Physics A: Mathematical and General* 30.1 (1997), pp. 1–26 (cit. on p. 24).

[29] S. C. Shadden, F. Lekien, and J. E. Marsden. "Definition and Properties of Lagrangian Coherent Structures from Finite-Time Lyapunov Exponents in Two-Dimensional Aperiodic Flows". In: *Physica D: Nonlinear Phenomena* 212.3 (2005), pp. 271–304 (cit. on p. 24).

[30] A. Kuhn. "Lagrangian Methods for Visualization and Analysis of Time-Dependent Vector Fields". PhD thesis. Otto-von-Guericke Universität Magdeburg, 2013 (cit. on p. 25).

[31] R. Peikert, A. Pobitzer, F. Sadlo, and B. Schindler. "A Comparison of Finite-Time and Finite-Size Lyapunov Exponents". In: *Topological Methods in Data Analysis and Visualization III*. Springer, 2014, pp. 187–200 (cit. on pp. 25, 26).

[32] F. d'Ovidio, V. Fernández, E. Hernández-García, and C. López. "Mixing Structures in the Mediterranean Sea from Finite-Size Lyapunov Exponents". In: *Geophysical Research Letters* 31.17 (2004) (cit. on p. 26).

[33] I. Hernández-Carrasco, C. López, E. Hernández-García, and A. Turiel. "How Reliable are Finite-Size Lyapunov Exponents for the Assessment of Ocean Dynamics?" In: *Ocean Modelling* 36.3 (2011), pp. 208–218 (cit. on p. 26).

[34] S. Pajevic and C. Pierpaoli. "Color Schemes to Represent the Orientation of Anisotropic Tissues from Diffusion Tensor Data: Application to White Matter Fiber Tract Mapping in the Human Brain". In: *Magnetic Resonance in Medicine* 42.3 (1999), pp. 526–540 (cit. on pp. 27, 29).

[35] G. Kindlmann, D. Weinstein, and D. Hart. "Strategies for Direct Volume Rendering of Diffusion Tensor Fields". In: *IEEE Transactions on Visualization and Computer Graphics* 6.2 (2000), pp. 124–138 (cit. on p. 27).

[36] X. Zheng and A. Pang. "HyperLIC". In: *IEEE Conference on Visualization (VIS)*. IEEE Computer Society, 2003, pp. 249–256 (cit. on p. 27).

[37] I. Hotz, L. Feng, H. Hagen, B. Hamann, B. Jeremić, and K. Joy. "Physically Based Methods for Tensor Field Visualization". In: *IEEE Conference on Visualization (VIS)*. IEEE Computer Society, 2004, pp. 123–130 (cit. on pp. 27, 28).

[38] G. Kindlmann and C.-F. Westin. "Diffusion Tensor Visualization With Glyph Packing". In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006) (cit. on p. 29).

[39] L. Feng, I. Hotz, B. Hamann, and K. I. Joy. "Anisotropic Noise Samples". In: *IEEE Transactions on Visualization and Computer Graphics* 14.2 (2008), pp. 342–354 (cit. on p. 29).

[40] P. J. Basser and C. Pierpaoli. "Microstructural and Physiological Features of Tissues Elucidated by Quantitative-Diffusion-Tensor MRI". In: *Journal of Magnetic Resonance, Series B* 111.3 (1996), pp. 209–219 (cit. on p. 29).

[41] M. R. Wiegell, H. B. W. Larsson, and V. J. Wedeen. "Fiber Crossing in Human Brain Depicted with Diffusion Tensor MR Imaging". In: *Radiology* 217.3 (2000), pp. 897–903 (cit. on p. 29).

[42] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit*. 4th ed. Kitware, 2006 (cit. on pp. 29, 124).

[43] G. Kindlmann. "Superquadric Tensor Glyphs". In: *Eurographics / IEEE VGTC Symposium on Visualization (VisSym)*. The Eurographics Association, 2004, pp. 147–154 (cit. on p. 29).

[44] Y. Hashash, J. I. Yao, D. C. Wotring, et al. "Glyph and Hyperstreamline Representation of Stress and Strain Tensors and Material Constitutive Response". In: *International Journal for Numerical and Analytical Methods in Geomechanics* 27.7 (2003), pp. 603–626 (cit. on p. 29).

[45] B. Jeremić, G. Scheuermann, J. Frey, Z. Yang, B. Hamann, K. I. Joy, and H. Hagen. "Tensor Visualizations in Computational Geomechanics". In: *International Journal for Numerical and Analytical Methods in Geomechanics* 26.10 (2002), pp. 925–944 (cit. on pp. 29, 32).

[46] T. Schultz and G. L. Kindlmann. "Superquadric Glyphs for Symmetric Second-Order Tensors". In: *IEEE Transactions on Visualization and Computer Graphics* 16.6 (2010), pp. 1595–1604 (cit. on pp. 29, 30).

[47] T. Gerrits, C. Rössl, and H. Theisel. "Glyphs for General Second-Order 2D and 3D Tensors". In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (2017), pp. 980–989 (cit. on p. 30).

[48] M. M. Focht. *Photoelasticity*. Vol. 1. John Wiley & Sons, 1962 (cit. on p. 31).

[49] S. P. Timoshenko. *History of Strength of Materials*. Dover Civil and Mechanical Engineering. Dover Publications, 1983 (cit. on p. 31).

[50] R. R. Dickinson. "A Unified Approach To The Design Of Visualization Software For The Analysis Of Field Problems". In: *Proc. SPIE 1083, Three-Dimensional Visualization and Display Technologies*. 1989, p. 10838 (cit. on p. 31).

[51] X. Tricoche, X. Zhang, and A. Pang. "Visualizing the Topology of Symmetric, Second-Order, Time-Varying Two-Dimensional Tensor Fields". In: *Visualization and Processing of Tensor Fields*. Ed. by J. Weickert and H. Hagen. Springer, 2006, pp. 225–240 (cit. on p. 31).

[52] D. Kelly and M. Tosh. "Interpreting Load Paths and Stress Trajectories in Elasticity". In: *Engineering Computations* 17.2 (2000), pp. 117–135 (cit. on p. 31).

[53] T. Delmarcelle and L. Hesselink. "Visualizing Second-Order Tensor Fields With Hyperstreamlines". In: *Computer Graphics and Applications* 13.4 (1993), pp. 25–33 (cit. on p. 31).

[54] D. Weinstein, G. Kindlmann, and E. Lundberg. "Tensorlines: Advection-Diffusion Based Propagation Through Diffusion Tensor Fields". In: *IEEE Conference on Visualization (VIS)*. IEEE Computer Society, 1999, pp. 249–253 (cit. on p. 32).

[55] X. Zheng, B. Parlett, and A. Pang. "Topological Structures of 3D Tensor Fields". In: *IEEE Conference on Visualization (VIS)*. IEEE Computer Society, 2005, pp. 551–558 (cit. on pp. 32, 33).

[56] T. Delmarcelle and L. Hesselink. "The Topology of Symmetric, Second-Order Tensor Fields". In: *IEEE Conference on Visualization (VIS)*. IEEE Computer Society, 1994, pp. 140–147 (cit. on p. 33).

[57] L. Hesselink, Y. Levy, and Y. Lavin. "The Topology of Symmetric, Second-Order 3D Tensor Fields". In: *IEEE Transactions on Visualization and Computer Graphics* 3.1 (1997), pp. 1–11 (cit. on p. 33).

[58] X. Zheng and A. Pang. "Topological Lines in 3D Tensor Fields". In: *IEEE Conference on Visualization (VIS)*. IEEE Computer Society, 2004, pp. 313–320 (cit. on pp. 33, 148, 150).

[59] X. Zheng, B. Parlett, and A. Pang. "Topological Lines in 3D Tensor Fields and Discriminant Hessian Factorization". In: *IEEE Transactions on Visualization and Computer Graphics* 11.4 (2005), pp. 395–407 (cit. on p. 33).

[60] X. Tricoche, G. Kindlmann, and C.-F. Westin. "Invariant Crease Lines for Topological and Structural Analysis of Tensor Fields". In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008), pp. 1627–1634 (cit. on p. 33).

[61] J. Palacios, H. Yeh, W. Wang, Y. Zhang, R. S. Laramee, R. Sharma, T. Schultz, and E. Zhang. "Feature Surfaces in Symmetric Tensor Fields Based on Eigenvalue Manifold". In: *IEEE Transactions on Visualization and Computer Graphics* 22.3 (2016), pp. 1248–1260 (cit. on p. 33).

[62] L. Roy, P. Kumar, Y. Zhang, and E. Zhang. "Robust and Fast Extraction of 3D Symmetric Tensor Field Topology". In: *IEEE Transactions on Visualization and Computer Graphics* 25.1 (2019), pp. 1102–1111 (cit. on p. 34).

[63] X. Zheng and A. Pang. "2D Asymmetric Tensor Analysis". In: *IEEE Conference on Visualization (VIS)*. IEEE Computer Society, 2005, pp. 3–10 (cit. on p. 34).

[64] E. Zhang, H. Yeh, Z. Lin, and R. S. Laramee. "Asymmetric Tensor Analysis for Flow Visualization". In: *IEEE Transactions on Visualization and Computer Graphics* 15.1 (2009), pp. 106–122 (cit. on p. 34).

[65] T. Poinsot and D. Veynante. *Theoretical and Numerical Combustion*. 2012 (cit. on pp. 37–39, 41, 42, 49, 67, 86).

[66] J. A. Miller, R. E. Mitchell, M. D. Smooke, and R. J. Kee. "Toward a Comprehensive Chemical Kinetic Mechanism for the Oxidation of Acetylene: Comparison of Model Predictions with Results from Flame and Shock Tube Experiments". In: *Symposium (International) on Combustion* 19.1 (1982), pp. 181–196 (cit. on pp. 46, 47).

[67] J. H. Chen, A. Choudhary, B. de Supinski, M. deVries, E. R. Hawkes, S. Klasky, W.-K. Liao, K.-L. Ma, J. Mellor-Crummey, N. Podhorszki, et al. "Terascale Direct Numerical Simulations of Turbulent Combustion Using S3D". In: *Computational Science & Discovery* 2.1 (2009), p. 015001 (cit. on p. 51).

[68] S. Treichler, M. Bauer, A. Bhagatwala, G. Borghesi, R. Sankaran, H. Kolla, P. S. McCormick, E. Slaughter, W. Lee, A. Aiken, and J. Chen. "S3D-Legion: An Exascale Software for Direct Numerical Simulation of Turbulent Combustion with Complex Multicomponent Chemistry". In: ed. by T. P. Straatsma, K. B. Antypas, and T. J. Williams. 1st ed. Chapman and Hall/CRC, 2017. Chap. 12, pp. 257–278 (cit. on p. 51).

[69] A. Abdelsamie, G. Fru, T. Oster, F. Dietzsch, G. Janiga, and D. Thévenin. "Towards Direct Numerical Simulations of Low-Mach Number Turbulent Reacting and Two-Phase Flows Using Immersed Boundaries". In: *Computers & Fluids* 131 (2016), pp. 123–141 (cit. on pp. 51, 55, 95).

[70] R. Sankaran, E. R. Hawkes, J. H. Chen, T. Lu, and C. K. Law. "Structure of a Spatially Developing Turbulent Lean Methane-Air Bunsen Flame". In: *Proceedings of the Combustion Institute* 31.1 (2007), pp. 1291–1298 (cit. on p. 52).

[71] E. R. Hawkes and J. H. Chen. "Comparison of Direct Numerical Simulation of Lean Premixed Methane-Air Flames with Strained Laminar Flame Calculations". In: *Combustion and Flame* 144.1 (2006), pp. 112–125 (cit. on p. 52).

[72] P. Yeung, S. Girimaji, and S. Pope. "Straining and Scalar Dissipation on Material Surfaces in Turbulence: Implications for Flamelets". In: *Combustion and Flame* 79.3 (1990), pp. 340–365 (cit. on pp. 52, 85, 86).

[73] P. Sripakagorn, S. Mitarai, G. Kosály, and H. Pitsch. "Extinction and Reignition in a Diffusion Flame: A Direct Numerical Simulation Study". In: *Journal of Fluid Mechanics* 518 (2004), pp. 231–259 (cit. on pp. 52, 85, 86, 108).

[74] A. Scholtissek, F. Dietzsch, M. Gauding, and C. Hasse. "In-Situ Tracking of Mixture Fraction Gradient Trajectories and Unsteady Flamelet Analysis in Turbulent Non-Premixed Combustion". In: *Combustion and Flame* 175 (2017), pp. 243–258 (cit. on pp. 52, 86, 108).

[75] H. Childs, E. Brugger, B. Whitlock, J. Meredith, S. Ahern, D. Pugmire, K. Biagas, M. Miller, C. Harrison, G. H. Weber, H. Krishnan, T. Fogal, A. Sanderson, C. Garth, E. W. Bethel, D. Camp, O. Rübel, M. Durant, J. M. Favre, and P. Navrátil. "VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data". In: *High Performance*

Visualization–Enabling Extreme-Scale Scientific Insight. Chapman and Hall/CRC, 2012, pp. 357–372 (cit. on p. 55).

[76] J. Ahrens, B. Geveci, C. Law, C. Hansen, and C. Johnson. "ParaView: An End-User Tool for Large-Data Visualization". In: *The Visualization Handbook*. Elsevier, 2005 (cit. on p. 55).

[77] C. Zistl, R. Hilbert, G. Janiga, and D. Thévenin. "Increasing the Efficiency of Postprocessing for Turbulent Reacting Flows". In: *Computing and Visualization in Science* 12.8 (2009), pp. 383–395 (cit. on pp. 56, 65).

[78] P.-T. Bremer, G. H. Weber, J. Tierny, V. Pascucci, M. S. Day, and J. B. Bell. "A Topological Framework for the Interactive Exploration of Large Scale Turbulent Combustion". In: *IEEE International Conference on e-Science*. IEEE Computer Society, 2009, pp. 247–254 (cit. on pp. 56, 61, 66).

[79] P. Bremer, G. Weber, J. Tierny, V. Pascucci, M. Day, and J. Bell. "Interactive Exploration and Analysis of Large-Scale Simulations Using Topology-Based Data Segmentation". In: *IEEE Transactions on Visualization and Computer Graphics* 17.9 (2011), pp. 1307–1324 (cit. on pp. 56, 61, 66).

[80] P.-T. Bremer, G. H. Weber, V. Pascucci, M. Day, and J. B. Bell. "Analyzing and Tracking Burning Structures in Lean Premixed Hydrogen Flames". In: *IEEE Transactions on Visualization and Computer Graphics* 16.2 (2010), pp. 248–260 (cit. on pp. 56, 66, 87).

[81] Y. Wang, H. Yu, and K.-L. Ma. "Scalable Parallel Feature Extraction and Tracking for Large Time-Varying 3D Volume Data." In: *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. The Eurographics Association, 2013, pp. 17–24 (cit. on pp. 56, 87).

[82] A. Schnorr, D. N. Helmrich, D. Denker, T. Kuhlen, and B. Hentschel. "Feature Tracking by Two-Step Optimization". In: *IEEE Transactions on Visualization and Computer Graphics* (2018). Advance online publication. doi: 10.1109/TVCG.2018.2883630 (cit. on p. 56).

[83] F. Sauer, H. Yu, and K.-L. Ma. "Trajectory-Based Flow Feature Tracking in Joint Particle/Volume Datasets". In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (2014), pp. 2565–2574 (cit. on pp. 56, 86).

[84] J. Wei, H. Yu, R. W. Grout, J. H. Chen, and K. Ma. "Dual Space Analysis of Turbulent Combustion Particle Data". In: *IEEE Pacific Visualization Symposium (PacificVis)*. IEEE Computer Society, 2011, pp. 91–98 (cit. on pp. 56, 57).

[85] K.-L. Ma. “In Situ Visualization at Extreme Scale: Challenges and Opportunities”. In: *Computer Graphics and Applications* 29.6 (2009), pp. 14–19 (cit. on pp. 56, 108).

[86] U. Ayachit, A. Bauer, B. Geveci, P. O’Leary, K. Moreland, N. Fabian, and J. Mauldin. “ParaView Catalyst: Enabling In Situ Data Analysis and Visualization”. In: *Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*. ACM, 2015, pp. 25–29 (cit. on p. 58).

[87] B. Whitlock, J. M. Favre, and J. S. Meredith. “Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System”. In: *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. The Eurographics Association, 2011, pp. 101–109 (cit. on p. 58).

[88] M. Larsen, E. Brugger, H. Childs, J. Eliot, K. Griffin, and C. Harrison. “Strawman: A Batch In Situ Visualization and Analysis Infrastructure for Multi-Physics Simulation Codes”. In: *Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*. ACM, 2015, pp. 30–35 (cit. on p. 58).

[89] J. S. Meredith, S. Ahern, D. Pugmire, and R. Sisneros. “EAVL: The Extreme-scale Analysis and Visualization Library”. In: *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. The Eurographics Association, 2012, pp. 21–30 (cit. on p. 58).

[90] K. Moreland, U. Ayachit, B. Geveci, and K.-L. Ma. “DAX Toolkit: A Proposed Framework for Data Analysis and Visualization at Extreme Scale”. In: *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*. IEEE Computer Society, 2011, pp. 97–104 (cit. on p. 58).

[91] L.-T. Lo, C. Sewell, and J. Ahrens. “PISTON: A Portable Cross-Platform Framework for Data-Parallel Visualization Operators”. In: *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. The Eurographics Association, 2012, pp. 11–20 (cit. on p. 58).

[92] K. Moreland, C. Sewell, W. Usher, L. Lo, J. Meredith, D. Pugmire, J. Kress, H. Schroots, K.-L. Ma, H. Childs, M. Larsen, C. Chen, R. Maynard, and B. Geveci. “VTK-m: Accelerating the Visualization Toolkit for Massively Threaded Architectures”. In: *Computer Graphics and Applications* 36.3 (2016), pp. 48–58 (cit. on p. 58).

[93] V. Vishwanath, M. Hereld, and M. E. Papka. “Toward Simulation-Time Data Analysis and I/O Acceleration on Leadership-Class Systems”. In: *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*. IEEE Computer Society, 2011, pp. 9–14 (cit. on p. 58).

[94] J. Biddiscombe, J. Soumagne, G. Oger, D. Guibert, and J.-G. Piccinali. "Parallel Computational Steering and Analysis for HPC Applications using a ParaView Interface and the HDF5 DSM Virtual File Driver". In: *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. The Eurographics Association, 2011, pp. 91–100 (cit. on p. 58).

[95] M. Dorier, R. Sisneros, T. Peterka, G. Antoniu, and D. Semeraro. "Damaris/Viz: A Nonintrusive, Adaptable and User-Friendly In Situ Visualization Framework". In: *IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)*. IEEE Computer Society, 2013, pp. 67–75 (cit. on p. 58).

[96] T. Fogal, F. Proch, A. Schiewe, O. Hasemann, A. Kempf, and J. Krüger. "Freeprocessing: Transparent in Situ Visualization via Data Interception". In: *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. The Eurographics Association, 2014, pp. 49–56 (cit. on p. 58).

[97] F. Zheng, H. Yu, C. Hantas, M. Wolf, G. Eisenhauer, K. Schwan, H. Abbasi, and S. Klasky. "GoldRush: Resource Efficient in Situ Scientific Data Analytics Using Fine-grained Interference Aware Execution". In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, 78:1–78:12 (cit. on p. 59).

[98] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, Q. Liu, S. Klasky, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf. "PreDatA – Preparatory Data Analytics on Peta-scale Machines". In: *IEEE International Symposium on Parallel Distributed Processing (IPDPS)*. IEEE Computer Society, 2010, pp. 1–12 (cit. on p. 59).

[99] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng. "DataStager: Scalable Data Staging Services for Petascale Applications". In: *Cluster Computing* 13.3 (2010), pp. 277–290 (cit. on p. 59).

[100] H. Abbasi, G. Eisenhauer, M. Wolf, K. Schwan, and S. Klasky. "Just in Time: Adding Value to the IO Pipelines of High Performance Applications with JITStaging". In: *Proceedings of the 20th International Symposium on High Performance Distributed Computing*. ACM, 2011, pp. 27–36 (cit. on p. 59).

[101] C. Docan, M. Parashar, J. Cummings, and S. Klasky. "Moving the Code to the Data - Dynamic Code Deployment Using ActiveSpaces". In: *IEEE International Parallel Distributed Processing Symposium*. IEEE Computer Society, 2011, pp. 758–769 (cit. on p. 59).

[102] C. Docan, M. Parashar, and S. Klasky. "DataSpaces: An Interaction and Coordination Framework for Coupled Simulation Workflows". In: *Cluster Computing* 15.2 (2012), pp. 163–181 (cit. on p. 59).

[103] J. C. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, P. Pebay, D. Thompson, H. Yu, F. Zhang, and J. Chen. "Combining In-situ and In-transit Processing to Enable Extreme-scale Scientific Analysis". In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, 2012, 49:1–49:9 (cit. on p. 59).

[104] H. Yu, C. Wang, and K.-L. Ma. "Massively Parallel Volume Rendering Using 2-3 Swap Image Compositing". In: *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. IEEE Computer Society, 2008, 48:1–48:11 (cit. on pp. 59, 60).

[105] W. Kendall, T. Peterka, J. Huang, H.-W. Shen, and R. Ross. "Accelerating and Benchmarking Radix-k Image Compositing at Large Scale". In: *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. The Eurographics Association, 2010, pp. 101–110 (cit. on p. 59).

[106] K. Moreland, W. Kendall, T. Peterka, and J. Huang. "An Image Compositing Solution at Scale". In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 2011, 25:1–25:10 (cit. on p. 59).

[107] X. Cavin and O. Demengeon. "Shift-Based Parallel Image Compositing on InfiniBand Fat-Trees". In: *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. The Eurographics Association, 2012, pp. 129–138 (cit. on p. 59).

[108] J. Nonaka, K. Ono, and M. Fujita. "Multi-step Image Compositing for Massively Parallel Rendering". In: *2014 International Conference on High Performance Computing Simulation (HPCS)*. 2014, pp. 627–634 (cit. on p. 59).

[109] A. V. P. Grosset, A. Knoll, and C. Hansen. "Dynamically Scheduled Region-based Image Compositing". In: *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. The Eurographics Association, 2016, pp. 79–88 (cit. on p. 59).

[110] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. F. Samatova. "Compressing the Incompressible with ISABELA: In-situ Reduction of Spatio-temporal Data". In: *European Conference on Parallel Processing (Euro-Par)*. Springer, 2011, pp. 366–379 (cit. on pp. 59, 66).

[111] A. Kageyama and T. Yamada. "An Approach to Exascale Visualization: Interactive Viewing of In-Situ Visualization". In: *Computer Physics Communications* 185.1 (2014), pp. 79–85 (cit. on pp. 59, 60).

[112] J. Ahrens, S. Jourdain, P. O'Leary, J. Patchett, D. H. Rogers, and M. Petersen. "An Image-based Approach to Extreme Scale in Situ Visualization and Analysis". In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, 2014, pp. 424–434 (cit. on pp. 59, 60).

[113] H. Yu, C. Wang, R. W. Grout, J. H. Chen, and K.-L. Ma. "In Situ Visualization for Large-Scale Combustion Simulations". In: *Computer Graphics and Applications* 30.3 (2010), pp. 45–57 (cit. on p. 60).

[114] A. Tikhonova, H. Yu, C. D. Correa, J. H. Chen, and K.-L. Ma. "A Preview and Exploratory Technique for Large-Scale Scientific Simulations". In: *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. The Eurographics Association, 2011, pp. 111–120 (cit. on p. 60).

[115] J. Chen, D. Silver, and M. Parashar. "Real-time Feature Extraction and Tracking in a Computational Steering Environment". In: *Proceedings of the Advanced Simulations Technologies Confenrence*. 2003 (cit. on p. 60).

[116] F. Zhang, S. Lasluisa, T. Jin, I. Rodero, H. Bui, and M. Parashar. "In-situ Feature-Based Objects Tracking for Large-Scale Scientific Simulations". In: *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. 2012, pp. 736–740 (cit. on p. 60).

[117] A. Quiroz, N. Gnanasambandam, M. Parashar, and N. Sharma. "Robust Clustering Analysis for the Management of Self-monitoring Distributed Systems". In: *Cluster Computing* 12.1 (2008), p. 73 (cit. on p. 60).

[118] E. P. N. Duque, D. Hiepler, C. P. Stone, S. M. Legensky, K.-L. Ma, C. Muelder, and J. Wei. "IFDT – Intelligent In-Situ Feature Detection, Extraction, Tracking and Visualization for Turbulent Flow Simulations". In: *7th International Conference on Computational Fluid Dynamics*. 2012 (cit. on pp. 61, 86).

[119] Y. C. Ye, Y. Wang, R. Miller, K.-L. Ma, and K. Ono. "In Situ Depth Maps Based Feature Extraction and Tracking". In: *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*. 2015, pp. 1–8 (cit. on p. 61).

[120] A. G. Landge, V. Pascucci, A. Gyulassy, J. C. Bennett, H. Kolla, J. Chen, and P.-T. Bremer. "In-Situ Feature Extraction of Large Scale Combustion Simulations Using Segmented Merge Trees". In: *Proceedings of the International Conference on High Performance Computing, Networking,*

Storage and Analysis. IEEE Computer Society, 2014, pp. 1020–1031 (cit. on pp. 61, 66).

[121] Y. C. Ye, T. Neuroth, F. Sauer, K.-L. Ma, G. Borghesi, A. Konduri, H. Kolla, and J. Chen. "In Situ Generated Probability Distribution Functions for Interactive Post Hoc Visualization and Analysis". In: *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*. IEEE Computer Society, 2016, pp. 65–74 (cit. on pp. 61, 66).

[122] T. Oster, D. J. Lehmann, G. Fru, H. Theisel, and D. Thévenin. "Sparse Representation and Visualization for Direct Numerical Simulation of Premixed Combustion". In: *Computer Graphics Forum* 33.3 (2014), pp. 321–330 (cit. on p. 65).

[123] S. Lakshminarasimhan, J. Jenkins, I. Arkatkar, Z. Gong, H. Kolla, S.-H. Ku, S. Ethier, J. Chen, C. S. Chang, S. Klasky, R. Latham, R. Ross, and N. F. Samatova. "ISABELA-QA: Query-driven Analytics with ISABELA-compressed Extreme-scale Scientific Data". In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 2011, 31:1–31:11 (cit. on p. 66).

[124] J.-D. Boissonnat and S. Oudot. "Provably Good Sampling and Meshing of Surfaces". In: *Graphical Models* 67.5 (2005), pp. 405–451 (cit. on p. 67).

[125] G. Kindlmann, R. Whitaker, T. Tasdizen, and T. Möller. "Curvature-Based Transfer Functions for Direct Volume Rendering: Methods and Applications". In: *IEEE Conference on Visualization (VIS)*. IEEE Computer Society, 2003, pp. 513–520 (cit. on p. 67).

[126] B. Jähne. *Digital Image Processing*. Springer, 2005 (cit. on p. 70).

[127] B. T. Phong. "Illumination for Computer Generated Pictures". In: *Communications of the ACM* 18.6 (1975), pp. 311–317 (cit. on p. 76).

[128] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. "An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions". In: *Journal of the ACM* 45.6 (1998), pp. 891–923 (cit. on p. 78).

[129] D. Shepard. "A Two-dimensional Interpolation Function for Irregularly-spaced Data". In: *Proceedings of the 1968 23rd ACM National Conference*. ACM, 1968, pp. 517–524 (cit. on p. 78).

[130] B.-J. Kim, Z. Xiong, and W. A. Pearlman. "Low Bit-Rate Scalable Video Coding With 3-D Set Partitioning in Hierarchical Trees (3-D SPIHT)". In: *IEEE Transactions on Circuits and Systems for Video Technology* 10.8 (2000), pp. 1374–1387 (cit. on p. 79).

[131] J. E. Fowler. "QccPack: An Open-Source Software Library for Quantization, Compression, and Coding". In: *International Symposium on Optical Science and Technology*. International Society for Optics and Photonics. 2000, pp. 294–301 (cit. on p. 79).

[132] A. Ferrante and S. Elghobashi. "On the Physical Mechanisms of Two-Way Coupling in Particle-Laden Isotropic Turbulence". In: *Physics of Fluids* 15.2 (2003), pp. 315–329 (cit. on p. 82).

[133] T. Oster, A. Abdelsamie, M. Motejat, T. Gerrits, C. Rössl, D. Thévenin, and H. Theisel. "On-The-Fly Tracking of Flame Surfaces for the Visual Analysis of Combustion Processes". In: *Computer Graphics Forum* 37.6 (2018), pp. 358–369 (cit. on pp. 85, 99).

[134] C. Garth, A. Wiebel, X. Tricoche, K. Joy, and G. Scheuermann. "Lagrangian Visualization of Flow-Embedded Surface Structures". In: *Computer Graphics Forum* 27.3 (2008), pp. 1007–1014 (cit. on p. 86).

[135] D. Silver and X.Wang. "Tracking and Visualizing Turbulent 3D Features". In: *IEEE Transactions on Visualization and Computer Graphics* 3.2 (1997), pp. 129–141 (cit. on p. 86).

[136] J. Clyne, P. Mininni, and A. Norton. "Physically-Based Feature Tracking for CFD Data". In: *IEEE Transactions on Visualization and Computer Graphics* 19.6 (2013), pp. 1020–1033 (cit. on p. 86).

[137] A. Mascarenhas, R. W. Grout, C. S. Yoo, and J. H. Chen. "Tracking Flame Base Movement and Interaction With Ignition Kernels Using Topological Methods". In: *Journal of Physics: Conference Series* 180.1 (2009), p. 012086 (cit. on p. 86).

[138] C. Muelder and K.-L. Ma. "Interactive Feature Extraction and Tracking by Utilizing Region Coherency". In: *IEEE Pacific Visualization Symposium (PacificVis)*. IEEE Computer Society, 2009, pp. 17–24 (cit. on p. 86).

[139] C. Garth, X. Tricoche, and G. Scheuermann. "Tracking of Vector Field Singularities in Unstructured 3D Time-Dependent Datasets". In: *IEEE Conference on Visualization (VIS)*. IEEE Computer Society, 2004, pp. 329–336 (cit. on p. 87).

[140] H. Theisel and H.-P. Seidel. "Feature Flow Fields". In: *Eurographics / IEEE VGTC Symposium on Visualization (VisSym)*. The Eurographics Association, 2003, pp. 141–148 (cit. on pp. 87, 116).

[141] P. Crossno and E. Angel. "Isosurface Extraction Using Particle Systems". In: *IEEE Conference on Visualization (VIS)*. IEEE Computer Society, 1997, pp. 495–498 (cit. on p. 87).

[142] H. Krishnan, C. Garth, and K. I. Joy. "Time and Streak Surfaces for Flow Visualization in Large Time-Varying Data Sets". In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (2009), pp. 1267–1274 (cit. on p. 87).

[143] K. Bürger, F. Ferstl, H. Theisel, and R. Westermann. "Interactive Streak Surface Visualization on the GPU". In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (2009), pp. 1259–1266 (cit. on p. 87).

[144] A. Berres, H. Obermaier, K. Joy, and H. Hagen. "Adaptive Particle Relaxation for Time Surfaces". In: *IEEE Pacific Visualization Symposium (PacificVis)*. IEEE Computer Society, 2015, pp. 147–151 (cit. on p. 87).

[145] D. Camp, H. Childs, C. Garth, D. Pugmire, and K. I. Joy. "Parallel Stream Surface Computation for Large Data Sets". In: *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*. IEEE Computer Society, 2012, pp. 39–47 (cit. on p. 87).

[146] MPI Forum. *MPI: A Message-Passing Interface Standard, Version 3.1*. High Performance Computing Center Stuttgart (HLRS), 2015 (cit. on p. 96).

[147] G. Haller. "Lagrangian Coherent Structures From Approximate Velocity Data". In: *Physics of Fluids* 14.6 (2002), pp. 1851–1861 (cit. on p. 99).

[148] T. Oster, C. Rössl, and H. Theisel. "The Parallel Eigenvectors Operator". In: *International Symposium on Vision, Modeling and Visualization (VMV)*. The Eurographics Association, 2018 (cit. on p. 113).

[149] R. Haralick. "Ridges and Valleys on Digital Images". In: *Computer Vision, Graphics, and Image Processing* 22 (1983), pp. 28–38 (cit. on p. 116).

[150] D. Kenwright, C. Henze, and C. Levit. "Feature Extraction of Separation and Attachment Lines". In: *IEEE Transactions on Visualization and Computer Graphics* 5.2 (1999), pp. 135–144 (cit. on p. 116).

[151] D. C. Banks and B. A. Singer. "A Predictor-Corrector Technique for Visualizing Unsteady Flow". In: *IEEE Transactions on Visualization and Computer Graphics* 1.2 (1995), pp. 151–163 (cit. on p. 116).

[152] M. Roth. "Automatic Extraction of Vortex Core Lines and Other Line Type Features for Scientific Visualization". PhD thesis. ETH Zürich, 2000 (cit. on p. 116).

[153] H. Miura and S. Kida. "Identification of Tubular Vortices in Turbulence". In: *Journal of the Physical Society of Japan* 66.5 (1997), pp. 1331–1334 (cit. on p. 116).

[154] J. Sukharev, X. Zheng, and A. Pang. "Tracing Parallel Vectors". In: *Proc. SPIE 6060, Visualization and Data Analysis*. International Society for Optics and Photonics. 2006, p. 606011 (cit. on p. 116).

[155] A. van Gelder and A. Pang. "Using PVsolve to Analyze and Locate Positions of Parallel Vectors". In: *IEEE Transactions on Visualization and Computer Graphics* 15.4 (2009), pp. 682–695 (cit. on p. 116).

[156] T. Weinkauf, H. Theisel, A. V. Gelder, and A. Pang. "Stable Feature Flow Fields". In: *IEEE Transactions on Visualization and Computer Graphics* 17.6 (2011), pp. 770–780 (cit. on p. 116).

[157] M. Roth and R. Peikert. "A Higher-Order Method for Finding Vortex Core Lines". In: *IEEE Conference on Visualization (VIS)*. IEEE Computer Society, 1998, pp. 143–150 (cit. on p. 116).

[158] D. Bauer and R. Peikert. "Vortex Tracking in Scale-Space". In: *Eurographics / IEEE VGTC Symposium on Visualization (VisSym)*. The Eurographics Association, 2002, pp. 233–240 (cit. on p. 116).

[159] H. Theisel, T. Weinkauf, H.-C. Hege, and H.-P. Seidel. "Topological Methods for 2D Time-Dependent Vector Fields Based on Stream Lines and Path Lines". In: *IEEE Transactions on Visualization and Computer Graphics* 4 (2005), pp. 383–394 (cit. on p. 116).

[160] R. Fuchs, R. Peikert, H. Hauser, F. Sadlo, and P. Muigg. "Parallel Vectors Criteria for Unsteady Flow Vortices". In: *IEEE Transactions on Visualization and Computer Graphics* 14 (2007), pp. 615–626 (cit. on p. 116).

[161] C. Pagot, D. Osmari, F. Sadlo, D. Weiskopf, T. Ertl, and J. Comba. "Efficient Parallel Vectors Feature Extraction From Higher-Order Data". In: *Computer Graphics Forum* 30.3 (2011), pp. 751–760 (cit. on p. 116).

[162] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. 4th ed. Boston: Academic Press, 1997 (cit. on p. 120).

[163] B. S. Everitt, S. Landau, M. Leese, and D. Stahl. "Hierarchical Clustering". In: *Cluster Analysis*. Wiley-Blackwell, 2011. Chap. 4, pp. 71–110 (cit. on p. 122).

[164] A. S. Saada. *Elasticity: Theory and Applications*. Vol. 16. Elsevier, 2013 (cit. on p. 124).

[165] *OpenFOAM: The Open Source CFD Toolbox*. http://www.openfoam.org (cit. on p. 126).

[166] T. Oster, C. Rössl, and H. Theisel. "Core Lines in 3D Second-Order Tensor Fields". In: *Computer Graphics Forum* 37.3 (2018), pp. 327–337 (cit. on p. 131).

[167] G. M. Machado, F. Sadlo, and T. Ertl. "Local Extraction of Bifurcation Lines". In: *International Symposium on Vision, Modeling and Visualization (VMV)*. The Eurographics Association, 2013, pp. 17–24 (cit. on pp. 133, 151).

[168] G. M. Machado, S. Boblest, T. Ertl, and F. Sadlo. "Space-Time Bifurcation Lines for Extraction of 2D Lagrangian Coherent Structures". In: *Computer Graphics Forum* 35.3 (2016), pp. 91–100 (cit. on pp. 133, 151).

[169] J. Hoschek and D. Lasser. *Fundamentals of Computer Aided Geometric Design*. A K Peters, 1993 (cit. on pp. 136, 138).

[170] A. Rockwood, K. Heaton, and T. Davis. "Real-Time Rendering of Trimmed Surfaces". In: *ACM SIGGRAPH Computer Graphics* 23.3 (1989), pp. 107–116 (cit. on p. 138).

[171] OpenMP Architecture Review Board. *OpenMP Application Program Interface Version 4.0*. 2013 (cit. on p. 144).

[172] G. Kindlmann, X. Tricoche, and C.-F. Westin. "Delineating White Matter Structure in Diffusion Tensor MRI With Anisotropy Creases". In: *Medical Image Analysis* 11.5 (2007), pp. 492–502 (cit. on p. 151).

[173] X. Zheng and P. Palffy-Muhoray. "Eigenvalue Decomposition for Tensors of Arbitrary Rank". In: *electronic-Liquid Crystal Communications* (2007) (cit. on p. 151).