# Utilizing Topology Based Domain Segmentation for In-Situ Identification and Classification of Vortices in Simulations of Turbulent Flows

*by*

Marius Koch

Department of Aeronautics

Imperial College London

This thesis is submitted for the degree of

Doctor of Philosophy

## Abstract

Fully turbulent flow fields are populated with features such as vortices or eddies. Analysing flow features and their interaction can lead to insight into the physics of turbulence. An enhanced understanding of turbulence benefits, for example, the development of turbulence models, which can help dramatically reducing the computational costs of turbulent flow simulations. The vast abundance of features in fully turbulent flow fields demands an automated identification and analysis process. To address this issue Tracer was developed, an in-situ software framework to extract flow features from data produced by high-fidelity unsteady computational fluid dynamics (CFD) simulations conducted on GPU systems. Unsteady CFD simulations can produce a significant amount of data. The majority of this data cannot be accessed following the classical paradigm of storing the flow field to disk due to the bottleneck of writing the data disk and limitations of available storage capacity. To avoid the restrictions related to moving and storing data, Tracer is able to run concurrently with the simulation, analysing the data while it is still on the GPU's system memory. The intensities of flow features generally span multiple orders of magnitude making their identification a challenging task. Contrary to the classic approach of extracting features by defining a global threshold of a scalar $f$, Tracer identifies an individual threshold for each feature based on the topology of $f$, facilitating the identification of features over a wide range of scales and intensities. The individual thresholds are identified using a join tree, which tracks the topology of superlevel sets of $f$. A performance analysis showed that Tracer requires less system memory than the standalone CFD solver. Adding Tracer in-situ to a CFD simulation hence comes with a reasonable increase in system memory usage. The increase in runtime depends on the number of time steps that the CFD solver takes between two applications of Tracer to the flow field, but is typically within the single-digit percentage range. The advantages of topology based feature identification using Tracer versus the classic approach of using a global threshold are demonstrated qualitatively by visualising vortices in flow around an SD7003 aerofoil. The application of Tracer is demonstrated on the simulation of the turbulent transition of a Taylor-Green vortex with the aim of counting the number of vortices over time. If vortices are extracted based on the topolgy of the $Q$-criterion field a steep increase in the number of vortices can be observed during turbulent breakdown and a decrease during the decay of turbulence. Additionally a feature based analysis of turbulent channel flows up to $Re_\tau = 550$ was conducted examining the topology and the geometry of vortices. While the topological organisation of individual vortices in the near-wall and in the central region are found to be indistinguishable, there is a difference when vortex clusters are extracted. It was also found that vortices less than 70 wall units away from the wall tend to align in the streamwise direction; vortices further away from the wall were found to be geometrically isotropic. These results furthermore support the assumption that the diameter of elongated vortices scales with the Kolmogorov length.

## Acknowledgements

This thesis is dedicated to my mum, who passed away way too young just months before this project was finished. I could write so many pages about her achievements as a strong and independent woman, which made her a true role model. How she raised me on her own in her 20s and 30s while working full time. How altruistic she was, always looking after the well-being of others before looking after herself. I am eternally grateful for her dedication and patience with me and for teaching me the right values. But what actually makes her such a special person is that she had this rare gift of making everyone happy. Virtually every photo I have of her shows her in high spirits. It is impossible to escape her similes and laughter with which she infected all around her. Everyone I talked to since has told me how they remember her as such a cheerful person, brightening everyone's day. I always felt so happy when I was with her. *I miss you mum.*

I would like to express my sincere gratitude to my supervisors Prof. Peter Vincent and Prof. Paul Kelly. They couldn't have been more supportive of my mum and me spending her last months together. I cannot stress enough how important this was for the both of us and how grateful I am to have had the chance to be with her. Throughout my PhD they gave me as much freedom as one can ask while providing valuable advice whenever I needed it. Their passion and guidance motivated me from the beginning until the end of this project.

I would also like to thank Prof. Chris Keylock whose ideas sparked many inspiring discussions. My sincere thanks goes to Prof. Hamish Carr, who took the time to teach me about topological methods. I want to thank the PyFR group, working with you guys was always fun and I appreciate the numerous lunches and pub visits we had.

A big thank you goes out to my friends, who make life what it mostly is: fun! You also were there when I needed you most, looking after me and accompanying me on the way out of the most difficult times. I'd also like to express my gratitude to my mum's friends, who supported her fighting against the disease and kept supporting me when all of a sudden I was on my own.

Lastly I thank Jasmin for bearing with me throughout all this time. As if a long-distant relationship isn't hard enough already, on top it got torpedoed by a pandemic and all this other shit that has happened in the last year. *I'm so happy to have you by my side.*

## Statement of Originality

The work presented in this thesis comprises only the author's original work unless where due acknowledgement has been made. None of the original material has been submitted for the award of any other degree or diploma of the university or any other institute of higher learning.

## Copyright Declaration

# Contents

# Nomenclature

## Latin Symbols

| | |
|---|---|
| $AsD$ | asymptotic decider |
| $activeEdges$ | active graph storing all edges which are processed at a given step |
| $ap$ | ascending path and associated peak |
| $\mathbf{C}$ | coordinate vector of centre of a feature |
| $\langle C_r \rangle$ | average number of supAs in a hypA of order $r$ |
| $DisC$ | discriminant criterion describing the topology of an isosurface |
| $\widetilde{F}$ | interpolation function |
| $ep$ | associated peak of edges |
| $f$ | scalar field the topology of which is analysed |
| $f_{min}$ | scalar value above which the topology of the scalar field is analysed |
| $\mathbb{G}$ | edge graph |
| $\mathcal{H}$ | hexahedral cell |
| $\mathbf{K}$ | function space |
| $l$ | length |
| $\langle M_r \rangle$ | average magnitude of a hypA of order $r$ |
| $Ma$ | Mach number |
| $m$ | occupied system memory |
| $mfs$ | minimal feature size |
| $N$ | neighbourhood around a critical point |
| $N_r$ | number of hypAs of order $r$ |
| $n_R$ | number of features extracted with $R_{cut}$ |
| $n_{rr}$ | number of connected relevant regions with $f > f_{min}$ |
| $n_{edges}$ | number of edges in domain |
| $n_{feat}$ | number of features |
| $n_{hyp}$ | number of hyper nodes |
| $n_{impVerts}$ | number of vertices in the relevant regions |
| $n_{max}$ | number of maxima in relevant domain |
| $n_Q$ | number of features extracted with $Q_{cut}$ |
| $n_{sup}$ | number of super nodes |
| $n_{verts}$ | number of vertices in domain |

| | |
|---|---|
| $ovh$ | offset vector into hyper structure |
| $ovs$ | offset vector into super structure |
| $ova$ | offset vector into augmented structure |
| $ovf$ | offset vector into list of features |
| $P_C$ | HS ratio comparing $\bar{C}_r$ of consecutive orders |
| $P_M$ | HS ratio comparing $\bar{M}_r$ of consecutive orders |
| $P_N$ | HS ratio comparing $N_r$ of consecutive orders |
| $P_T$ | ratio comparing Tokunaga indices of consecutive orders |
| $Q_{cut}$ | $Q$ threshold for feature extraction |
| $R$ | relevance criterion |
| $R_{cut}$ | relevance threshold for feature extraction |
| $Re$ | Reynolds number |
| $r$ | Horton-Strahler order |
| $r_{max}$ | highest $r$ occurring in a given region |
| $T$ | Tokunaga index |
| $t$ | time |
| $\mathcal{T}$ | face of a cell |
| $V$ | volume |
| $V_i$ | vertex with index $i$ |

## Greek Symbols

| | |
|---|---|
| $\delta$ | channel half height |
| $\eta$ | cell type |
| $\lambda$ | degree of node |
| $\sigma$ | scalar field, which is defined in the same domain as $s$ |
| $\tau$ | runtime |
| $\tau_w$ | wall shear stress |
| $\Omega$ | domain |
| $\omega$ | cell |

## Abbreviations

| | |
|---|---|
| AEH | attached eddy hypothesis |
| AEM | attached eddy model |
| augS | augmented structure |
| CFD | computational fluid dynamics |
| CPU | central processing unit |
| CT | contour tree |
| CTG | critical topology graph |
| GPU | graphics processing unit |
| HS | Horton-Strahler |
| hypA | hyper arc |
| hypG | governing saddle of a hypN, lower end of a hypA |
| hypN | hyper node, upper end of a hypA |
| hypS | hyper structure |
| ID | index |
| JT | join tree |
| pdf | probability density function |
| PPP | parallel peak pruning |
| supA | super arc |
| supG | lower end of a supA |
| supN | super node |
| supS | super structure |
| TGV | Taylor-Green vortex |

# 1 Introduction

Fully turbulent flow fields are populated with features such as vortices or eddies. Analysing flow features and their interactions facilitates drawing conclusions on the physics of turbulence and with it helping modelling small scale turbulent contributions or detecting extreme events. The vast abundance of features in fully turbulent flow fields demands feature based analyses of such flows to be automated. To address this issue Tracer was developed, an in-situ software framework to extract flow features in data of unsteady computational fluid dynamics (CFD) simulations conducted on GPU systems.

## 1.1 Importance of Flow Features in Unsteady Flows

Vortices are important features of unsteady flows. While the problem of defining a general and precise vortex definition is not considered solved [74, 110], there is agreement that the impact of vortices ranges from causing local events to characterizing the overall behaviour of the flow field. So has a vortex ring been identified to aid the filling of the heart [56]. A correlation between the shape of the vortex ring and the shape of the inflow area was established [34]. The alteration of the inflow was suggested to cause pathologies [58]. Such kind of knowledge is of importance in designing prosthetic valves [88]. Also in engineering applications on a larger scale vortices are taken into account, extreme aerodynamic loads on roofs of houses for example were correlated to the appearance of a strong vortex in that area [98]. Furthermore vortices are generated with the purpose of controlling flows like suppressing the separation of a boundary layer [66]. In wall bounded flows quasi-streamwise vortices which are in close proximity to the wall are correlated to an increase in skin friction [12, 83]. Drag is particularly strong in regions nearby quasi-streamwise vortices [61, 82]. Turbulent flow features in the near-wall region regenerate through a self-sustaining cycle [41, 54, 90]. Some progress has been made in diminishing drag by interrupting this cycle both passively [8, 84] and actively [2, 26, 36]. Furthermore near-wall vortices are modulated by large-scale features in the outer region [83, 89]. A range of studies have been conducted with the aim of reducing skin friction by controlling such large-scale outer features [1, 7, 108].

Wall bounded turbulent flows are characterised by the presence of eddies, which are described as inertial or energy containing motions [102]. Studying the abundance and characteristics of eddies has significantly advanced modeling of wall bounded turbulence. Models help understanding and predicting properties of flows and have the potential to minimise computational costs of flow simulations. Via the attached eddy model (AEM) for example conclusions on the physics of the logarithmic layer of wall bounded flows can be drawn based on the assumption of simple self-similar attached eddies [72]. The AEM builds on Townsend's attached eddy hypothesis [101, 102], which assumes that wall-bounded turbulent flows are governed by geometrically

self-similar features which scale with the wall-distance of their centre. The hierarchical form of such features was later described in more detail by Perry and Chong [87]. The predictions of the AEM were confirmed in experiments [28, 75] and in direct numerical simulations [65, 92]. In a filtered large eddy simulation Hwang [48] found self-similar statistical motions that are in agreementwith the AEM. Hwang and Bengana [49] showed that these motions are also self-sustaining. Furthermore the AEM has found applications in describing and analysing athmospheric surface layer flows [45, 55] as well as building structure-based models for large eddy simulations [3, 35]. Next to extending the AEM to regions beyond the logarithmic layer and including a wider range of scales open questions are to which extend eddies are attached and whether self-similar attached eddies are dominant flow feature at very high Reynolds numbers [72].

The above examples of unsteady flows characterised by vortices and eddies are far from providing a complete overview of flows in which features play an important role. The importance of various flow features in a wide range on unsteady flows motivates examining such flows via feature based analyses.

## 1.2 Feature-Based Analysis of Unsteady Flow Fields

Evaluating individual features in turbulent flows can provide valuable insight in their physics. Silver [93] has suggested the following benefits of feature based analysis:

- Reduction of visual clutter.
- Measuring features individually.
- Classification of features.
- Juxtaposition of features.
- Feature cardinality.
- Data reduction.
- Tracking.

Feature based visualisation *reduces visual clutter* as artefacts and noise-induced structures can be removed. Flow features like vortices or eddies can be defined as connected regions encapsulated by contours. Identifying the volume occupied by a feature facilitates *measuring features individually.* Geometric measures like location, volume and shape of such connected regions can be determined and certain quantities can be integrated over the region occupied by the feature. Describing features by these parameters makes their *classification* and *juxtaposition* possible. The identification of features is furthermore a prerequisite for *tracking* them in time-dependent data sets, e.g. by comparing the overlap of occupied regions in consecutive snapshots. This way insight in lifetime, evolution and interaction of features can be obtained. *Feature cardinality* provides insight in the state of a flow field, so is e.g. turbulent transition reflected in the number of vortices. Certain features are correlated to other events, e.g. how the number of quasi-streamwise vortices near the wall correlates with with wall-friction. As features are generally made up of a good amount of mesh points, alone for reasons of resolution, storing a list

of parametrised features *reduces the amount of data* by multiple orders of magnitude compared to storing the full flow field data.

A prerequisite for a feature based analysis is the identification of features. Extracting a features as a connected volume encapsulated by a contour is closely related to visualisation techniques. The standard approach of visualising flow features is rendering an isosurface associated with a global isovalue of a scalar indicator function. If for example vortices shall be rendered the indicator function can e.g. be the $Q$-criterion or $\lambda_2$. There have been advances on locally adapting a global threshold by normalizing it with local statistics of the flow field [68, 92] or recently in defining so called objective vortices by locally changing the frame of reference [40, 113]. Topological methods have been applied to unsteady flows to identify features in a post-processing step [11, 63] and in-situ or in-transit [10, 62]. All of these investigations have however been restricted to structured grids. Despite those advances the visualisation of flow features via a fixed global isovalue of a scalar remains in practice the most commonly used technique. This approach however suffers from a significant drawback: the level of such an indicator function usually varies throughout a flow field by multiple orders of magnitude. As a consequence there generally is no single global threshold to extract all features in a given domain. Features can be missed in regions where the indicator function has low values, while multiple features are amalgamated in regions where the indicator function has high values. Topology based feature extraction addresses this issue by defining an individual threshold for each feature based on the topology of the surrounding indicator function.

## 1.3 Tracer

The main objective of the current work is developing Tracer, an in-situ framework for topology based identification and classification of flow features in unsteady CFD data produced by PyFR [107]. PyFR is well suited to producing the flow field, since its highly parallel nature allows simulations of large problems and its high-order capability enables efficient resolution of the unsteady turbulent flow physics. PyFR is known to perform well on Nvidia GPUs. To avoid having to write the data to disk and analysing it in a post-processing step Tracer has the capability to run on GPUs alongside PyFR so the analysis happens concurrently with the simulation. That way insight is extracted on the fly while the data is still in system memory. In addition Tracer has the capability of analysing data read from disk in the form of a vtu file, an unstructured mesh format used in the VTK library [91].

The intensities of flow features generally span multiple orders of magnitude making their identification a challenging task. Tracer overcomes this issue by determining an individual threshold for each feature based on the topology of a scalar indicator function. The topological analysis is conducted via a join tree representation of the scalar field. The join tree is built via a state-of-the-art construction algorithm designed for high performance on shared memory architectures, which has been adapted to enable the processing of CFD data. Furthermore the join tree can be used to integrate quantites over the volume occupied by features.

Having identified flow features, Tracer can visualise them directly by generating png files in-transit. Geometry and location of extracted features are assessed alongside their topological organisation and written to disk in the form of a csv file. The size of such a csv file is multiple

orders of magnitude below the size of a file representing the full flow field. The parametrisation of features in combination with the significant reduction of data allows them to be classified and compared in post-processing at small computational costs.

## 1.4 Outline

The join tree (JT) repersents changes in the topology of superlevel sets of a scalar field as the isovalue associated with the superlevel set is altered. Chapter 2 of this thesis describes in detail what a JT is and how it can be used for the extraction of flow features via a topology based domain segmentation. This chapter furthermore explains how the organisation of a JT can be parametrised and and how geometric measures and integral quantities of features are obtained by accumulation over subtrees. Tracer is relying on the JT construction algorithm parallel peak pruning [24] to build the JT on the GPU. In addition to an explanation of the algorithm, adaptions are presented which are necessary to enable processing of data from CFD. To obtain the correct JT of a scalar field the input data is required to include all maxima and join saddles of the scalar field at the vertices of the mesh. This requirement is not fulfilled by data discretised on meshes containing hexahedral cells. Chapter 3 introduces a new method to identify a super-set of saddle points in the body and on the face of such cells, which is correcting an error in an established method [105]. The corrected method has been presented in [60]. Tracer, the in-situ framework for identifying flow features in CFD data, is presented in Chapter 4. Specifically Tracer's capabilities are highlighted alongside the implementation of the methods. The software is verified using a range of simple and complex tests. On the example of a Taylor-Green vortex simulation it is shown that adding Tracer in-situ to a CFD simulation has a reasonable impact on the usage of system memory and causes only a small overhead in runtime. Chapter 5 demonstrates the advantages of object-oriented visualisation over rendering an isosurface set associated with a global isovalue on the example of the flow around a SD7003 aerofoil. How the turbulent breakdown and decay of a Taylor-Green vortex is reflected in the number of identified features is shown in Chapter 6. Chapter 7 compares the topology of vortices in the near-wall region of a channel flow with the topology of vortices in the channel centre. Furthermore the self-similarity and scaling of vortices with wall-distance are assessed. Finally, Chapter 8 presents a summary and proposes further functionality to be added to Tracer and applications that can be explored using the framework.

# 2 Topology Based Domain Segmentation via Join Trees

Identifying vortices based on a scalar vortex criterion poses the problem of extracting features from a scalar field $f$ in the domain $\Omega$. Identifying such features is related to rendering isosurfaces, which can be interpreted as surfaces of features. Generally the value of $f$ changes significantly throughout $\Omega$, as a consequence features need to be extracted via a locally defined threshold. Thus choosing a global threshold $f_{cut}$ and defining features as areas encapsulated by contours associated with $f_{cut}$ will fail to extract all features in the domain. This problem is illustrated in Figure 2.1, in which the features that shall be identified are mountains on the height function of a pointy body. Choosing $f_{cut}$ high enough to resolve the major peaks on the right, like in Figure 2.1 (a), fails to detect the smaller mountains. However, lowering $f_{cut}$ so the mountains having a lower elevation are detected results in features in the higher region not being resolved as is depicted in Figure 2.1 (b). Topology based domain segmentation is an approach in which an individual threshold is obtained for each feature, such that features are detected and resolved throughout $\Omega$.

Section 2.1 describes advances in topology based domain segmentation leading to the relevance criterion $R$ [73]. For computing $R$ at any given point $i \in \Omega$ the highest maximum of $f$, which can be reached from $i$ via a monotonous path, must be known. Due to the vast number of possible paths, finding those maxima in the physical domain is computationally expensive. The join tree (JT), a graph that tracks topologies of superlevel sets of $f$, can be used as a vehicle to identify

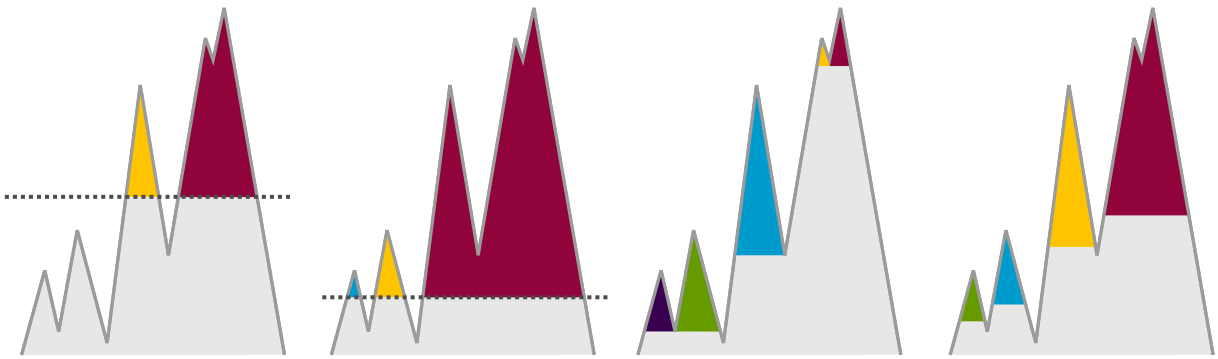



Figure 2.1: Feature extraction on a height function of a pointy body via (a) a medium global threshold $f_{cut}$, which misses the two lowest peaks, (b) a low global $f_{cut}$, which cannot resolve the peaks on the right as individual features, (c) a largest contour segmentation, by which the main peak and side peak on the left are identified as individual features, (d) $R_{cut}$: using Eq. (2.1) the $R$ field is computed. Setting a global threshold $R_{cut}$ returns an individual threshold in $f$ for each feature.

such peaks with less computational costs. Section 2.2 explains what a JT is, presents parameters that describe its organisation and describes how it can be used to efficiently accumulate integral and maximal quantities of regions *above* any given point in $\Omega$. Section 2.3 deals with the construction of JTs from scalar data discretised on linear meshes. Specifically parallel peak pruning (PPP) [24] is described, an algorithm developed to efficiently build JTs on shared memory platforms. Scalar fields from CFD simulations cannot fulfil all of PPP's requirements for an input graph. Section 2.4 discusses the impact of violations of requirements on the resulting JT and provides adaptations that have to be made to PPP in order to process CFD data.

## 2.1 From Object Oriented Visualisation to the Relevance Criterion

Topology based domain segmentation has its roots in object oriented visualisation introduced by Silver [93]. Sliver describes how extracting individual features of interest can not only benefit visualisation purposes, but also help building reduced mathematical models that explain the behaviour of such features. Various benefits of a feature based analysis which were suggested by Silver are provided in Chapter 1. To extract features Silver proposes a method called maxima separation, which assumes that each feature includes exactly a single local maximum. Around each maximum a contour is grown such that the contours are not overlapping. Manders et al. [71] introduced largest contour segmentation, which identifies the largest contour that wraps around a single maximum. Contours are seeded at local maxima iteratively grown by decreasing the isovalue associated with the contour until the contour includes a saddle which separates two or more maxima. The problem with both methods is that also maxima introduced by noise will be identified as individual features and prevent the identification of large features by potentially splitting them up. Such an instance is illustrated in Figure 2.1 (c), where the two highest peaks are identified as individual features rather than as main peak and side peak of a single feature.

Mascarenhas et al. [73] introduced the relevance criterion $R$, which has been applied to extract vortices in a jet in cross-flow simulation by Bremer et al. [14]. $R$ relates the scalar value $f_i$ at point $i$ to the maximum value $f_{max,i}$ that can be reached from $i$ via a monotonously ascending path. The relevance $R_i$ at $i$ is defined as:

$$R_i = \frac{f_i - f_{min}}{f_{max,i} - f_{min}}; \tag{2.1}$$

where $f_{min}$ denotes the global minimum of $f$ in $\Omega$. Identifying $f_{max,i}$ for each point in the physical domain gets computationally expensive even for small data sets. Using the JT of $f$ as a vehicle to do so can significantly reduce the costs for finding $f_{max,i}$. Further information on JTs and what else they can be used for is provided in Section 2.2.

Features can be extracted from the $R$ field by setting a global relevance threshold $R_{cut}$. For each feature $j$ $R_{cut}$ returns an individual threshold $f_{cut,j}$ based on the topology of $f$ in the neighbourhood of $j$. This approach is illustrated in Figure 2.1 (d). Compared with maxima separation and largest contour segmentation feature extraction via $R_{cut}$ is more robust to noise splitting up large structures. However, in neighbourhoods with a low ambient level of $f$ even the smallest local maxima are identified as features. Such small maxima can be occuring in large amounts in data that includes noise. These artefacts can be filtered by e.g. removing all features

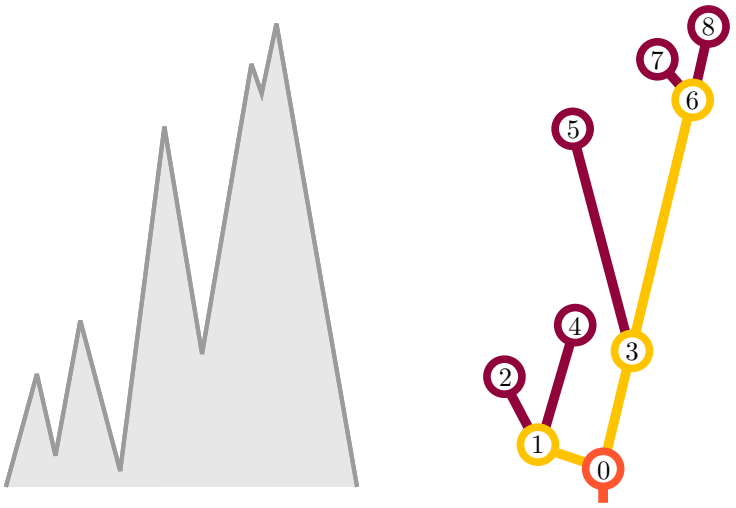
which are smaller than a user-defined size. If the $R$ field is written to file extracted features can be visualised using standard visualisation tools by rendering isosurfaces of $R$.

## 2.2 Join Trees

The theory of JTs is underpinned by Morse Theory [76], which examines changes in the topology of level-sets as $f_{iso}$ is altered. A level-set of a $C^0$ continuous scalar function $f$ in $n$-dimensional space $\mathbb{R}^n$ at some isovalue $f_{iso}$ is the set $\{\mathbf{x} \in \mathbb{R}^n | s(\mathbf{x}) = f_{iso}\}$ and consists of zero, one, or more connected components, the so-called contours. As $f_{iso}$ sweeps through the whole range of $f$ from $+\infty$ to $-\infty$, one can observe contours of $f$ appearing, joining, splitting and disappearing. Isovalues and points at which this happens are called critical isovalues or critical points, respectively. A JT is a graph that tracks appearing and joining of superlevel sets associated with such contours, it therefore records peaks and saddles where peaks meet, so called joins. An example for a JT where $f$ is the height function is provided in Figure 2.2.

### 2.2.1 Horton-Strahler Orders, Horton Ratios and Tokunaga Indices

The organisation of a JT and its arcs is described by Horton-Strahler (HS) orders, Horton ratios [43, 96] and Tokunaga indices [99, 100], all of which originate from hydrology. Those topological parameters have been successfully applied to characterise river networks [112]. In addition to geometric description of flow features e.g. based on volume or shape a topological parametrisation of features could potentially provide further information on the basis of which such features can be classified and compared. The following paragraphs provide a brief overview over how those indicators describe the organisation of JTs. For a comprehensive discussion the reader is referred to [112] and the citations therein.




(a) (b)

Figure 2.2: (a) Pointy body and (b) the JT of its height function. The red colour indicates a HS order of $r = 0$, yellow $r = 1$ and orange $r = 2$. The critical nodes are numbered in ascending order.

### Horton-Strahler Orders

HS orders $r$ are assigned to arcs and their upper critical nodes by iteratively pruning all leaves of the JT down to the root. The order of an arc and its upper end is identical to the number of the iteration in which the pair is pruned. Let us exercise this on the JT of Figure 2.2 (b), in which arcs and nodes are coloured according to their HS order: in iteration 0 all red leaves are pruned. Hence arcs 2-1, 4-1, 5-3, 7-6 and 8-6 alongside their upper nodes will be assigned $r = 0$. The pruned JT has the yellow arcs 6-0 and 1-0 as its leaves. These arcs are pruned and $r = 1$ is assigned to 1-0, 3-0 and 6-3. Now only the root of the JT is left which therefore is the only arc with $r = 2$.

Note that after iteration 0 node 3 has become a regular node, which is why in iteration 1 arcs 6-3 and 3-0 are pruned together. Arcs between critical nodes are called super arc (supA), the union of super arcs which are pruned as a single leave are called hyper arc (hypA). An example for a supA is 6-3, an example for a hypA is 6-0. Arcs can be a supA and a hypA at the same time, an example for this is arc 2-1. More information on supAs and hypAs is provided in Section 2.3.1.

### Horton Ratios

Horton ratios [43, 96] describe the organisation of arcs in the JT. The numerators and denominators of the Horton ratios are made up of the following numbers, which will be called Horton numbers throughout this thesis: $N_r$ is the number of hyper arcs of order $r$ in the tree. $\langle M_r \rangle$ is the average magnitude of all hypAs of order $r$. The magnitude of a hypA $i$ is defined as the number of leaves that can be reached from the lower end of $i$ via a monotone path. $\langle C_r \rangle$ provides the average number of supAs in a hypA of order $r$. Note that all supAs of $r = 0$ are also hypAs and that always $\langle M_0 \rangle = 1$ and $\langle C_0 \rangle = 1$. The Horton ratios are defined as [112]:

$$P_N = \frac{N_r}{N_{r+1}}, \tag{2.2}$$

$$P_M = \frac{\langle M_{r+1} \rangle}{\langle M_r \rangle}, \tag{2.3}$$

$$P_C = \frac{\langle C_{r+1} \rangle}{\langle C_r \rangle}. \tag{2.4}$$

Consider a JT the root of which is of HS order $r_{max}$. Such a JT is called Horton self-similar if $\lim_{r_{max}-r\to\infty} \frac{N_r}{N_{r+1}} = P_N$, $\lim_{r\to\infty} \frac{\langle M_{r+1} \rangle}{\langle M_r \rangle} = P_M$ and $\lim_{r\to\infty} \frac{\langle C_{r+1} \rangle}{\langle C_r \rangle} = P_C$. Horton ratios find application in e.g. hydrology [43, 96, 86, 112] and biology [79].

### Tokunaga Indices

Let $N_{i,j}$ be the number of hypAs with $r = i$ that merge with a hypA of order $r = j$, where $i \leq j$. The Tokunaga index [99, 100] is defined as:

$$T_{i,j} = \frac{N_{i,j}}{N_j}. \tag{2.5}$$

The JT in Figure 2.2 (b) has the non-zero $N_{i,j}$ elements $N_{1,1} = 4$ (hypAs 2-1, 4-1, 7-6 and 8-6), $N_{1,2} = 1$ (hypA 5-3) and $N_{2,2} = 2$ (hypAs 1-0 and 3-0). Its non-zero $T_{i,j}$ elements are $T_{1,1} = \frac{4}{5}$, $T_{1,2} = \frac{1}{2}$ and $T_{2,2} = 1$. A JT is Tokunaga self-similar if all branches of a given order have the same side-branching structure and if

$$P_T = \frac{T_{k+1}}{T_k}, \tag{2.6}$$

is invariant wrt. $i$, with $T_k = T_{i,i+k}$. Tokunaga self-similarity has been studied in amongst other hydrology [86, 112], biology [79] and physics [111].

### 2.2.2 Accumulating Subtree Quantities

Consider the JT $K$ of $f$ defined on $\Omega \in \mathbb{R}^3$ and a node $i$ on $K$. Let $k_i \in K$ be the subtree of $i$, which includes all parts of $K$ that can be reached via an ascending path starting at $i$, including $i$ itself. $I \in \Omega$ is the region in physical space associated with $k_i$. The integral of a scalar field $\sigma$ over the volume occupied by $I$ can be obtained by integrating $\sigma$ over $k_i$.

Carr et al. [21, 22] presented a method to obtain the *hyper volume* of $k_i$, which is defined as the integral of a scalar quantity $\sigma$ over $k_i$. To guarantee the correct result at $i$ all other nodes of $k_i$ need to be processed before $i$. Thus $\sigma$ has to be integrated along hypAs starting at their upper end. All hypAs with $r = l$ must be processed before hypAs with $r = m > l$ and pass their result to their lower end. hypAs with of the same HS order can be processed in parallel.

In practice JTs are constructed from an edge graph of a mesh, on which $f$ is discretised. More information on discretisation of scalar data on meshes with piecewise linear interpolant is provided in Chapter 3. The JT consequently will consist of discrete nodes which are connected by arcs. Carr et al. approximate the integral along arcs via a Riemann sum over all nodes $p \in k_i$. With the scalar value $\sigma_p$ the volume $V_p$ associated with $p$, the *hyper volume* is approximated by:

$$\int_I \sigma \, d\mathbf{x} \approx \sum_{p \in k_i} V_p \sigma_p. \tag{2.7}$$

If the summation is done via a prefix sum along the hypAs the *hyper volume* of the subtree of every node in the JT is known. In addition this method can be used to find minima or maxima of the subtrees of all nodes by replacing $\sum$ with the respective operator and setting $V_p = 1$ for all nodes.

### 2.2.3 Computing the Relevance Field via Join Trees

The Relevance criterion $R_i$ can be computed at each point $i$ of a JT via Eq. (2.1). The maximum $f_{max,i}$ of the subtree of $i$ can be obtained using the method by Carr et al. [21, 22] presented in Section 2.2.2. Since on a given supA every point has the same subtree maximum, $f_{max,i}$ can be passed down on minimal version of the JT consisting of supAs only. Bremer et al. [14] have shown that any path in a JT which is monotone in $f$ will also be monotone in $R$. Thus a feature extracted via a global relevance threshold $R_{cut}$ can be defined as the union of all points which belong to the subtree of a cut through an arc at $R_{cut}$. Discontinuities in $R$ can arise on saddle points. This however only affects the smoothness of isosurfaces of $R$ if rendered.

## 2.3 Constructing Join Trees of Discretised Scalar Fields on Shared-Memory Platforms via Parallel Peak Pruning

The objective of the current work is the development of Tracer, a software analysing CFD data generated on GPUs. Hence a JT construction algorithm shall be implemented that performs well on shared-memory platforms. Such algorithms operate on an input graph $\mathbb{G}$ consisting of verices, on which $f$ is provided as discrete values and edges connecting those vertices.

JT construction algorithms based on a sweep and merge approach [20] sequentially add vertices sorted by $f$ to a union-find data structure. Gueunet et al. [38, 39] parallelised the sweep through $f$ splitting up the domain by ranges of $f$ and building individual JTs for each range. Smirnov and Morozov [94] presented an algorithm constructing the JT on shared-memory platforms in a new format called triplet merge tree representation. Maadasamy et al. [70] first construct a topology graph, which includes all critical points and monotone paths between them. The identification of saddles and their monotone paths to maxima is carried out on the GPU, while the JT is assembled on the host. The strategy in developing Tracer is to first aim for a version which constructs the JT locally on a single GPU via a shared memory algorithm. To enable the topological analysis of flow simulations running on multiple GPUs Tracer's capabilities shall be extended in a later step. Local JTs shall be stitched together via a distributed memory algorithm, for example with the distributed merge tree [78]. To build the JT locally on a GPU Tracer employs the shared memory algorithm parallel peak pruning (PPP), which was introduced by Carr et al.[24, 23] to construct the unaugmented JT and later extended to enable the construction of augmented trees [18]. In PPP peaks of $f$ are pruned incrementally down to their highest saddle until only a trunk is left. This procedure is equivalent to determining the HS order of hypAs as described in Section 2.2.1. In the resulting data structure arcs of the same HS order are grouped together, enabling an efficient analysis of the JT.

Section 2.3.1 presents the data structure the JT will be provided in as suggested in [18], Section 2.3.2 details how this format is obtained from $\mathbb{G}$ via the JT construction algorithm of PPP. Sorting of data by multiple indices will recur throughout the description of PPP and its implementation. The convention used in this thesis for sorting by a primary index $a$ and a secondary index $b$ will be that the data is sorted by $(a, b)$.

### 2.3.1 Output Tree Format: Hyper, Super, Augmented and Physical Structure

PPP constructs the JT in the form of the hyper structure, the super structure and the augmented structure [18]. The input graph $\mathbb{G}$ can be interpreted as a fourth, physical structure. The format in which they are stored is illustrated in Figure 2.3, the content is interpreted as follows:

- *The hyper structure (hypS)*
  stores the hypAs sorted by their HS order. A hypA is defined by its upper end and its lower end, both of which are critical nodes. The upper ends of hypAs are hyper nodes (hypN). The lower ends are not necessarily upper ends of a different hypA, thus they are guaranteed to be super nodes (supN) and can also be hypNs.

- *The super structure (supS)*
  stores the supAs, bearing the full information of the *unaugmented* JT. A supA is defined

by its upper end and its lower end, both of which are supNs. The supAs are sorted by ($r$, hypA ID, $f_{\text{upper end}}$ [descending]) so that all supAs of a hypA appear together in descending order.

- *The augmented structure (augS)*
  stores the fully augmented JT. All nodes are stored in an array sorted by ($r$, hypA ID, supA ID, $f$ [descending]). Nodes belonging to the same supA are grouped together and appear in descending order. Furthermore each node stores the ID of the hypA and the supA they belong to. Each node points to its right neighbour in the augS if the neighbour lies on the same hypA. If the right neighbour belongs to a different hypA, the node points to the lower end of its hypA.

The hypS, supS and augS of the JT from Figure 2.2 (b) is provided in Figure 2.4.

### 2.3.2 Parallel Peak Pruning Join Tree Algorithm

PPP iteratively pairs each peak with its governing saddle and prunes the peak down to that saddle. The governing saddle of a peak $p$ is the highest saddle from which $p$ can be reached via

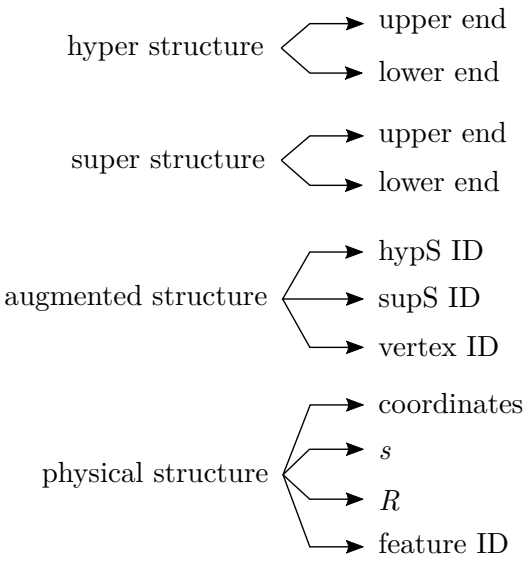



Figure 2.3: Formats of tree structures.

| hyper structure | | super structure | | augmented structure | | |
|---|---|---|---|---|---|---|
| upper end | lower end | upper end | lower end | hypAID | supAID | vertID |
| 2, | 1 | 2, | 1 | 0, | 0, | 2 |
| 4, | 1 | 4, | 1 | 1, | 1, | 4 |
| 5, | 3 | 5, | 3 | 2, | 2, | 5 |
| 7, | 6 | 7, | 6 | 3, | 3, | 7 |
| 8, | 6 | 8, | 6 | 4, | 4, | 8 |
| 1, | 0 | 1, | 0 | 5, | 5, | 1 |
| 6, | 0 | 6, | 3 | 6, | 6, | 6 |
| 0, | root | 3, | 0 | 6, | 7, | 3 |
| | | 0, | root | 7, | 8, | 0 |

Figure 2.4: hypS, supS and augS of the JT from Figure 2.2 (b).

a monotonously ascending path. Pruning converts the saddles into maxima or regular points. The iterations continue until there are no saddles left. A peak-saddle pair corresponds to the upper and lower end of a hypA. After having identified all peak-saddle pairs, the hypAs are assembled to build the hypS and supS of the JT. In a final step regular points are added to the JT forming the augS. The necessary steps to achieve the above are illustrated in Figure 2.5 and are explained in the following paragraphs. For a more comprehensive discussion the reader is referred to [18, 23, 24].

1. *monotone path construction*
   Each vertex $i$ is assigned to a peak, which can be reached from $i$ via a monotonously ascending path.

2. *critical topology graph construction*
   To improve efficiency the JT construction is carried out on a critical topology graph (CTG). A topology graph consists of critical points and monotonous paths between them [25]. The CTG consists of peaks, saddle candidates and monotone paths between them. Consider the set $E$ of edges which have vertex $i$ as lower end. $i$ is a saddle candidate, if not all upper ends of $E$ are assigned to the same peak. If $i$ is a saddle candidate, $i$, all peaks assigned to the upper ends of $E$ and monotone paths from $i$ to these peaks are added to the CTG.

3. *trunk construction iteration*
   The peaks of the CTG are incrementally paired with their governing saddle. The governing saddle of a peak $i$ is identified by selecting the highest saddle candidate that is connected to $i$ via a monotone path. To do so all paths of the CTG are sorted by (vertex ID of upper end, $f_{\text{lower end}}$). The peak-saddle pairs are added as upper and lower end of a hypA to the hypS, additionally both vertices are flagged as supNs. The peaks are pruned by removing them from the CTG and redirecting all upper ends of paths to peaks of the pruned CTG. Former saddle candidates which are guaranteed to be regular points of the pruned CTG and their paths are also removed from the CTG. To keep track of the hypA which they belong to, such former saddle candidates have their associated peak updated until the hypA they lie on gets pruned.
   These steps are repeated until there are no saddle candidates are left and the CTG only consists of a trunk.

4. *building the hyper structure*
   The hypS is a direct result from *trunk construction iteration.*

5. *building the super structure*
   To obtain the supS arcs connecting the supNs need to be constructed. Each supN was assigned to a peak in *monotone path construction* which was updated during *trunk construction iteration.* Those peaks are upper ends of hypAs, the assignment of the supNs to peaks is changed to the ID of the associated hypA. The supNs are stored in an array $a$ which is sorted by (assigned hypA $hyp$, $f$ [descending]). Each supN $a_i$ is the upper end of a supA $i$. The lower end of $i$ is the right neighbour $a_j = a_{i+1}$ of $a_i$, unless $hyp_j \neq hyp_i$, in which case the lower end of $i$ is the lower end of $hyp_i$.

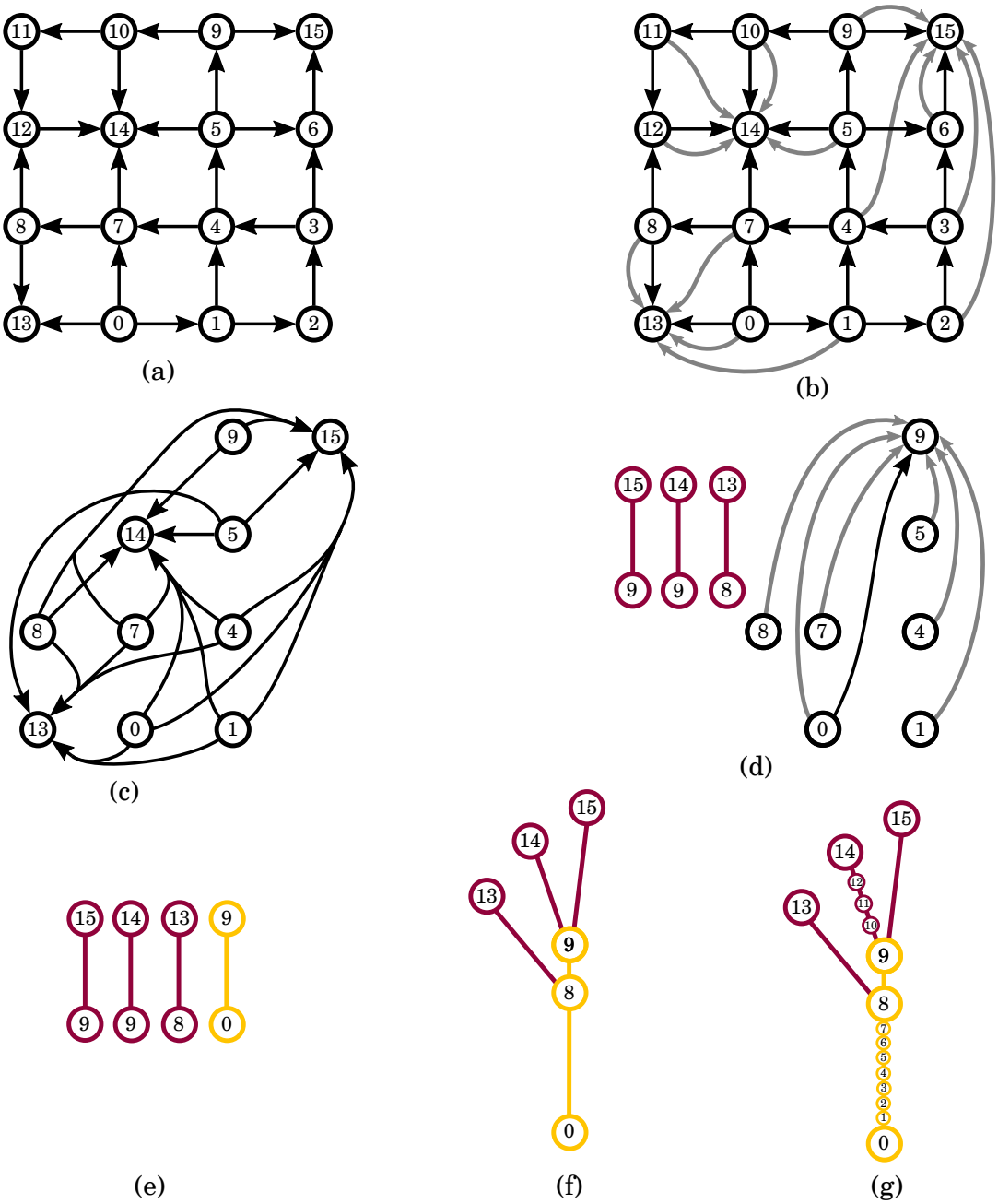



Figure 2.5: Steps for building a JT using PPP on the edge graph depicted in (a). The function value of vertices is identical to their ID provided in the circles, edges are depicted as black arrows pointing in ascending direction. (b) During monotone path construction each vertex is assigned to a peak, assignments are indicated with grey arrows. (c) The CTG is constructed with saddle candidates, peaks and monotone paths from saddle candidates to peaks. (d) During the first iteration of trunk construction three peaks get paired with their governing saddle, the pairs are recorded has hypAs. The peaks get pruned from the CTG and the remainder vertices get reassigned to the new peak at 9. The CTG is updated, only the minimum at 0 and the maximum at 9 remain as a critical points. (e) In the second iteration the pair 9-0 is added to the hypAs. (f) The JT is assembled by constructing the supS using the asignments of lower ends of hypAs to pekas. (e) The JT is augmented using information of assigned peaks of regular vertices.

6. *augmenting the tree with regular nodes*
   Also all regular vertices were assigned to a peak in *monotone path construction*. Again those peaks are replaced with their associated hypA ID. Since the peaks of regular nodes were not updated during *trunk construction iteration* the vertices are be assigned to a hypA of their subtree with $r = 0$. The correct hypA is identified as follows: consider a vertex $i$ with scalar value $f_i$ which is assigned to hypA $hyp_i$ with lower end $l$. If $f_l < f_i$ then $i$ is assigned to the correct hypA. If $f_l > f_i$ then $hyp_i$ is replaced with the hypA $hyp_l$ to which $l$ belongs to. This procedure is repeated until $f_l < f_i$ is fulfilled. Similarly the correct supA of $i$ is found by repeating the same procedure with all supAs belonging to $hyp_i$. Finally augS is obtained by sorting the vertices by (hypA ID, supA ID, $f$ [descending]).

## 2.4 Enabling Parallel Peak Pruning to Process CFD Data

JT construction requires $\mathbb{G}$ to meet certain demands [19]:

i. The edges of $\mathbb{G}$ represent paths in $\Omega$ that are monotone in $f$.

ii. The vertices and edges of $\mathbb{G}$ contain all joins and local maxima of $f$.

iii. For any scalar value $f_k$, two vertices $i, j \in \mathbb{G}$ are connected above $f_k$ iff $i, j$ are connected in the mesh by a path above $f_k$.

In addition the vertices need to be unambiguously sortable by having unique scalar values. Data from CFD discretised on unstructured meshes however can include:

- **non-unique vertex values**.
- **non-simplicial cells**, causing the presence of off-vertex saddles.
- **multiple disconnected subdomains**.

The next paragraphs layout in what way violations in input graph requirements affect the resulting JT and which adaptations have to be made to PPP. The implementation of the adapted PPP algorithm in Tracer is described in Section 4.2.3.

### Non-Unique Vertex Values

The input graph can include non-unique scalar function values at the vertices. To make the vertices unambiguously sortable, simplicity can be simulated [33] by comparing the IDs of the vertices in case their scalar values are identical.

### Non-Simplicial Cells

Data from CFD can come on three-dimensional unstructured meshes and includes non-simplicial cells. Such cells can have join saddles in the volume or on faces of cells. Chapter 3 describes a way to identify such off-mesh saddles in hexahedral cells. If interpolation of function values is dismissed in the volume and on the faces of cells and only the edge graph is used to construct the JT the differences to the *continuous* JT are expected to be minor. For the purpose of using JTs to find individual thresholds for features such differences are negligible.

### Multiple Disconnected Subdomains

In many cases regions can be dismissed from containing features based on the value of $f$ alone. If e.g. vortices shall be identified in a $Q$-criterion field, regions where $Q < 0$ will not include any vortices. Constructing the JT on regions above a global threshold $f_{min}$ returns an individual JT for each connected region with $f > f_{min}$. These regions have positive values of $R$ and will therefore be called *relevant* regions throughout this thesis. As a consequence of processing all relevant regions simultaneously the JTs are returned in an interleaved format. The individual arcs are not grouped by the relevant region to which they belong, but are grouped according to their HS order. That way JTs of multiple individual relevant regions can be built and analysed in parallel.

During *trunk construction iteration* upper ends of root hypAs will not be paired with a saddle and hence do not have a HS order assigned to them. Since upper ends of hypAs of order $r$ are lower ends of at least two hypAs of order $r - 1$ the HS order of root hypAs can be obtained via the lower ends of non-root hypAs: for each upper end $i$ of a root hypA $j$ all elements which have $i$ as their lower end are identified in the list of peak-saddle pairs. As the list is ordered by HS order, the HS order of the last two of these pairs $k$ and $l$ will be identical $r_k = r_l$. Hence $r_j = r_l + 1$. Once all root hypAs have their correct HS order assigned, they can be added to the peak-saddle pairs of the same HS order.

Relevant regions the JT of which has a root of HS order $r_{max} = 0$ contain only a single maximum. This maximum is not included in the list of peak-saddle pairs generated in *trunk construction iteration*. Hence maxima of regions with $r_{max} = 0$ need to be added as upper ends to the list of root hypAs.

# 3 Identifying 1-Saddles and 2-Saddles in Meshes of unstructured Linear Lagrange Hexahedra

This chapter introduces a method that correctly identifies all 1-saddles and 2-saddles in cell bodies and on cell faces for scalar data on unstructured hexahedral meshes. This method is correcting an error in an established method [105], was presented on a conference and has been published in [60]. 1-saddles and 2-saddles are defined as follows [32]: consider the local neighbourhood of a point $p$ with function value $f_p$. The neighbourhood of $p$ can have regions $A^+$ in which $f(x \in A^+) > f_p$ and regions $A^-$ in which $f(x \in A^-) < f_p$. $p$ is called a 1-saddle if $A^+$ is comprised of a single connected component and $A^-$ is comprised of two connected components. 1-saddles consequently are a super set of split saddles. $p$ is called a 2-saddle if $A^+$ is comprised of two connected components and $A^-$ is comprised of a single connected component. 2-saddles consequently are a super set of join saddles. Examples for local neighbourhoods of 1-saddles, 2-saddles, regular points and extrema are illustrated in Figure 3.1.

Motivation for identifying off-vertex 1-saddles and 2-saddles is provided in Section 3.1 alongside background information of the discretisation on hexhedral meshes. Section 3.2 details the contour tree (CT), Section 3.3 presents a brief overview of methods to build the CT on hexhedral meshes. Section 3.4 provides the mapping of a hexahedron to a trilinear cell in reference space, and describes how to identify and classify body saddles. Subsequently it explains how to identify critical points on faces in the local context of a single hexahedron, and then presents the main contribution of this chapter - a process for classifying critical points on faces in the local context of a single hexahedron, and a process whereby this information can be used to identify and classify critical points on faces in the global context of the entire mesh. Finally it describes how to handle the case when a body saddle coincides with a face. Section 3.5 proceeds to compare the

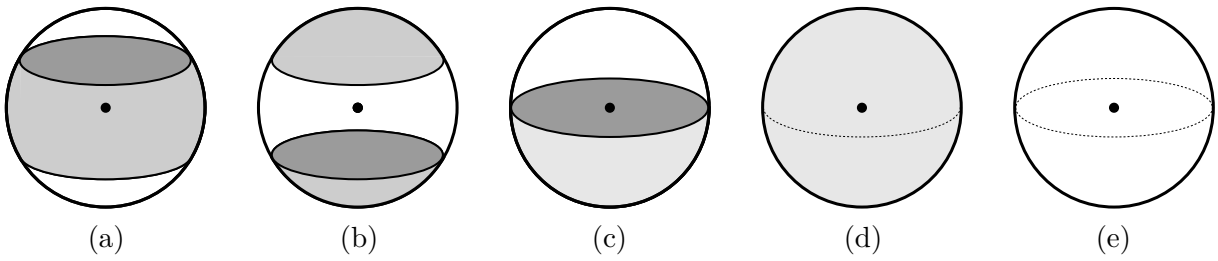



Figure 3.1: Local pictures of $p$ according to [32] depicted by a small sphere around $p$. Shaded areas indicate regions on which the sphere has function values higher than $f_p$, white areas indicate regions on which the sphere has function values lower than $f_p$. $p$ is (a) a 1-saddle, (b) a 2-saddle, (c) a regular point, (d) a minimum and (e) a maximum.

current approach for identifying/classifying critical points on faces with the established method of Weber [105]. It is shown that the established approach in fact contains an error, which the current approach corrects. Finally, Section 3.6 presents a series of examples and experiments that demonstrate the utility of the current approach.

## 3.1 Background and Motivation

Consider a level-set of a $C^0$ continuous scalar function $\mathcal{F}$ in $n$-dimensional space $\mathbb{R}^n$. In practice, the domain $\Omega$ on which scalar data is provided is often subdivided into a mesh $\mathbb{M}$ of $n_e$ non-overlapping cells $\tau_i$ of various cell types $\eta$. For each cell type one can construct a reference cell $\widetilde{\omega}_\eta$ within which the scalar data is represented in a finite dimensional function space $\widetilde{\mathbf{K}}_\eta$, which can be transformed to the function space $\mathbf{K}_i$ for each cell $\tau_i$. Let $f$ be the projection of $\mathcal{F}$ onto $\bigcup_{i=0}^{n_e-1} \mathbf{K}_i$. The correct JT of $f$ can only be constructed from an edge graph $\mathbb{G}$ extracted from $\mathbb{M}$ if $\mathbb{G}$ contains all maxima and join saddles of $f$. Building the JT from an edge graph which does not include all of those points can still return a JT with small, for the present purposes neglibile errors. This however is not true for building the CT. The CT, which is explained in more detail in Section 3.2, is a combination of the JT and the split tree, which is the JT of $-f$ mirrored on $f = 0$. If join or split saddles, which are saddles separating minima, are missing in $\mathbb{G}$, the construction of a CT can fail. One way to obtain a CT from $\mathbb{G}$ that is missing such points is augmenting $\mathbb{G}$ with 1-saddles, which are a superset of split saddles, and 2-saddles, which are a superset of join saddles [32].

## 3.2 Contour Trees

A level-set of a $C^0$ continuous scalar function $f$ in $n$-dimensional space $\mathbb{R}^n$ at some isovalue $f_{iso}$ is the set $\{\mathbf{x} \in \mathbb{R}^n | s(\mathbf{x}) = f_{iso}\}$ and consists of zero, one, or more connected components, the so-called contours. In 2D these contours are isolines, in 3D isosurfaces. As $f_{iso}$ sweeps through the whole range of $f$ from $+\infty$ to $-\infty$, one can observe contours appearing at maxima, joining and splitting at saddles, and disappearing at minima. A CT is a graph that tracks all these changes. Like the theory of JTs also the theory of CTs is underpinned by Morse Theory [76]. Recall that points at which the topology of level-sets change are called critical points, all other points are regular points.

Figure 3.2 shows isolines of a 2D scalar field and its corresponding CT. The scalar field contains two local maxima and one local minimum. Consider now topology changes of the isolines as $f_{iso}$ sweeps from $+\infty$ to $-\infty$. At an isovalue of 3 the maximum on the left hand side appears (node 3 on the associated CT). At an isovalue of 2.5 the maximum on the right hand side appears (node 2.5), the contour of which splits up into two at an isovalue of 2 (node 2). At an isovalue of 1.5 one of these contours disappears at the minimum (node 1.5), while the other contour joins with at an isovalue of 1 with the contour associated with the left hand side peak (node 1).

A graph $\mathbb{G}$ constructed from the vertices and edges of all mesh cells in $\Omega$ can be used for CT construction, if $\mathbb{G}$ satisfies the following requirements [19]:

i. The edges of $\mathbb{G}$ are monotone in $f$.

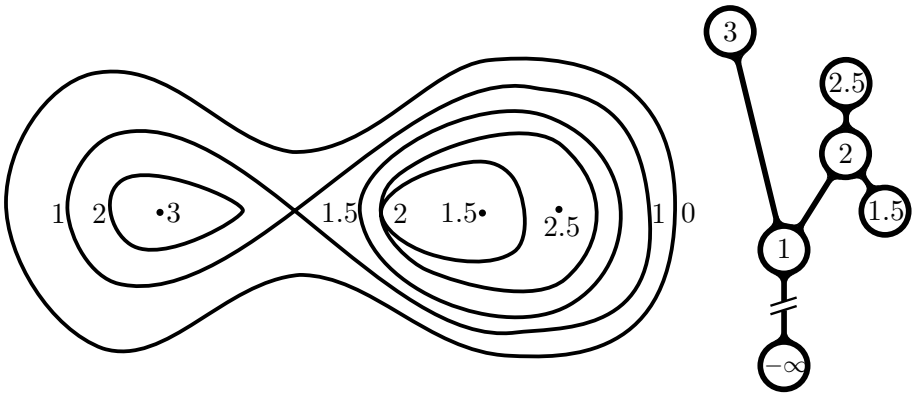



Figure 3.2: Isolines of a 2D scalar field (left) and its corresponding CT (right). Contours of the extreme values at 1.5, 2.5, and 3 collapse to points.

ii. The vertices and edges of $\mathbb{G}$ contain all joins and local maxima of $f$.

iii. For any scalar value $k$, two vertices $V_i, V_j \in \mathbb{G}$ are connected above $k$ iff $V_i, V_j$ are connected in the mesh by a path above $k$.

iv. The vertices and edges of $\mathbb{G}$ contain all splits and local minima of $f$.

v. For any scalar value $k$, two vertices $V_i, V_j \in \mathbb{G}$ are connected below $k$ iff $V_i, V_j$ are connected in the mesh by a path below $k$.

These requirements are inherently satisfied for scalar data on meshes consisting of simplex cells with linear $\widetilde{\mathbf{K}}_\eta$, where all saddles occur at cell vertices. However, hexahedral cells with trilinear $\widetilde{\mathbf{K}}_\eta$ can contain additional off-vertex saddle points in cell bodies and on cell faces. If these additional saddles are identified and used to augment the original mesh along with associated new edges, then it will also fulfil i-v and can be used for CT construction. Moreover, note that a CT is a merge of the JT, which only tracks contour joins, and a split tree, which only tracks contour splits. To construct the JT, $\mathbb{G}$ only needs to satisfy requirements i-iii. Hence, for scalar data on unstructured meshes containing hexahedra, $\mathbb{G}$ only needs be augmented with additional join saddles, and associated new edges connecting them to the maxima of their face (face saddles) or cell (body saddles). Similarly, to construct the split tree $\mathbb{G}$ only needs to satisfy requirements i, iv, and v. Hence, for scalar data on meshes containing hexahedra, $\mathbb{G}$ only needs be augmented with additional split saddles and associated new edges connecting them to the minima of their face or cell. Consequently, classifying critical points avoids addition of un-needed split saddles when constructing join trees and visa versa, as well as un-needed edges, which connect the critical points to the mesh, thus reducing the computational cost of CT construction.

Determining whether a critical point is a join or a split saddle requires a global view of the scalar field. However, critical points can be classified using only local data, and hence with lower computational cost, into minima, maxima, 1-saddles, and 2-saddles; where 1-saddles are a superset of split saddles and 2-saddles are a superset of join saddles [32], and where, importantly, adding 1-saddles and 2-saddles which are not split or join saddles to the mesh does not violate i)-v). While minima and maxima occur only on vertices, 1-saddles and 2-saddles can occur in cell bodies, and on cell faces for scalar data on meshes of hexahedral cells.

## 3.3 Related Work

Pascucci and Cole-McLaughlin [85] were the first to introduce a method to build the CT on meshes of hexahedra. They describe a divide-and-conquer algorithm, which successively combines pairs of trees starting with cell-local join/split trees, which were constructed using an oracle. As the combination iterations proceed, however, the trees to be combined grow in size, and the number of independent combinations that can be carried out simultaneously decreases, reducing available parallelism. Hence this method is ill-suited for modern hardware architectures such as GPUs. Moreover, they present an oracle which, for a given hexahedron, returns the number of saddles based on the number of maxima in that cell. Consequently the saddle value and location as well as the position of the critical points in the join and split tree need to be computed explicitly. Subsequently, Carr and Snoeyink [19] presented two approaches which can be used as an oracle to build a cell-local join and split tree: the finite state machine and the widget. Finite state machines are formulated for sweep-based CT construction methods storing the connected components of each cell as the underlying scalar field is swept. However, more recent CT construction methods like PPP [24, 23] do not sweep the through the data, hence a finite state machine is not applicable for such algorithms. A widget constructs the topologically most complicated cell adding points and edges for all possible saddles to each cell. Consequently, adding a widget to each cell can significantly increase the number of edges in the mesh leading to a significant increase in computational cost and storage requirements. Many of the additional edges can be discarded if the widget is used to compute an cell-local join/split graph first. However, this requires determining whether these additional points are regular or critical, and computation of their scalar values. Finally other algorithms to construct a CT on meshes of trilinear interpolants analyse the link of the vertices [23, 70]. On regular meshes the neighbourhood and therefore the link of each vertex can be analysed efficiently, on unstructured meshes, however, extracting the link of a vertex can be computationally expensive or, if done initially for each vertex, require a substantial amount of memory.

## 3.4 Off-Vertex Saddle Points on Unstructured Meshes of Hexahedra

Critical points can occur within the body and on the faces of hexahedra with trilinear $\widetilde{\mathbf{K}}_\eta$. Section 3.4.1 details how to identify cell-local saddles within the body of hexahedra, body saddles, and a novel method to classify them. Section 3.4.2 presents the main contribution of this chapter, which is a new approach to identify and classify critical points on the faces of hexahedra. Finally, Section 3.4.3 details the special case of body saddles that coincide with the faces of hexahedra.

### 3.4.1 Body Saddles

Body saddles in hexahedra are located at points within the body of the hexahedra where all gradient components of the associated interpolation function are equal to zero. Such saddles are guaranteed to be of degree 3, i.e. merging exactly two extrema. The following paragraph details the mapping of a given hexahedron from physical space to reference space, within which body

saddles are identified and classified. Subsequently the identification of body saddles is explained. Finally, the classification of body saddles as either peak or trough type is be presented.

### Mapping

Consider a given hexahedron $\mathcal{H}$ in physical space $\mathbf{x} = (x, y, z)$ with vertices $V_i$ as per Figure 3.3 (a), and function values $F_i$ at each of its vertices $V_i$. Now consider that $\mathcal{H}$ is mapped to a hexahedron $\widetilde{\mathcal{H}}$ in reference space $\tilde{x} = (\tilde{x}, \tilde{y}, \tilde{z})$ that takes the form of an axis-aligned unit cube with vertices $V_i$ as per Figure 3.3 (b), retaining function values $F_i$ at each of its vertices $V_i$.

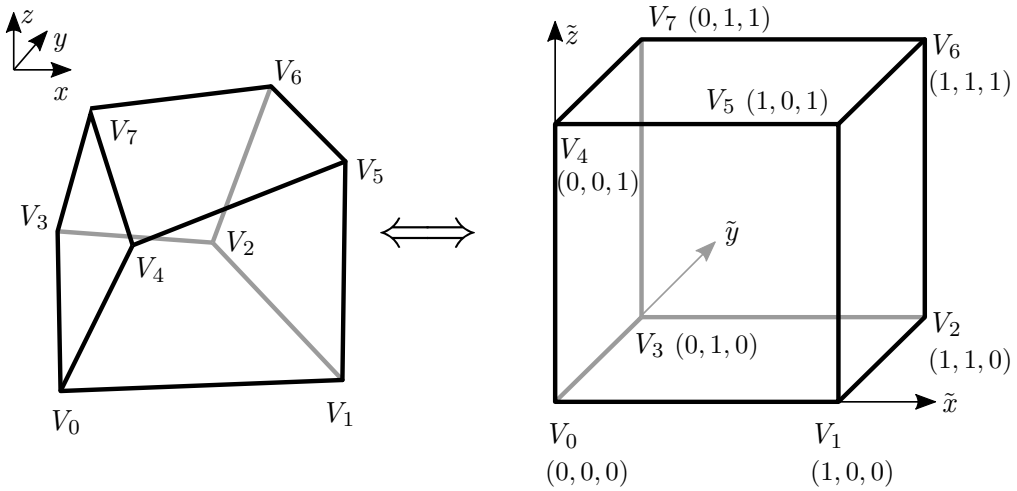
Points within $\mathcal{H}$ are related to points within the $\widetilde{\mathcal{H}}$ via

$$\begin{aligned}&\mathbf{x}(\tilde{x},\tilde{y},\tilde{z}) = \\ &(1-\tilde{z})[(1-\tilde{x})(1-\tilde{y})\mathbf{x_0} + \tilde{x}(1-\tilde{y})\mathbf{x_1} + \tilde{x}\tilde{y}\mathbf{x_2} + (1-\tilde{x})\tilde{y}\mathbf{x_3}] \\ &\quad +\tilde{z}\ [(1-\tilde{x})(1-\tilde{y})\mathbf{x_4} + \tilde{x}(1-\tilde{y})\mathbf{x_5} + \tilde{x}\tilde{y}\mathbf{x_6} + (1-\tilde{x})\tilde{y}\mathbf{x_7}],\end{aligned} \tag{3.1}$$

where $\mathbf{x}_i$ are the locations of the vertices $V_i$ of $\mathcal{H}$. Moreover, the interpolation function within $\widetilde{\mathcal{H}}$ can be written as

$$\widetilde{F}(\tilde{x}, \tilde{y}, \tilde{z}) = a\tilde{x}\tilde{y}\tilde{z} + b\tilde{x}\tilde{y} + c\tilde{y}\tilde{z} + d\tilde{x}\tilde{z} + e\tilde{x} + f\tilde{y} + g\tilde{z} + h, \tag{3.2}$$

where




(a) (b)

Figure 3.3: A hexahedron $\mathcal{H}$ in physical space $\mathbf{x} = (x, y, z)$ with vertices $V_i$ (a) and its associated mapped hexahedron $\widetilde{\mathcal{H}}$ in reference space $\tilde{x} = (\tilde{x}, \tilde{y}, \tilde{z})$ that takes the form of an axis-aligned unit cube with vertices $V_i$ (b).

$$
\begin{aligned}
a &= -F_0 + F_1 - F_2 + F_3 + F_4 - F_5 + F_6 - F_7, \\
b &= \phantom{-}F_0 - F_1 + F_2 - F_3, \\
c &= \phantom{-}F_0 - F_3 - F_4 + F_7, \\
d &= \phantom{-}F_0 - F_1 - F_4 + F_5, \\
e &= -F_0 + F_1, \\
f &= -F_0 + F_3, \\
g &= -F_0 + F_4, \\
h &= \phantom{-}F_0.
\end{aligned}
\tag{3.3}
$$

A detailed discussion of $\widetilde{F}$ and its isosurfaces is provided by Nielson [80]. Note that points within $\widetilde{\mathcal{H}}$ where all three gradient components of the interpolation function are zero map directly to points within $\mathcal{H}$ where all three gradient components of the interpolation function are zero, hence identification of a body saddle in $\widetilde{\mathcal{H}}$ implies the existence of a body saddle in $\mathcal{H}$. Moreover, since the mapping from physical to reference space of a single cell is continuous and non-singular, the topology of the scalar field and hence the type of a saddle identified in $\widetilde{\mathcal{H}}$ is the same as the type of its associated saddle in $\mathcal{H}$.

#### Identifying Body Saddles

This section summarises the method for locating body saddles according to Nielson [80]. Coordinates of saddles in $\widetilde{F}(\tilde{x}, \tilde{y}, \tilde{z})$ are given by the roots of

$$
\nabla \widetilde{F}(\tilde{x}, \tilde{y}, \tilde{z}) = \begin{bmatrix} a\tilde{y}\tilde{z} + b\tilde{y} + d\tilde{z} + e \\ a\tilde{x}\tilde{z} + b\tilde{x} + c\tilde{z} + f \\ a\tilde{x}\tilde{y} + d\tilde{x} + c\tilde{y} + g \end{bmatrix}. \tag{3.4}
$$

If

$$
a_{\tilde{x}} a_{\tilde{y}} a_{\tilde{z}} < 0, \tag{3.5}
$$

where

$$
\begin{aligned}
a_{\tilde{x}} &= ae - bd, \\
a_{\tilde{y}} &= af - bc, \\
a_{\tilde{z}} &= ag - cd,
\end{aligned}
\tag{3.6}
$$

then either one or two roots of $\nabla \widetilde{F}$ exist. Specifically, if $a_{\tilde{x}} a_{\tilde{y}} a_{\tilde{z}} < 0$ and $a = 0$ there exists exactly one root, and hence exactly one saddle $s_0$, at $(\tilde{x}, \tilde{y}, \tilde{z}) = (\tilde{x}_{s_0}, \tilde{y}_{s_0}, \tilde{z}_{s_0})$ where

$$
\begin{aligned}
\tilde{x}_{s_0} &= \frac{ce - bg - df}{2bd}, \\
\tilde{y}_{s_0} &= \frac{df - bg - ce}{2bc}, \\
\tilde{z}_{s_0} &= \frac{bg - ce - df}{2cd},
\end{aligned}
\tag{3.7}
$$

with the function value at the saddle

$$\begin{aligned} k_{s_0} &= \widetilde{F}(\tilde{x}_{s_0}, \tilde{y}_{s_0}, \tilde{z}_{s_0}) \\ &= \frac{1}{4bcd}\Big((bg+df)^2 + (ce)^2 - 2cdef - 2bceg\Big) + h. \end{aligned} \tag{3.8}$$

However, if $a_{\tilde{x}}a_{\tilde{y}}a_{\tilde{z}} < 0$ and $a \neq 0$ there exists exactly two roots, and hence exactly two saddles $s_+$ and $s_-$, at $(\tilde{x}, \tilde{y}, \tilde{z}) = (\tilde{x}_{s_+}, \tilde{y}_{s_+}, \tilde{z}_{s_+})$ and $(\tilde{x}, \tilde{y}, \tilde{z}) = (\tilde{x}_{s_-}, \tilde{y}_{s_-}, \tilde{z}_{s_-})$, respectively, where

$$\begin{aligned} \tilde{x}_{s_+} &= -\frac{c}{a} + \frac{\sqrt{-a_{\tilde{x}}a_{\tilde{y}}a_{\tilde{z}}}}{aa_{\tilde{x}}}, \\ \tilde{y}_{s_+} &= -\frac{d}{a} + \frac{\sqrt{-a_{\tilde{x}}a_{\tilde{y}}a_{\tilde{z}}}}{aa_{\tilde{y}}}, \\ \tilde{z}_{s_+} &= -\frac{b}{a} + \frac{\sqrt{-a_{\tilde{x}}a_{\tilde{y}}a_{\tilde{z}}}}{aa_{\tilde{z}}}, \end{aligned} \tag{3.9}$$

and

$$\begin{aligned} \tilde{x}_{s_-} &= -\frac{c}{a} - \frac{\sqrt{-a_{\tilde{x}}a_{\tilde{y}}a_{\tilde{z}}}}{aa_{\tilde{x}}}, \\ \tilde{y}_{s_-} &= -\frac{d}{a} - \frac{\sqrt{-a_{\tilde{x}}a_{\tilde{y}}a_{\tilde{z}}}}{aa_{\tilde{y}}}, \\ \tilde{z}_{s_-} &= -\frac{b}{a} - \frac{\sqrt{-a_{\tilde{x}}a_{\tilde{y}}a_{\tilde{z}}}}{aa_{\tilde{z}}}, \end{aligned} \tag{3.10}$$

with the function values at the saddles given by

$$\begin{aligned} k_{s_+} &= \widetilde{F}(\tilde{x}_{s_+}, \tilde{y}_{s_+}, \tilde{z}_{s_+}) \\ &= \frac{1}{a^2}\left(-a(bg+ce+df) + 2bcd + 2\sqrt{-a_{\tilde{x}}a_{\tilde{y}}a_{\tilde{z}}}\right) + h, \end{aligned} \tag{3.11}$$

and

$$\begin{aligned} k_{s_-} &= \widetilde{F}(\tilde{x}_{s_-}, \tilde{y}_{s_-}, \tilde{z}_{s_-}) \\ &= \frac{1}{a^2}\left(-a(bg+ce+df) + 2bcd - 2\sqrt{-a_{\tilde{x}}a_{\tilde{y}}a_{\tilde{z}}}\right) + h. \end{aligned} \tag{3.12}$$

If all three coordinate components of any saddles are within the interval $]0,1[$ then $\widetilde{\mathcal{H}}$ contains a body saddle at that point. Thus $\mathcal{H}$ will contain a body saddle at an associated point in physical space, which will be a 1-saddle or 2-saddle in the *global* context of the entire mesh.

There are no saddles if $a_{\tilde{x}}a_{\tilde{y}}a_{\tilde{z}} > 0$, cases for which $a_{\tilde{x}}a_{\tilde{y}}a_{\tilde{z}} = 0$ the areas for $\nabla\widetilde{F} = \mathbf{0}$ are not points. Such cases need to be detected and treated separately, e.g. by a small perturbation of the scalar values at the vertices.

#### Classifying Body Saddles

In cell local context body saddles can be classified as either join or split depending on the nature of the extrema that they separate. Specifically, if a body saddle separates two maxima it is classified as join, whereas if a body saddle separates two minima it is classified as split. In this paragraph the discussion is restricted to the cell local context; a join in the cell local context will be a 2-saddle in the global context and a split in the cell local context will be 1-saddle in

the global context.

**Remark 3.4.1.** *Following [80], the topology of a set of isosurfaces* $S_k \equiv \{(\tilde{x}, \tilde{y}, \tilde{z}) : \widetilde{F}(\tilde{x}, \tilde{y}, \tilde{z}) = k\}$ *can be obtained by considering the discriminant*

$$\begin{aligned} DisC[k] = &(a(h-k))^2 + (bg)^2 + (ce)^2 + (df)^2 \\ &- 2abg(h-k) - 2ace(h-k) - 2adf(h-k) \\ &- 2bceg - 2bdfg - 2cdef + 4aefg + 4bcd(h-k). \end{aligned} \tag{3.13}$$

*Specifically, if* $a_{\tilde{x}} a_{\tilde{y}} a_{\tilde{z}} < 0$ *and* $a = 0$ *the isosurface set for* $DisC[k] < 0$ *consists of two separated segments which touch for* $DisC[k] = 0$*, occurring when* $k = \widetilde{F}(\tilde{x}_{s,0}, \tilde{y}_{s,0}, \tilde{z}_{s,0})$*, and merge into a single segment for* $DisC[k] > 0$*. However, if* $a_{\tilde{x}} a_{\tilde{y}} a_{\tilde{z}} < 0$ *and* $a \neq 0$ *the isosurface set for* $DisC[k] < 0$ *consists of three separated segments, two of which touch for* $DisC[k] = 0$*, occurring when* $k = \widetilde{F}(\tilde{x}_{s,+}, \tilde{y}_{s,+}, \tilde{z}_{s,+}), \widetilde{F}(\tilde{x}_{s,-}, \tilde{y}_{s,-}, \tilde{z}_{s,-})$*, and merge into a single segment for* $DisC[k] > 0$*, resulting in two separated segments. In summary, isosurfaces join as* $DisC[k]$ *changes from being negative to positive.*

**Lemma 3.4.1.** *Let* $k_s$ *be the function value* $F(\tilde{x}_s, \tilde{y}_s, \tilde{z}_s)$ *at a body saddle point* $s$ *located at* $(\tilde{x}_s, \tilde{y}_s, \tilde{z}_s)$*. The type of* $s$ *is determined by the sign of*

$$T(k_s) = \left.\frac{\partial DisC[k]}{\partial k}\right|_{k_s}. \tag{3.14}$$

*Specifically, if* $T(k_s) < 0$ *then* $s$ *is a join in the cell local context, whereas if* $T(k_s) > 0$ *then* $s$ *is a split in the cell local context.*

*Proof.* Consider $k_s$ to be the function value $F(\tilde{x}_s, \tilde{y}_s, \tilde{z}_s)$ at a body saddle point $s$ located at $(\tilde{x}_s, \tilde{y}_s, \tilde{z}_s)$. $DisC[k_s] = 0$ always, and if $T(k_s) < 0$ then via Remark 3.4.1 it is the case that decreasing $k_s$ will cause isosurfaces to join at $(\tilde{x}_s, \tilde{y}_s, \tilde{z}_s)$. Conversely, if $T(k_s) > 0$ then via Remark 3.4.1 it is the case that decreasing $k_s$ will cause isosurfaces to split at $(\tilde{x}_s, \tilde{y}_s, \tilde{z}_s)$. □

If $a_{\tilde{x}} a_{\tilde{y}} a_{\tilde{z}} < 0$ and $a = 0$ then

$$T(k_{s_0}) = -4bcd. \tag{3.15}$$

Hence via Lemma 3.4.1 $s_0$ will be a join in cell local context if $bcd > 0$ or a split in cell local context if $bcd < 0$. However, if $a_{\tilde{x}} a_{\tilde{y}} a_{\tilde{z}} < 0$ and $a \neq 0$ then

$$T(k_{s_-}) = -4\sqrt{-a_{\tilde{x}} a_{\tilde{y}} a_{\tilde{z}}} \tag{3.16}$$

and

$$T(k_{s_+}) = +4\sqrt{-a_{\tilde{x}} a_{\tilde{y}} a_{\tilde{z}}}. \tag{3.17}$$

Hence via Lemma 3.4.1 in cell local context $s_-$ will be a join saddle and $s_+$ will be a split saddle.

### 3.4.2 Face Saddles

Identifying and classifying critical points on faces requires special consideration since solution gradients on faces can be discontinuous across faces. Section 3.4.2 details how to identify and

classify critical points in the *local* context of each cell that shares a face. Subsequently, Section 3.4.2 details how to combine information about such local face saddles in order to determine if a true 1-saddle or 2-saddle exists.

#### Identifying and Classifying Element Local Face Saddles

Without loss of generality the analysis is restricted to face $\widetilde{\mathcal{T}}$ with $\tilde{z} = 0$ and vertices $V_0$, $V_1$, $V_2$ and $V_3$, on which $\widetilde{F}$ reduces to a bilinear function

$$\widetilde{F}_B(\tilde{x}, \tilde{y}) = \widetilde{F}(\tilde{x}, \tilde{y}, 0) = b\tilde{x}\tilde{y} + e\tilde{x} + f\tilde{y} + h. \tag{3.18}$$

Coordinates of saddles in $\widetilde{F}_B(\tilde{x}, \tilde{y})$ are given by the roots of

$$\nabla \widetilde{F}_B(\tilde{x}, \tilde{y}) = \begin{bmatrix} b\tilde{y} + e \\ b\tilde{x} + f \end{bmatrix}. \tag{3.19}$$

If and only if values of $\widetilde{F}_B$ at two opposite vertices of $\widetilde{\mathcal{T}}$ are maxima, with the other two values minima, will $\widetilde{\mathcal{T}}$ contain exactly one saddle $s_B$ at $(\tilde{x}, \tilde{y}) = (\tilde{x}_{s_B}, \tilde{y}_{s_B})$ where

$$\begin{aligned} \tilde{x}_{s_B} &= -\frac{f}{b}, \\ \tilde{y}_{s_B} &= -\frac{e}{b}, \end{aligned} \tag{3.20}$$

with the function value at the saddle given by

$$k_{s_B} = \widetilde{F}_B(\tilde{x}_{s_B}, \tilde{y}_{s_B}) = h - \frac{ef}{b}. \tag{3.21}$$

**Lemma 3.4.2.** *For the case of* $\nabla \widetilde{F}^{\tilde{z}}\Big|_{(\tilde{x}_{s_B}, \tilde{y}_{s_B}, 0)} \neq 0$*, a saddle* $s_B$ *of* $\widetilde{F}_B(\tilde{x}, \tilde{y})$ *in* $\widetilde{\mathcal{T}}$ *is also a critical point* $s_{FSL}$ *of* $\widetilde{F}$ *in the* local *context of* $\widetilde{\mathcal{H}}$*. Whether* $s_{FSL}$ *is a 1-saddle or a 2-saddle in the* local *context of* $\widetilde{\mathcal{H}}$ *is determined by the sign of* $\nabla \widetilde{F}^{\tilde{z}}\Big|_{(\tilde{x}_{s_B}, \tilde{y}_{s_B}, 0)}$*. Specifically, if* $\nabla \widetilde{F}^{\tilde{z}}\Big|_{(\tilde{x}_{s_B}, \tilde{y}_{s_B}, 0)} > 0$ *then* $s_{FSL}$ *is a 1-saddle, whereas if* $\nabla \widetilde{F}^{\tilde{z}}\Big|_{(\tilde{x}_{s_B}, \tilde{y}_{s_B}, 0)} < 0$ *then* $s_{FSL}$ *is a 2-saddle.*

*Proof.* To begin, consider face-parallel gradients. At $s_B$ it is the case that $\nabla \widetilde{F}^{\tilde{x}}\Big|_{(\tilde{x}_{s_B}, \tilde{y}_{s_B}, 0)} = \nabla \widetilde{F}^{\tilde{y}}\Big|_{(\tilde{x}_{s_B}, \tilde{y}_{s_B}, 0)} = 0$. On $\widetilde{\mathcal{T}}$ it is the case that $\nabla \widetilde{F}^{\tilde{x}}$ is constant in the $\tilde{x}$-direction and $\nabla \widetilde{F}^{\tilde{y}}$ is constant in the $\tilde{y}$-direction. Therefore $\nabla \widetilde{F}^{\tilde{x}} = 0$ and $\widetilde{F} = k_{s_B}$ along the line $\tilde{y} = \tilde{y}_{s_B}$ and $\nabla \widetilde{F}^{\tilde{y}} = 0$ and $\widetilde{F} = k_{s_B}$ along the line $\tilde{x} = \tilde{x}_{s_B}$, where $\tilde{y} = \tilde{y}_{s_B}$ and $\tilde{x} = \tilde{x}_{s_B}$ divide $\widetilde{\mathcal{T}}$ into four quadrants $q_1^-$, $q_2^-$, $q_1^+$, $q_2^+$ as per Figure 3.4, and where without loss of generality $q_1^-$ and $q_2^-$ are opposite quadrants that contain minima of $\widetilde{F}_B$ and $q_1^+$ and $q_2^+$ are opposite quadrants that contain maxima of $\widetilde{F}_B$. Also on $\widetilde{\mathcal{T}}$ it is the case that $\nabla \widetilde{F}^{\tilde{x}}$ varies linearly in the $\tilde{y}$-direction and $\nabla \widetilde{F}^{\tilde{y}}$ varies linearly in the in the $\tilde{x}$-direction. Therefore along $\tilde{y} = \tilde{y}_{s_B}$ and $\tilde{x} = \tilde{x}_{s_B}$ face parallel gradients always point into $q_1^+$ and $q_2^+$ as per Figure 3.4. Consequently, assuming that $\tilde{y} = \tilde{y}_{s_B}$ and $\tilde{x} = \tilde{x}_{s_B}$ do not belong to $q_1^-$, $q_2^-$, $q_1^+$, and $q_2^+$, then

$$\widetilde{F}\left(\tilde{x}, \tilde{y}, 0\right) > k_{s_B} \quad \forall \quad (\tilde{x}, \tilde{y}) \in q_1^+, q_2^+, \tag{3.22}$$

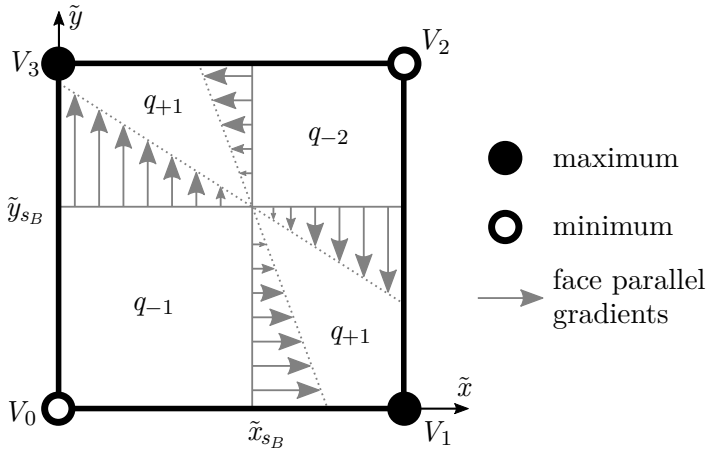



Figure 3.4: The asymptotes split $\mathcal{T}$ into four quadrants $q_1^-$, $q_2^-$, $q_1^+$, $q_2^+$, and where without loss of generality $q_1^-$ and $q_2^-$ are opposite quadrants that contain minima of $\widetilde{F}_B$ and $q_1^+$ and $q_2^+$ are opposite quadrants that contain maxima of $\widetilde{F}_B$. $\widetilde{F}_B > k_{s_B}$ throughout $q_1^-$ and $q_2^-$ and $\widetilde{F}_B < k_{s_B}$ throughout $q_1^+$ and $q_2^+$. Along the asymptotes the face parallel gradients point into $q_1^+$ and $q_2^+$.

$$\widetilde{F}\left(\tilde{x},\tilde{y},0\right) < k_{s_B} \quad \forall \quad (\tilde{x},\tilde{y}) \in q_1^-, q_2^-. \tag{3.23}$$

Taken together the above results imply that from every point in $q_1^+$ and $q_2^+$ their exists a monotone path to their respective maxima, and that from every point in $q_1^-$ and $q_2^-$ their exists a monotone path to their respective minima.

Now consider face-normal gradients. At $s_B$ it is generally the case that $\nabla\widetilde{F}^{\tilde{z}}\Big|_{(\tilde{x}_{s_B},\tilde{y}_{s_B},0)}$ can attain any value. Here it is considered that $\nabla\widetilde{F}^{\tilde{z}}\Big|_{(\tilde{x}_{s_B},\tilde{y}_{s_B},0)} \neq 0$. The special case where $\nabla\widetilde{F}^{\tilde{z}}\Big|_{(\tilde{x}_{s_B},\tilde{y}_{s_B},0)} = 0$ will be considered separately in Section 3.4.3. Furthermore, consider an axis-aligned cuboidal sub-region $N$ of $\widetilde{\mathcal{H}}$, centred on $s_B$ but restricted isotropically in $\tilde{x}$ and $\tilde{y}$ such that $\nabla\widetilde{F}^{\tilde{z}}\Big|_{(\tilde{x}_{s_B},\tilde{y}_{s_B},0)}$ does not change sign. Consider the case $\nabla\widetilde{F}^{\tilde{z}}\Big|_{(\tilde{x}_{s_B},\tilde{y}_{s_B},0)} > 0$. Now consider moving in the positive $\tilde{z}$-direction from each point of $q_1^- \cap N$ and $q_2^- \cap N$. Since $\nabla\widetilde{F}^{\tilde{z}}$ is constant in the $\tilde{z}$-direction $\widetilde{F} = k_{s_B}$ will occur exactly once. Furthermore consider moving in the positive $\tilde{z}$-direction from each point of $q_1^+ \cap N$ and $q_2^+ \cap N$. Since $\nabla\widetilde{F}^{\tilde{z}}$ is constant in the $\tilde{z}$-direction $\widetilde{F} = k_{s_B}$ will never occur. It therefore follows that there exists segments of an isosurface of $k_{s_B}$ within $\widetilde{\mathcal{H}}$ above $q_1^- \cap N$ and $q_1^- \cap N$, where the segments intersect $\widetilde{\mathcal{T}}$ along $\tilde{y} = \tilde{y}_{s_B}$ and $\tilde{x} = \tilde{x}_{s_B}$, and where the segments touch at $s_B$. Consequently $s_B$ of $\widetilde{F}_B(\tilde{x},\tilde{y})$ in $\widetilde{\mathcal{T}}$ is also a critical point $s_{FSL}$ of $\widetilde{F}$ in the *local* context of $\widetilde{\mathcal{H}}$. Moreover, from every point within $N$ bounded between the isosurface of $k_{s_B}$ and $q_1^-$ there will exist a monotone path to $q_1^- \cap N$, and hence via the discussions above a monotone path to the minima on $q_1^- \cap N$, and similarly from every point within $N$ bounded between the isosurface of $k_{s_B}$ and $q_2^-$ there will exist a monotone path to $q_2^- \cap N$, and hence via the discussions above a monotone path to the minima on $q_2^- \cap N$. Hence for the case $\nabla\widetilde{F}^{\tilde{z}}\Big|_{(\tilde{x}_{s_B},\tilde{y}_{s_B},0)} > 0$ $s_{FSL}$ separates two minima and is hence a 1-saddle. An example for an isosurface associated with a local face 1-saddle is provided in Figure 3.5. Repeating the above analysis for the case $\nabla\widetilde{F}^{\tilde{z}}\Big|_{(\tilde{x}_{s_B},\tilde{y}_{s_B},0)} < 0$ yields analogous results, except there exists segments of an isosurface of $k_{s_B}$ within $\widetilde{\mathcal{H}}$ above $q_1^+ \cap N$ and $q_2^+ \cap N$, and $s_{FSL}$ separates

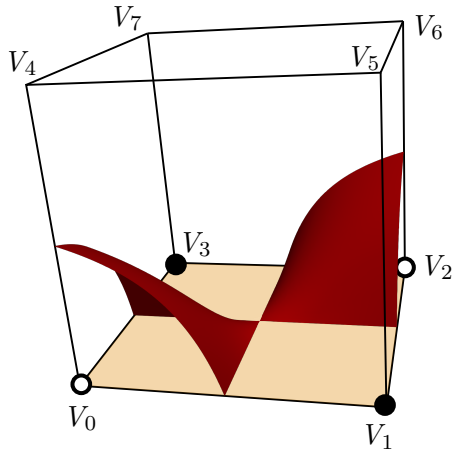



Figure 3.5: Isosurface for a trough face saddle on face $\widetilde{\mathcal{T}}$ separating the two minima marked with white vertices.

two maxima and is hence a 2-saddle in the context of $\widetilde{\mathcal{H}}$.

□

**Remark 3.4.2.** *It can be shown trivially that*

$$\begin{aligned}
&\nabla \widetilde{F}^{\tilde{z}}\Big|_{(\tilde{x}_{s_B},\tilde{y}_{s_B},0)} = \\
&\frac{-1}{(F_0 - F_1 + F_2 - F_3)^2}[(F_0 - F_1)[(F_0 - F_3)(F_2 - F_6) \\
&-(F_1 - F_2)(F_3 - F_7)] + (F_2 - F_3)[(F_0 - F_3)(F_1 - F_5) \\
&-(F_0 - F_4)(F_1 - F_2)]].
\end{aligned} \tag{3.24}$$

**Remark 3.4.3.** *The coefficients $b$, $e$, $f$, $h$, and the mapping of $\widetilde{\mathcal{T}}$ between physical and reference space only depend on information at the four vertices of $\widetilde{\mathcal{T}}$. Therefore, two cells that share $\widetilde{T}$ will always agree regarding the existence, function value, and location of any $s_B$ of $\widetilde{F}_B(\tilde{x},\tilde{y})$ and their associated $s_{FSL}$. However the type of $s_{FSL}$ in the* local *context of $\widetilde{\mathcal{H}}$ depends on information at the six vertices of $\widetilde{\mathcal{H}}$. Therefore, two cells that share $\widetilde{T}$ will not always agree regarding the type of their $s_{FSL}$.*

### Identifying and Classifying Face Saddles

**Lemma 3.4.3.** *If $\widetilde{\mathcal{T}}$ contains a saddle $s_B$, and the two cells that share $\widetilde{\mathcal{T}}$ agree regarding the type of the associated $s_{FSL}$, then the combined $s_{FSL}$ constitute a 'true' critical point of the agreed type in the* global *context of the entire mesh. However, if the two cells that share $\widetilde{\mathcal{T}}$ disagree regarding the type of the associated $s_{FSL}$ then the combined $s_{FSL}$ do not constitute a 'true' critical point in the* global *context of the entire mesh.*

*Proof.* Consider that $\widetilde{\mathcal{T}}$ containing a saddle $s_B$ is now shared between $\widetilde{\mathcal{H}}$ and an abutting reference cell. If $\widetilde{\mathcal{H}}$ and the abutting reference cell both have associated $s_{FSL}$ as 1-saddle as per Figure 3.6 (a), then within $N$ and an analogous sub-region of the abutting reference cell their respective segments of an isosurface of $k_{s_B}$ will cover $q_1^- \cap N$ and $q_2^- \cap N$. Consequently two isosurface segments are formed, that touch at $s_B$. Hence the combined $s_{FSL}$ saddles constitute

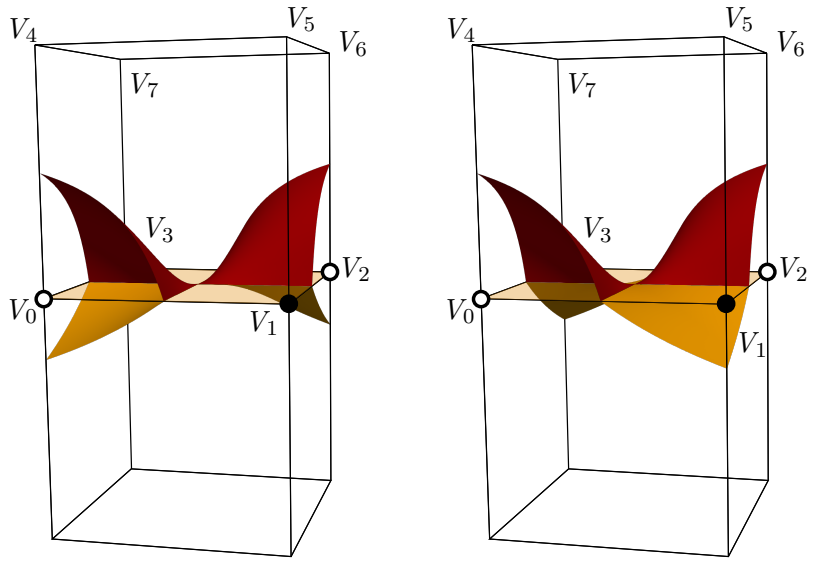



Figure 3.6: Isosurfaces of critical points on faces in two adjacent cells: In (a) $\widetilde{\mathcal{H}}$ and the abutting reference cell both have associated $s_{FSL}$ as 1-saddle, the combined $s_{FSL}$ constitute a 'true' 1-saddle in the *global* context of the entire mesh. In (b) $\widetilde{\mathcal{H}}$ has an associated $s_{FSL}$ as 1-saddle and the abutting reference cell has an associated $s_{FSL}$ as 2-saddle, the combined $s_{FSL}$ do not constitute a 'true' critical point in the *global* context of the entire mesh.

a 'true' critical point in the *global* context of the entire mesh. Moreover, there will exist a monotone path from any point bound between the isosurface segments above and below $q_1^- \cap N$ and the the minima on $q_1^- \cap N$, and from any point bound between the isosurface segments above and below $q_2^- \cap N$ and the minima on $q_2^- \cap N$. Hence the 'true' critical point in the *global* context of the entire mesh will be a 1-saddle. Repeating the above analysis for the case where $\widetilde{\mathcal{H}}$ and the abutting reference cell both have associated $s_{FSL}$ as 2-saddles leads to an analogous result, except the isosurface segments will cover $q_1^+ \cap N$ and $q_2^+ \cap N$, and the 'true' critical point in the *global* context of the entire mesh will be a 2-saddle.

Finally, if $\widetilde{\mathcal{H}}$ has an associated $s_{FSL}$ as 1-saddle and the abutting reference cell has an associated $s_{FSL}$ as 2-saddle as per Figure 3.6 (b), then within $N$ segments of an isosurface of $k_{s_B}$ will cover $q_1^- \cap N$ and $q_2^- \cap N$, but within an analogous sub-region of the abutting reference cell segments of an isosurface of $k_{s_B}$ will cover $q_1^+ \cap N$ and $q_2^+ \cap N$. Consequently a single isosurface segment is formed. Hence the combined $s_{FSL}$ saddles do not constitute a 'true' critical point in the *global* context of the entire mesh. Repeating the above analysis for the case where $\widetilde{\mathcal{H}}$ has an associated $s_{FSL}$ as 2-saddle and the abutting reference cell has an associated $s_{FSL}$ as 1-saddle leads to an analogous result. □

**Remark 3.4.4.** *If $\widetilde{T}$ contains a saddle $s_B$, and $\widetilde{T}$ is on a domain boundary, then the associated $s_{FSL}$ constitutes a 'true' critical point of the same type in the* global *context of the entire mesh.*

### 3.4.3 Body Saddles Coinciding with Faces

Once more the analysis is restricted to $\widetilde{\mathcal{T}}$ without loss of generality.

**Lemma 3.4.4.** *For the case of* $\nabla\widetilde{F}^{\tilde{z}}\Big|_{(\tilde{x}_{s_B},\tilde{y}_{s_B},0)} = 0$*, a saddle* $s_B$ *of* $\widetilde{F}_B(\tilde{x},\tilde{y})$ *in* $\widetilde{\mathcal{T}}$ *is also a saddle* $s_{VFS}$ *of* $\widetilde{F}$ *in the* local *context of* $\widetilde{\mathcal{H}}$*. Also* $s_{VFS}$ *is a body saddle coinciding with a face. Hence the type of* $s_{VFS}$ *in the* local *context of* $\widetilde{\mathcal{H}}$ *is determined according to Section 3.4.1. The topology of the isosurface of* $k_{s_B}$*,* $S_{VFS} \equiv \left\{(\tilde{x},\tilde{y},\tilde{z}) \in \widetilde{\mathcal{H}} : \widetilde{F}\,(\tilde{x},\tilde{y},\tilde{z}) = k_{s_B}\right\}$*, and its intersection with* $\widetilde{\mathcal{T}}$ *are identical to the isosurfaces of critical points on faces with* $\nabla\widetilde{F}^{\tilde{z}}\Big|_{(\tilde{x}_{s_B},\tilde{y}_{s_B},0)} \neq 0$*. Consequently its* global *status and type are determined according to Section 3.4.2.*

*Proof.* That for $\nabla\widetilde{F}^{\tilde{z}}\Big|_{(\tilde{x}_{s_B},\tilde{y}_{s_B},0)} = 0$ a body saddle coincides with $\widetilde{\mathcal{T}}$ is a consequence of both face parallel and face normal gradient components being zero. The type of $s_{VFS}$ in the *local* context of $\widetilde{\mathcal{H}}$ can be determined according to Section 3.4.1.

That $s_{VFS}$ is a saddle in the *local* context of $\widetilde{\mathcal{H}}$ follows from the topology of its associated isosurface. To analyse the topology first consider its behaviour on $\widetilde{\mathcal{T}}$. All analysis of function values and face parallel gradients in the proof of Lemma 3.4.2 also holds true in this case. $\nabla\widetilde{F}^{\tilde{z}}$ is described by a bilinear interpolation provided in Eq. (3.4). The isoline $h_{\nabla\widetilde{F}^{\tilde{z}}_B} \equiv \{(\tilde{x},\tilde{y},0) : \nabla\widetilde{F}^{\tilde{z}} = 0\}$ runs through $s_{VFS}$ and splits the area around $s_{VFS}$ into a region $A^+ \equiv \{(\tilde{x},\tilde{y},0) : \nabla\widetilde{F}^{\tilde{z}} > 0\}$ and a region $A^- \equiv \{(\tilde{x},\tilde{y},0) : \nabla\widetilde{F}^{\tilde{z}} < 0\}$.

Following the arguments in the proof of Lemma 3.4.2, the isosurface of $k_{s_B}$ covers the areas $A^- \cap q^+_{1,2}$ and $A^+ \cap q^-_{1,2}$. The structure of $A^- \cap q^+_{1,2}$ and $A^+ \cap q^-_{1,2}$ can be obtained, if the path of $h_{\nabla\widetilde{F}^{\tilde{z}}_B}$ is known. In the following will be show that if $s_{VFS}$ has peak type, $h_{\nabla\widetilde{F}^{\tilde{z}}_B}$ passes through and divides $q^-_{1,2}$, if $s_{VFS}$ has trough type, $h_{\nabla\widetilde{F}^{\tilde{z}}_B}$ passes through and divides $q^+_{1,2}$. Firstly, it shall be proven that in the case of a body 2-saddle coinciding with $\widetilde{\mathcal{T}}$ and for $a \neq 0$ $h_{\nabla\widetilde{F}^{\tilde{z}}_B}$ divides $q^-_{1,2}$. For $a \neq 0$, $h_{\nabla\widetilde{F}^{\tilde{z}}_B}$ is a hyperbola with axis aligned asymptotes. $h_{\nabla\widetilde{F}^{\tilde{z}}_B}$ divides $q^-_{1,2}$, if the centre of the hyperbola lies within $Q^+_{1,2}$, which is the extension of $q^+_{1,2}$ beyond the bounds of $\widetilde{\mathcal{T}}$. The coordinates of the centre of the hyperbola are:

$$\tilde{x}_g = -\frac{c}{a} \qquad \text{and} \qquad \tilde{y}_g = -\frac{d}{a}. \tag{3.25}$$

The centre of the hyperbola cannot trivially be assigned to any of the quadrants. This problem can be reformulated using relations Eq. (3.22) and Eq. (3.23), which state that $\widetilde{F}_B > k_{s_B}$ throughout $q^+_{1,2}$ and $\widetilde{F}_B < k_{s_B}$ throughout $q^-_{1,2}$. These relations can without further proof be extended to $Q^+_{1,2}$ and $Q^-_{1,2}$, which is the extension of $q^-_{1,2}$ beyond the bounds of $\widetilde{\mathcal{T}}$. In that way the the condition on the location of $(\tilde{x}_g,\tilde{y}_g)$ can be reformulated into a condition on its function value:

$$\widetilde{F}_B\,(\tilde{x}_g,\tilde{y}_g) > \widetilde{F}_B\,(\tilde{x}_{s,-},\tilde{y}_{s,-})\,. \tag{3.26}$$

Eq. (3.26) can be rearranged as follows:

$$\widetilde{F}_B\,(\tilde{x}_{s,-},\tilde{y}_{s,-}) - \widetilde{F}_B\,(\tilde{x}_g,\tilde{y}_g) < 0. \tag{3.27}$$

Using Eqs. (3.6, 3.12, 3.18, 3.25), the left hand side of Eq. (3.27) can be written as

$$\widetilde{F}_B\left(\tilde{x}_{s,-}, \tilde{y}_{s,-}\right) - \widetilde{F}_B\left(\tilde{x}_g, \tilde{y}_g\right) = -\frac{1}{a^2}\left[ba_{\tilde{z}} + 2\sqrt{-a_{\tilde{x}}a_{\tilde{y}}a_{\tilde{z}}}\right], \tag{3.28}$$

By substituting $\tilde{z}_{s_-} = 0$ in Eq. (3.10) and solving for $ba_{\tilde{z}}$, this term can be replaced in Eq. 3.28 with $-\sqrt{-a_{\tilde{x}}a_{\tilde{y}}a_{\tilde{z}}}$

$$\widetilde{F}_B\left(\tilde{x}_{s,-}, \tilde{y}_{s,-}\right) - \widetilde{F}_B\left(\tilde{x}_g, \tilde{y}_g\right) = -\frac{1}{a^2}\sqrt{-a_{\tilde{x}}a_{\tilde{y}}a_{\tilde{z}}} < 0. \tag{3.29}$$

For $a \neq 0$ it is therefore proven that $(\tilde{x}_g, \tilde{y}_g)$ lies within $Q^+_{1,2}$ and $h_{\nabla\widetilde{F}^{\tilde{z}}_B}$ passes through and divides $q^-_{1,2}$, if a body join saddle coincides with $\widetilde{\mathcal{T}}$.

In the case of a body split saddle coinciding with $\widetilde{\mathcal{T}}$ it can analogously be shown for $a \neq 0$ that

$$\widetilde{F}_B\left(\tilde{x}_{s,+}, \tilde{y}_{s,+}\right) - \widetilde{F}_B\left(\tilde{x}_g, \tilde{y}_g\right) = \frac{1}{a^2}\sqrt{-a_{\tilde{x}}a_{\tilde{y}}a_{\tilde{z}}} > 0, \tag{3.30}$$

proving that $(\tilde{x}_g, \tilde{y}_g)$ lies within $Q^-_{1,2}$ and $h_{\nabla\widetilde{F}^{\tilde{z}}_B}$ runs through and divides $q^+_{1,2}$.

Consider now the case of $a = 0$, in which $h_{\nabla\widetilde{F}^{\tilde{z}}_B}$ degenerates to the straight line

$$\tilde{y} = -\frac{d}{c}\tilde{x} - \frac{g}{c}. \tag{3.31}$$

If $h_{\nabla\widetilde{F}^{\tilde{z}}_B}$ only runs through $q^-_{1,2}$, function values of $\widetilde{F}_B$ along $h_{\nabla\widetilde{F}^{\tilde{z}}_B}$ must have an absolute maximum at $(\tilde{x}_{s,1}, \tilde{y}_{s,1})$. Analogously, if $h_{\nabla\widetilde{F}^{\tilde{z}}_B}$ only runs through $q^+_{1,2}$, function values of $\widetilde{F}_B$ along $h_{\nabla\widetilde{F}^{\tilde{z}}_B}$ must have an absolute minimum at $(\tilde{x}_{s,2}, \tilde{y}_{s,2})$. To verify this, first express the function values of $\widetilde{F}_B$ along $h_{\nabla\widetilde{F}^{\tilde{z}}_B}$ in terms of $\tilde{x}$ by inserting Eq. (3.31) into Eq. (3.18):

$$\begin{aligned}\widetilde{F}_B\left(\tilde{x}\right) &= \widetilde{F}_B\left(\tilde{x}, -\frac{d}{c}\tilde{x} - \frac{g}{c}\right) \\ &= -\frac{bd}{c}\tilde{x}^2 - \frac{1}{c}\left(bg + df - ec\right)\tilde{x} - \frac{gf}{c} + h.\end{aligned} \tag{3.32}$$

This function has an extremum at $\tilde{x}_{s_B}$ defined by Eq. (3.7) and therefore at the saddle. The second derivative of Eq. (3.32) with respect to $\tilde{x}$ is:

$$\frac{\partial^2}{\partial\tilde{x}^2}\widetilde{F}_B\left(\tilde{x}\right) = -2\frac{bd}{c}. \tag{3.33}$$

The sign of this expression is identical to the sign of the right hand side of Eq. (3.15):

$$\operatorname{sgn}\left(-2\frac{bd}{c}\right) = \operatorname{sgn}\left(-4bcd\right). \tag{3.34}$$

If the body saddle of the trilinear function is a join saddle, the sign in Eq. (3.34) is positive and Eq. (3.33) describes a parabola that opens down and has its vertex and therefore its absolute maximum at the saddle. $h_{\nabla\widetilde{F}^{\tilde{z}}_B}$ consequently passes through and divides $q^-_{1,2}$ if a body join saddle coincides with $\widetilde{\mathcal{T}}$.

In the case of a body split saddle coinciding with $\widetilde{\mathcal{T}}$ and $a = 0$ the sign in Eq. (3.34) is positive and Eq. (3.33) describes a parabola that opens up and has its vertex and therefore its absolute

minimum at the saddle. $h_{\nabla \widetilde{F}_B^{\tilde{z}}}$ consequently always passes through and divides $q_{1,2}^+$ if a body split saddle coincides with $\widetilde{\mathcal{T}}$.

Now consider the body of $\widetilde{\mathcal{H}}$. For the analysis of the shape of $S_{VFS}$ consider an axis-aligned cuboidal sub-region $M$ of $\widetilde{\mathcal{H}}$, centered on $s_B$ but restricted isotropically in $\tilde{x}$ and $\tilde{y}$. For $a = 0$ $M$ is restricted by $\widetilde{\mathcal{H}}$, for $a \neq 0$ $M$ is restricted such that it does not include $(\tilde{x}_g, \tilde{y}_g)$. If $(\tilde{x}_g, \tilde{y}_g)$ were to coincide with $h_{\nabla \widetilde{F}_B^{\tilde{z}}}$, it follows that $a_{\tilde{x}} = a_{\tilde{y}} = a_{\tilde{z}} = 0$ resulting in a degenerate case in which critical isosurfaces intersect along lines [80].

As mentioned above, the isosurface of $k_{s_B}$ covers $\left(A^+ \cap q_{1,2}^-\right) \cup \left(A^- \cap q_{1,2}^+\right)$. Due to the limitations of the path of $h_{\nabla \widetilde{F}_B^{\tilde{z}}}$, $\left(A^+ \cap q_{1,2}^-\right) \cup \left(A^- \cap q_{1,2}^+\right)$ always consists of three areas: one being a full quadrant and two being a fraction of a quadrant each. In Figure 3.7 these three areas are shaded. The full quadrant is covered by a connected component of the isosurface of $k_{s_B}$, which intersects $\widetilde{\mathcal{T}}$ along the asymptotes. Above the areas that make up a fraction of a quadrant, the isosurface of $k_{s_B}$ intersects $\widetilde{\mathcal{T}}$ along the asymptotes and approaches the extrusion of $h_{\nabla \widetilde{F}_B^{\tilde{z}}}$ in the $\tilde{z}$-direction asymptotically with increasing $\tilde{z}$. Above $s_B$ these two parts of the isosuface of $k_{s_B}$ are connected along a line in the $\tilde{z}$-direction, and therefore make up a single connected component. Both connected components touch at $s_B$, which therefore is a saddle of the body saddle's type in the local context, but its global status is decided according to Section 3.4.2. □

### 3.4.4 Summary

This Section describes how to identify and classify critical points in unstructured meshes of hexahedra which do not coincide with vertices of the computational mesh. These can appear in the body and on the faces of the cells. As per Section 3.4.1 each hexahedron is transformed to a cube in reference space. Within the reference cube the data is represented by the triliniear interpolant $\widetilde{F}$ provided in Eq. (3.2). Since the mapping from physical to reference space preserves orientation, a critical point of a certain type identified in reference space is a critical point of the same type in physical space.

Section 3.4.1 shows how to identify saddles in the body of cells and how to classify them by the type of extremum that they separate. Such body saddles are located at roots of $\nabla \widetilde{F}$. If all three

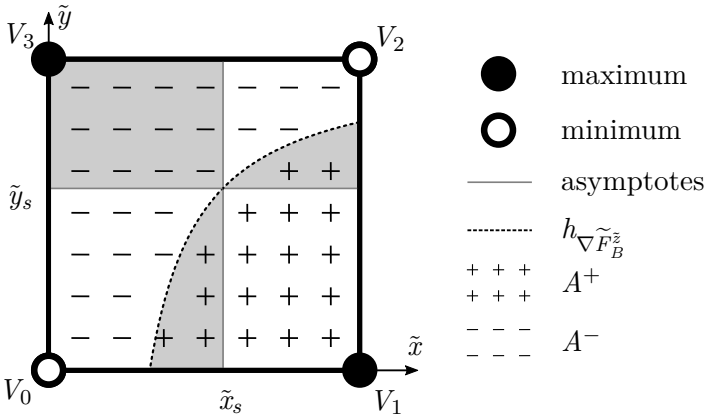



Figure 3.7: Example of a body join saddle on $\widetilde{\mathcal{T}}$. $h_{\nabla \widetilde{F}_B^{\tilde{z}}} = \{(\tilde{x}, \tilde{y}, 0) : \nabla \widetilde{F}_{tri}^{\tilde{z}} = 0\}$ divides $\widetilde{\mathcal{T}}$ into a section $A^+ = \{(\tilde{x}, \tilde{y}, 0) : \nabla \widetilde{F}_{tri}^{\tilde{z}} > 0\}$ and a section $A^- = \{(\tilde{x}, \tilde{y}, 0) : \nabla \widetilde{F}_{tri}^{\tilde{z}} < 0\}$. $S_{VFS}$ consists of two connected components covering the shaded areas.

coordinate components of a root of $\nabla\widetilde{F}$ have a value in the interval $]0,1[$ then a body saddle of that cell exists at that root, which is a critical point in the global context. In the case of $a \neq 0$ and $a_{\tilde{x}}a_{\tilde{y}}a_{\tilde{z}} < 0$, $\nabla\widetilde{F}$ has a root corresponding to a body split saddle at $(\tilde{x}_{s,+}, \tilde{y}_{s,+}, \tilde{z}_{s,+})$, as per Eq. (3.9) and a root corresponding to a body join saddle at $(\tilde{x}_{s,-}, \tilde{y}_{s,-}, \tilde{z}_{s,-})$, as per Eq. (3.10). In the case of $a = 0$ and $bcd \neq 0$, $\nabla\widetilde{F}$ has a root at $(\tilde{x}_s, \tilde{y}_s, \tilde{z}_s)$, as per Eq. (3.7). The latter root corresponds to a join saddle if $bcd > 0$ or it corresponds to a split saddle if $bcd < 0$.

On any face of the reference cell one coordinate is fixed and $\widetilde{F}$ reduces to a bilinear function. A face contains a saddle of the bilinear function, if two opposite vertices of the face are maxima and the other two are minima. Section 3.4.2 outlines how to identify these bilinear saddles. Such saddles are also critical points of $\widetilde{F}$ in the cell-local context. By analysing the topology of isosurfaces containing face saddles it was shown that the type of those points in the cell local context depends on the sign of $\nabla\widetilde{F}^{\mathbf{n}}\Big|_{FS}$, the face-normal inwards-pointing component of the gradient of $\widetilde{F}$ at the face saddle. A critical point on a face is a 1-saddle if $\nabla\widetilde{F}^{\mathbf{n}}\Big|_{FS} > 0$ or it is a 2-saddle if $\nabla\widetilde{F}^{\mathbf{n}}\Big|_{FS} < 0$. While saddles inside an cell's body are always critical in the global context, saddles on interior faces need to have the same type in both adjacent cells in order to be critical in the global context as was shown in Section 3.4.2.

If $\nabla\widetilde{F}^{\mathbf{n}}\Big|_{FS} = 0$ a body saddle coincides with the face. The topology of the isosurface containing a body saddle on a face, and its intersection with $\widetilde{\mathcal{T}}$ are identical to that of isosurfaces of face saddles. Consequently its *local* type can be obtained according to Section 3.4.1 and its *global* status and type are determined according to Section 3.4.2.

## 3.5 Comparison with Weber's Method

Our new method for identifying and classifying critical points on cell faces is based on analysing the face-normal gradient of $\widetilde{F}$ at the face, in each cell that shares the face. Weber et al. [105] proposed a similar method, based on the gradient of the asymptotic decider *AsD* in each cell that shares the face. However, as defined, the method of Weber et al. contains an error, and can lead to incorrect results. This Section defines the *AsD*, details the method of Weber et al. and explains how it can be corrected. Finally, it is shown that the corrected method of Weber et al. recovers the new method.

### 3.5.1 Asymptotic Decider

The *AsD* was developed by Nielson and Hamann [81] to resolve ambiguities in the marching cubes algorithm [67]. The marching cubes algorithm produces triangulated isosurfaces for an isovalue $k$ from a scalar data set discretised on voxels. An important step in the marching cubes algorithm involves drawing simplified isolines on voxel faces that intersect the isosurface. For such voxel faces where the isosurface only intersects two edges, a simplified isoline can be constructed by connecting the two edge-intersection points. However, if an isosurface intersects a voxel face that includes a saddle, the isosurface will intersect all four edges. In such a case it is not obvious which pairs of the four edge-intersection points need to be connected in order to reproduce topologically correct simplified isolines. The *AsD* returns the function value of the face's saddle without computing its coordinates. It is defined as the difference between

the product of function values at two (arbitrary) pairs of opposite face vertices divided by the difference of the sum of function values at the same two pairs of opposite face vertices. For example, on $\widetilde{\mathcal{T}}$ it is defined as:

$$AsD = \frac{F_0F_2 - F_1F_3}{F_0 + F_2 - F_1 - F_3}. \tag{3.35}$$

It can be shown that if $AsD > k$ the simplified isolines must separate two face minima, and if $AsD < k$ the simplified isolines must separate two face maxima; thus resolving the ambiguity as to which pairs of the four edge-intersection points need to be connected.

### 3.5.2 Weber's Method

Without loss of generality the discussion is restricted to face saddles on $\widetilde{\mathcal{T}}$ with face local maxima at $F_0$ and $F_2$. Weber et al. analyse the behaviour of $AsD$ on a *pseudo-face* $\widetilde{\mathcal{P}}$, parallel to $\widetilde{\mathcal{T}}$, as it moves a face-normal distance $\tilde{z}_{\widetilde{\mathcal{P}}}$ into the cell, where $P = \widetilde{\mathcal{T}}$ when $\tilde{z}_{\widetilde{\mathcal{P}}} = 0$, and where values at vertices of $\widetilde{\mathcal{P}}$ assume the local values of $\widetilde{F}$, thus changing linearly with $\tilde{z}_{\widetilde{\mathcal{P}}}$.

Defining:

$$AsD_{\tilde{z}_{\widetilde{\mathcal{P}}}} = \left.\frac{\partial AsD}{\partial \tilde{z}_{\widetilde{\mathcal{P}}}}\right|_{\tilde{z}_{\widetilde{\mathcal{P}}}=0} \tag{3.36}$$

Weber et al. show that if $AsD_{\tilde{z}_{\widetilde{\mathcal{P}}}} < 0$ the two maxima of $\widetilde{\mathcal{P}}$ will be separated by the intersection of $\widetilde{\mathcal{P}}$ with the isosurface associated with the face's saddle (this implicitly classifies it in local 3D context as 2-saddle), but if $AsD_{\tilde{z}_{\widetilde{\mathcal{P}}}} > 0$ the two minima of $\widetilde{\mathcal{P}}$ will be separated by the intersection of $\widetilde{\mathcal{P}}$ with the isosurface associated with the face's saddle (this implicitly classifies it in local 3D context as 1-saddle). Subsequently, it is reasoned that if $\text{sign}(AsD_{\tilde{z}_{\widetilde{\mathcal{P}}}})$ is the same in the context of each cell sharing the face, then the point is critical in the global context, but if $\text{sign}(AsD_{\tilde{z}_{\widetilde{\mathcal{P}}}})$ is different in the context of each cell sharing the face, then the point is regular in the global context.

However, Weber et al. make the erroneous assumption that $AsD$, and thus its numerator

$$\nu = F_0F_2 - F_1F_3, \tag{3.37}$$

are equal to zero on $\widetilde{\mathcal{T}}$, when in fact $AsD$ returns the function value at the face's saddle, which is not necessarily zero. Combined with the fact that the denominator of $AsD$

$$\xi = F_0 + F_2 - F_1 - F_3 \tag{3.38}$$

is always positive on $\widetilde{\mathcal{T}}$, since $F_0$ and $F_2$ are maxima, this leads to the erroneous conclusion that

$$\text{sign}\left(AsD_{\tilde{z}_{\widetilde{\mathcal{P}}}}\right) = \text{sign}\left(\nu_{\tilde{z}_{\widetilde{\mathcal{P}}}}\right), \tag{3.39}$$

and thus erroneous use of $\text{sign}\left(\nu_{\tilde{z}_{\widetilde{\mathcal{P}}}}\right)$ as a surrogate for $\text{sign}\left(AsD_{\tilde{z}_{\widetilde{\mathcal{P}}}}\right)$ to determine the existence of global critical points on faces as per the above logic. This can lead to those points being wrongly identified, missed, or correctly identified but mis-classified.

### 3.5.3 Correction of Weber's Method

The method of Weber et al. can be corrected by considering the sign of

$$AsD_{\tilde{z}_{\widetilde{\mathcal{P}}}} = \frac{\nu_{\tilde{z}_{\widetilde{\mathcal{P}}}}\xi - \nu\xi_{\tilde{z}_{\widetilde{\mathcal{P}}}}}{\xi^2}, \tag{3.40}$$

rather than using sign $\left(\nu_{\tilde{z}_{\widetilde{\mathcal{P}}}}\right)$ as a surrogate. The derivatives of $\nu$ and $\xi$ are:

$$\begin{aligned}\nu_{\tilde{z}_{\widetilde{\mathcal{P}}}} &= (F_0F_2 - F_1F_3)_{\tilde{z}_{\widetilde{\mathcal{P}}}} \\ &= \widetilde{F}_{\tilde{z}_{\widetilde{\mathcal{P}}}}\Big|_{V_0} F_2 + F_0\, \widetilde{F}_{\tilde{z}_{\widetilde{\mathcal{P}}}}\Big|_{V_2} - \widetilde{F}_{\tilde{z}_{\widetilde{\mathcal{P}}}}\Big|_{V_1} F_3 - F_1\, \widetilde{F}_{\tilde{z}_{\widetilde{\mathcal{P}}}}\Big|_{V_3},\end{aligned} \tag{3.41}$$

$$\begin{aligned}\xi_{\tilde{z}_{\widetilde{\mathcal{P}}}} &= (F_0 - F_1 + F_2 - F_3)_{\tilde{z}_{\widetilde{\mathcal{P}}}} \\ &= \widetilde{F}_{\tilde{z}_{\widetilde{\mathcal{P}}}}\Big|_{V_0} - \widetilde{F}_{\tilde{z}_{\widetilde{\mathcal{P}}}}\Big|_{V_1} + \widetilde{F}_{\tilde{z}_{\widetilde{\mathcal{P}}}}\Big|_{V_2} - \widetilde{F}_{\tilde{z}_{\widetilde{\mathcal{P}}}}\Big|_{V_3},\end{aligned} \tag{3.42}$$

with the derivatives at the vertices being:

$$\widetilde{F}_{\tilde{z}_{\widetilde{\mathcal{P}}}}\Big|_{V_0} = F_4 - F_0, \tag{3.43}$$

$$\widetilde{F}_{\tilde{z}_{\widetilde{\mathcal{P}}}}\Big|_{V_1} = F_5 - F_1, \tag{3.44}$$

$$\widetilde{F}_{\tilde{z}_{\widetilde{\mathcal{P}}}}\Big|_{V_2} = F_6 - F_2, \tag{3.45}$$

$$\widetilde{F}_{\tilde{z}_{\widetilde{\mathcal{P}}}}\Big|_{V_3} = F_7 - F_3. \tag{3.46}$$

By inserting Eq. (3.37), Eq. (3.38), and Eqs. (3.41 - 3.46) into Eq. (3.40), $AsD_{\tilde{z}_{\widetilde{\mathcal{P}}}}$ can be expressed in terms of scalar values at vertices:

$$\begin{aligned}AsD_{\tilde{z}_{\widetilde{\mathcal{P}}}} =& \frac{1}{(F_0 - F_1 + F_2 - F_3)^2}[(F_0F_2 - F_1F_3) \\ & (F_0 - F_1 + F_2 - F_3 - F_4 + F_5 - F_6 + F_7) \\ & +(F_0 - F_1 + F_2 - F_3)[-F_0(F_2 - F_6) \\ & +F_1(F_3 - F_7) - F_2(F_0 - F_4) + F_3(F_1 - F_5)]].\end{aligned} \tag{3.47}$$

The right hand side of Eq. (3.47) is identical to the right hand side of Eq. (3.24). Thus the corrected method of Weber et al. is identical to the new method.

## 3.6 Verification and Real World Examples

In order to verify that the new approach works as expected, a series of one-cell and two-cell test cases were employed. For all test cases the ground truth, in terms of local and global identification and classification of critical points on faces, was established by visual inspection. Results from both the new method and the method of Weber et al. [105] were then compared with this ground truth.

The new approach and the method of Weber were subsequently applied to identify and classify critical points on faces globally for two real world examples — specifically snapshots of $Q$-criterion fields from a Taylor-Green Vortex simulation and an SD7003 Aerofoil simulation.

### 3.6.1 Verification

#### Local Classification via One-Element Test Cases

One-cell test cases were used to verify local classification of critical points on faces. They consist of a cube with unit edge length as shown in Figure 3.8. Specifically, five individual test cases were constructed by selecting scalar values at $V_i$ to be $F_i$ from a given test case set in Table 3.1. For all test cases, there exists point $P$, which is critical in the local context, on the face $T_{0123}$ defined by vertices $V_0$, $V_1$, $V_2$, and $V_3$.

Table 3.2 provides classifications of $P$ for each test case. Specifically, a ground truth determined by visual inspection is provided, along with results obtained using the new method and the method of Weber et al.. Note that the new method provides correct classifications for all test cases, whereas the method of Weber et al. misclassifies $P$ in three of the five cases.

#### Identification and Classification in the Global Context via Two-Element Test Cases

Two-cell test cases were used to verify identification and classification of critical points of faces in the global context. They consisted of two cubes with unit edge length as shown in Figure 3.9. Specifically, 15 individual test cases were constructed by selecting scalar values at $V_i$ to be $F_i$ from a given test case set in Table 3.3. For all test cases, there exists a critical point $P$ on the face $T_{4567}$ defined by vertices $V_4$, $V_5$, $V_6$, and $V_7$.

Table 3.4 identifies whether $P$ is critical point in global context for each test case, and if it does provides a classification. Specifically, a ground truth determined by visual inspection is provided, along with results obtained using the new method and the method of Weber et al.. For each test case there are three possibilities:

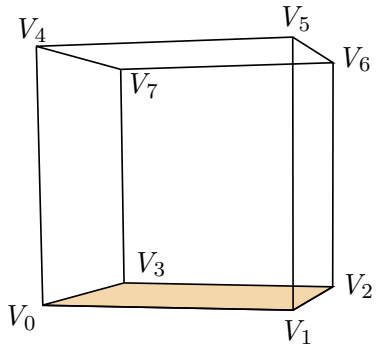



Figure 3.8: One-cell test case consisting of a cube with unit edge length. $P$ is on the highlighted face $T_{0123}$.

Table 3.1: Scalar values for one-cell test cases.

| test case | $F_0$ | $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ | $F_6$ | $F_7$ |
|---|---|---|---|---|---|---|---|---|
| I | 3 | 1 | 4 | 2 | 4 | 2 | 5 | 3 |
| II | 3 | 1 | 4 | 2 | 4 | 2 | 0 | 3 |
| III | 3 | 1 | 4 | 2 | 1 | 2 | 1 | 0 |
| IV | 3 | 1 | 4 | 2 | 2 | −3 | 2 | −2 |
| V | 3 | 1 | 4 | 2 | 5 | 0 | 2 | 1 |

Table 3.2: Classifications of $P$ for each one-cell test case. Specifically, a ground truth determined by visual inspection is provided, along with results obtained using the new method and the method of Weber et al..

| test case | classification | | | correctness | |
|---|---|---|---|---|---|
| | ground truth | new method | method of Weber | new method | Weber's method |
| I | 1-saddle | 1-saddle | 1-saddle | ✓ | ✓ |
| II | 1-saddle | 1-saddle | 2-saddle | ✓ | ✗ |
| III | 2-saddle | 2-saddle | 2-saddle | ✓ | ✓ |
| IV | 2-saddle | 2-saddle | 1-saddle | ✓ | ✗ |
| V | body 2-saddle | body 2-saddle | 1-saddle | ✓ | ✗ |

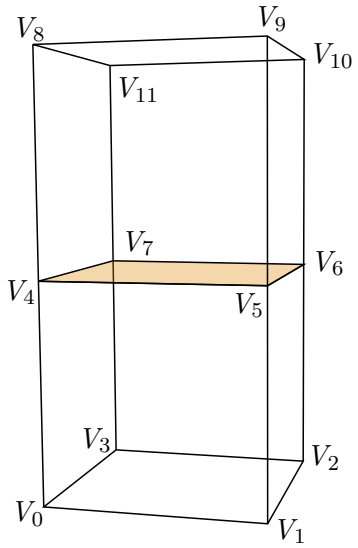



Figure 3.9: Two-cell test case consisting of two cubes with unit edge length. $P$ is on the highlighted face $T_{4567}$.

- RP: regular point in global context, e.g. test case V, see Figure 3.10 (a).
- 1-S: global 1-saddle, e.g. test case I, see Figure 3.10 (b).
- 2-S: global 2-saddle, e.g. test case XI, see Figure 3.10 (c).

The new method correctly identifies whether a global critical point exists on a face and classifies it correctly for all test cases, whereas the method of Weber et al. misses critical points, e.g. case XI, misidentifies regular points as critical points, e.g. case V, and misclassifies critical points, e.g. case VI.

Table 3.3: Scalar values for two-cell test cases.

| test case | $F_0$ | $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ | $F_6$ | $F_7$ | $F_8$ | $F_9$ | $F_{10}$ | $F_{11}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I | 4 | 2 | 5 | 3 | 3 | 1 | 4 | 2 | 4 | 2 | 5 | 3 |
| II | 4 | 2 | 0 | 3 | 3 | 1 | 4 | 2 | 4 | 2 | 5 | 3 |
| III | 1 | 2 | 1 | 0 | 3 | 1 | 4 | 2 | 4 | 2 | 5 | 3 |
| IV | 2 | −3 | 2 | −2 | 3 | 1 | 4 | 2 | 4 | 2 | 5 | 3 |
| V | 5 | 0 | 2 | 1 | 3 | 1 | 4 | 2 | 4 | 2 | 5 | 3 |
| VI | 4 | 2 | 0 | 3 | 3 | 1 | 4 | 2 | 4 | 2 | 0 | 3 |
| VII | 1 | 2 | 1 | 0 | 3 | 1 | 4 | 2 | 4 | 2 | 0 | 3 |
| VIII | 2 | −3 | 2 | −2 | 3 | 1 | 4 | 2 | 4 | 2 | 0 | 3 |
| IX | 5 | 0 | 2 | 1 | 3 | 1 | 4 | 2 | 4 | 2 | 0 | 3 |
| X | 1 | 2 | 1 | 0 | 3 | 1 | 4 | 2 | 1 | 2 | 1 | 0 |
| XI | 2 | −3 | 2 | −2 | 3 | 1 | 4 | 2 | 1 | 2 | 1 | 0 |
| XII | 5 | 0 | 2 | 1 | 3 | 1 | 4 | 2 | 1 | 2 | 1 | 0 |
| XIII | 2 | −3 | 2 | −2 | 3 | 1 | 4 | 2 | 2 | −3 | 2 | −2 |
| XIV | 5 | 0 | 2 | 1 | 3 | 1 | 4 | 2 | 2 | −3 | 2 | −2 |
| XV | 5 | 0 | 2 | 1 | 3 | 1 | 4 | 2 | 5 | 0 | 2 | 1 |

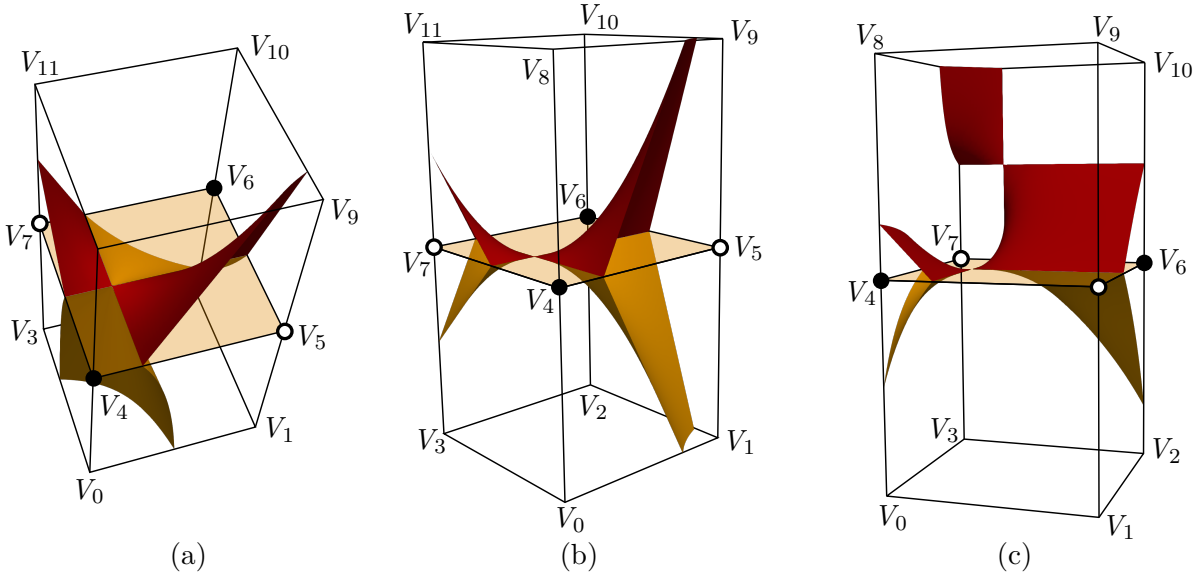



Figure 3.10: Isosurfaces through $P$ for two-cell test cases V (a), I (b), and XI (c). Minima and maxima on $T_{4567}$ are marked with white and black circles, respectively.

### 3.6.2 Real World Examples

The new approach has been applied to identify and classify critical points in the body and on faces of cells in snapshots of $Q$-criterion fields from a Taylor-Green Vortex simulation from Chapter 6 and a simulation of a flow over an SD7003 aerofoil from Chapter 5. Weber's method was applied to the same cases to identify and classify critical points on faces. Both simulations were previously undertaken by Vermeire et al. [103] using the the open-source PyFR [107]

Table 3.4: Identification of whether $P$ is critical in the global context for each two-cell test case, and if it is provision of a classification. Specifically, a ground truth determined by visual inspection is provided, along with results obtained using the new method and the method of Weber et al.. For each test case there are three possibilities: regular point RP, 1-saddle 1-S, and 2-saddle 2-S. $*$ indicates that the method of Weber et al. correctly identified RP at $P$ via incorrect classification of the local face saddles in both cells.

| test case | classification | | | correctness | |
|---|---|---|---|---|---|
| | ground truth | new method | Weber's method | new method | Weber's method |
| I | 1-S | 1-S | 1-S | ✓ | ✓ |
| II | 1-S | 1-S | RP | ✓ | ✗ |
| III | RP | RP | RP | ✓ | ✓ |
| IV | RP | RP | 1-S | ✓ | ✗ |
| V | RP | RP | 1-S | ✓ | ✗ |
| VI | 1-S | 1-S | 2-S | ✓ | ✗ |
| VII | RP | RP | 2-S | ✓ | ✗ |
| VIII | RP | RP | RP | ✓ | ✓ * |
| IX | RP | RP | RP | ✓ | ✓ * |
| X | 2-S | 2-S | 2-S | ✓ | ✓ |
| XI | 2-S | 2-S | RP | ✓ | ✗ |
| XII | 2-S | 2-S | RP | ✓ | ✗ |
| XIII | 2-S | 2-S | 1-S | ✓ | ✗ |
| XIV | 2-S | 2-S | 1-S | ✓ | ✗ |
| XV | 2-S | 2-S | 1-S | ✓ | ✗ |

solver. Isosurfaces of the $Q$-criterion field, which is derived analytically from co-projected velocity gradients, are considered to bound vortical structures in the flow. Correctly identifying and classifying saddles in a $Q$-criterion field is an important step in building a CT of the $Q$-criterion field, which can be used to facilitate the identification of vortical structures.

#### Taylor-Green Vortex

Details of the Taylor-Green vortex (TGV) simulation conducted by Vermeire et al. [103] from which the $Q$-criterion field was obtained are given in Table 3.5. Specifically, a snapshot of the $Q$-criterion field taken at 15 convective time units was analysed. A $Q = 0.01$ iso-surface of this snapshot is shown in Figure 3.11. The simulation was performed on a mesh consisting of $29^3$ high-order hexahedra, in which the solution was represented with an eighth-order polynomial. Hence, in order to enable analysis, the high-order solution was projected onto a mesh of $203^3$

Table 3.5: Details of the Taylor-Green Vortex simulation [103] from which the $Q$-criterion field was obtained for identifying critical off-vertex points.

| | |
|---|---|
| solver: | PyFR |
| solution basis order: | 8 |
| mesh: | $29^3$ hexahedra |
| Reynolds number: | 1 600 |
| Mach number: | 0.1 |

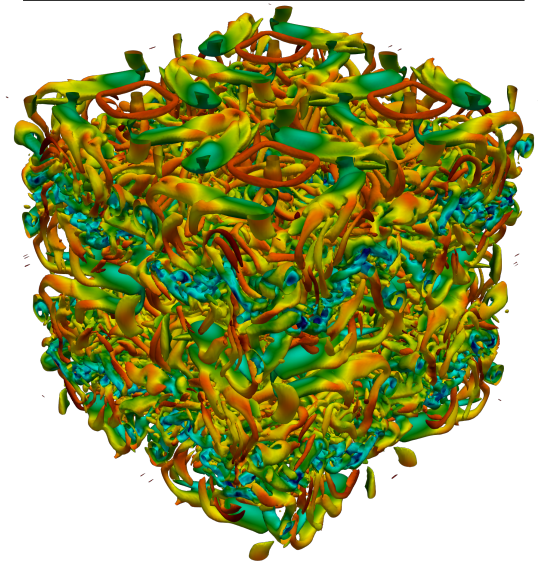


Figure 3.11: A $Q = 0.01$ iso-surface of the $Q$-criterion field for the Taylor-Green Vortex simulation at 15 convective time units, coloured by density.

Table 3.6: Number of global 1-saddles $N_{1-S}$ and global 2-saddles $N_{2-S}$ identified in the snapshot of the $Q$-criterion field from the Taylor-Green Vortex simulation using both the new method and the method of Weber. Also presented are the number of critical points missed by the method of Weber $N_{MS}$, the number of critical points identified but misclassified by the method of Weber $N_{WT}$, and the number of number of regular points wrongly classified as critical points by the method of Weber $N_{RP}$. Values for $N_{MS}$, $N_{WT}$ and $N_{RP}$ are based on the assumption that the new method is correct.

| | $N_{1-S}$ | $N_{2-S}$ | $N_{MS}$ | $N_{WT}$ | $N_{RP}$ |
|---|---|---|---|---|---|
| new method | 182 695 | 134 638 | - | - | - |
| method of Weber | 207 689 | 140 112 | 111 486 | 7 203 | 141 954 |

hexahedra.

31 721 body 1-saddles and 75 910 body 2-saddles were found in the $Q$-criterion field. Table 3.6 provides the number of critical points on faces identified and classified by each method. Specifically, the number of 1-saddles on faces $N_{1-S}$ and 2-saddles $N_{2-S}$ are provided. Also presented are the number of critical points missed by the method of Weber $N_{MS}$, the number of critical points identified but misclassified by the method of Weber $N_{WT}$, and the number of number of regular points wrongly identified as critical points by the method of Weber $N_{RP}$. Values for $N_{MS}$, $N_{WT}$ and $N_{RP}$ are based on the assumption that the new method is correct.

### SD7003 Aerofoil

The simulation of the flow around an SD7003 aerofoil conducted by Vermeire et al. [103] from which the $Q$-criterion field was obtained is presented in detail in Section 5.1. A $Q = 1.0$ iso-surface of this snapshot is shown in Figure 3.12. The simulation was performed on a mesh consisting of 137 916 high-order hexahedra. Hence, in order to enable analysis, the high-order solution was projected onto a mesh of 8 826 624 hexahedra.

73 253 body 1-saddles and 93 970 body 2-saddles were found in the $Q$-criterion field. Table 3.7 provide the number of critical points identified and classified by each method. Specifically, the number of global 1-saddles $N_{1-S}$ and global 2-saddles $N_{2-S}$ are provided. Also presented are the number of critical points missed by the method of Weber $N_{MS}$, the number of critical points identified but misclassified by the method of Weber $N_{WT}$, and the number of number of regular points wrongly identified as critical points by the method of Weber $N_{RP}$. Values for $N_{MS}$, $N_{WT}$ and $N_{RP}$ are based on the assumption that the new method is correct.

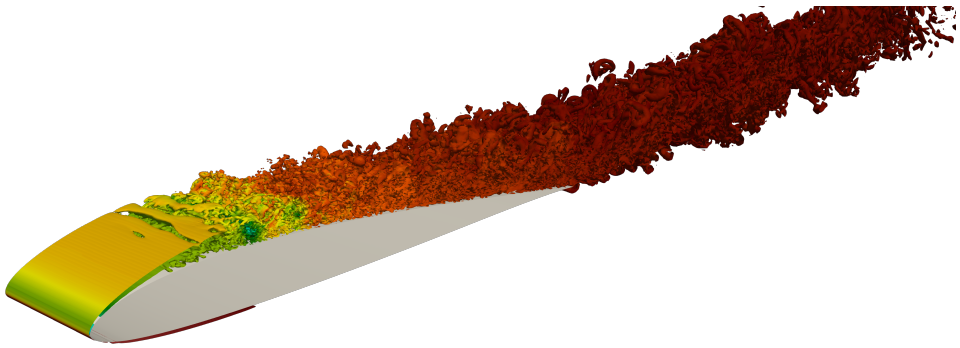


Figure 3.12: A $Q = 1.0$ iso-surface of the $Q$-criterion field for the SD7003 Aerofoil simulation at 40 convective time units, coloured according to solution density.

Table 3.7: Number of global 1-saddles $N_{1-S}$ and global 2-saddles $N_{2-S}$ identified in the snapshot of the $Q$-criterion field from the SD7003 Aerofoil simulation using both the new method and the method of Weber. Also presented are the number of critical points missed by the method of Weber $N_{MS}$, the number of critical points identified but misclassified by the method of Weber $N_{WT}$, and the number of number of regular points wrongly classified as critical points by the method of Weber $N_{RP}$. Values for $N_{MS}$, $N_{WT}$ and $N_{RP}$ are based on the assumption that the new method is correct.

| | $N_{1-S}$ | $N_{2-S}$ | $N_{MS}$ | $N_{WT}$ | $N_{RP}$ |
|---|---|---|---|---|---|
| new method | 291 001 | 281 835 | - | - | - |
| method of Weber | 520 625 | 202 678 | 162 958 | 55 717 | 313 425 |

# 4 Tracer - an In-Situ Framework for Identifying Flow Features in CFD Data

Tracer is a software to identify features in flow fields of high-fidelity unsteady turbulent flow simulations conducted on unstructured meshes. To do so Tracer utilises the JT of a scalar field $f$ to carry out a topology based domain segmentation as outlined in Chapter 2. Section 4.1 provides an overview of Tracer detailing its capabilities, the in-situ version which runs concurrently with the CFD simulation, and the standalone version, which reads a file from disk. Section 4.2 presents the design and architecture of preprocessing, JT construction and analysis components. Section 4.3 explains how the correctness of the software was tested and Section 4.4 provides a performance analysis.

## 4.1 Overview

The main objective of Tracer is to extract features in scalar data sets discretised on unstructured edge graphs. If, for example, the underlying scalar $f$ is the $Q$-criterion, the extracted features would be considered to represent vortices. Each feature is defined by the area encapsulated in a contour associated to an individual isovalue of $f$. Those isovalues are determined based on the topology of the surrounding $f$. As explained in Section 2.1 Tracer achieves this by constructing the JT from the edge graph, through which the relevance field can be computed. The individual $f$ isovalues are obtained by defining a global relevance threshold. Via the JT any scalar quantity can be efficiently integrated over extracted features and geometric measures like centre of mass or bounding boxes can be obtained. Additionally the structure of the JT provides insight in the topological organisation of $f$ and as well as of individual features.

The following paragraphs detail which results Tracer can provide, the work flow that leads to these results for the in-situ version, which analyses data on the fly as it is produced by another software and the standalone version, which reads a vtu file from disk and the use of third-party software in Tracer.

### 4.1.1 Capabilities and Output

Tracer can produce

- **a csv file containing geometric measures and topological HS ratios of features.** This file contains geometric measures and HS ratios of extracted features. Geometric measures include volume, bounding box, centre of features and in which region of the domain the feature is located. The HS ratios describe the topological structure of the features as explained in Section 2.2.1. In the in-situ configuration Tracer will produce a csv file at every time step it is called.

- **a vtu file including the relevance field and vortex IDs.**
  The vtu file produced by Tracer contains $f$ and the relevance field. Other variables like the velocity vector field can also be included provided they are present in the input data. In the in-situ configuration Tracer will produce a vtu file at every time step it is called.

- **png files depicting renderings of isosufaces of the relevance field.**
  Renderings can be produced in-transit via Alpine/Ascent [64], further information is provided in Section 4.2.6. In the in-situ configuration Tracer will produce a png for every camera position at every time step it is called.

- **histograms of HS numbers of the full domain.**
  Histograms of the HS numbers described in Section 2.2.1 are provided for each set of hypAs of the same HS order individually. In the in-situ configuration Tracer accumulates the HS numbers over all time steps and outputs a single set of histograms after the simulation has finished.

- **histograms of HS ratios of individual features.**
  Histograms of the distribution of HS ratios are provided individually for each set of features with same topological complexity. In case there are user-defined subregions of the domain Tracer will provide a histogram for the the vortices of each subregion as well as of the full domain. In the in-situ configuration Tracer accumulates the HS ratios over all time steps and outputs a single set of histograms for each HS ratio after the simulation has finished.

### 4.1.2 Work Flow of Tracer's In-Situ Version and of Tracer's Standalone Version

In its in-situ version Tracer runs concurrently with PyFR [107] analysing the data as it is produced on the GPU, in its standalone version Tracer analyses a vtu file read from disk. In its current version Tracer is restiricted to analyse flow fields locally on a single GPU. A flow diagram of the in-situ version and of the standalone version can be found in Figures 4.1 and 4.2, respectively.

PyFR provides the in-situ version of Tracer with a vtu like representation of the flow field discretised on an unstructured linearly subdivided representation of the computational mesh. During the initialisation Tracer resolves ambiguities in the subdivided mesh, constructs the edge graph and stores it on the GPU. Details of the linearly subdivided mesh and the preprocessing are provided in Section 4.2.1. At every time step at which Tracer is called, PyFR provides the solution of the flow field in the form of conservative variables on the GPU. Tracer converts these into primitive variables and computes $Q$ at every vertex, which serves as underlying scalar field $f$ for the topological analysis.

The standalone version of Tracer reads in a vtu file from disk, which is has to to fulfil the requirements on the input graph and has to have $f$ present. More information in the input requirements of the standalone version are provided in Section 4.2.2.

After the initial stage the workflow of both versions is identical: on the GPU Tracer constructs the JT, computes the relevance field and extracts features. The topology of $f$ is analysed and depending on the configurations set by the user the appropriate selection of files from Section 4.1.1 is written to disk.

### 4.1.3 Third-Party Software and Libraries

As PyFR is highly performant on Nvidia GPUs, Tracer also targets them and is hence written in Cuda [79]. A significant amount of computational costs during the JT construction with PPP is accounted for sorting. Efficient sorting algorithms are included in Cuda's template library Thrust [9]. Amongst other, Tracer also makes use of Thrust's segmented prefix sums and stream compaction. Thrust functions have a pretty wide flexibility as they can be adapted via custom operators.

Reading and writing vtu files is done via the VTK [91] library. In-transit visualisations are rendered using Alpine/Ascent [64].

## 4.2 Implementation

The following paragraphs present the implementation of the methods in Tracer. The in-situ version and the standalone version of Tracer mainly differ in the pre-processing step, which is described in Section 4.2.1 and Section 4.2.2 respectively. The JT construction via the adapted PPP presented in Sections 2.3 and 2.4 are explained in Section 4.2.3. Sections 4.2.4 and onwards deal with the analysis of the JT, specifically on extracting features and the topological organisation of the underlying scalar field $f$.

### 4.2.1 Preprocessing and Computation of the Scalar Field $f$ in the In-Situ Version

As outlined in Figure 4.1, PyFR provides Tracer with the solution discretised on a subdivided mesh. Subdivision is a method by which each high-order element is subdivided into an equal number of smaller cells in order to capture the underlying high-order polynomial data. That way the properties of an unstructured mesh are maintained via a simple and computationally cheap method. Jallepalli et al. [52] have shown that subdivision successfully captures coarse features and achieves good results in separating features. The order $p$ of the polynomial, which represents the solution within the high-order elements, determines the lower bound for level of subdivision to $p + 1$. The authors furthermore demonstrated that a strong increase in the level of subdivision can lower the quality of the result as artefacts are arising from discontinuities across the interfaces of high-order elements. In the applications in this thesis hence a level of subdivision was chosen, which is equal to or slightly larger than the lower boundary of $p + 1$.

The solution is provided in the form of conservative variables on the device. The mesh format is identical the one used for vtu files by VTK [91]. The mesh information, that is vertex coordinates, cell types and the connectivity array which links cell local vertex IDs to global vertex IDs, is provided on the host side. On the faces and edges of high-order elements, the point cloud of the super-sampled mesh includes coinciding vertices, which do not agree in the solution they hold. This is due to the discontinuous nature of the flux reconstruction method employed in PyFR.

During initialisation Tracer extracts the edge graph of the provided mesh. First the mesh is converted into a *continuous* form by resolving ambiguities caused by coinciding vertices: in the connectivity array each vertex that belongs to a set of coinciding vertices is replaced with the vertex of this set which has the lowest ID. This step is equivalent to applying the

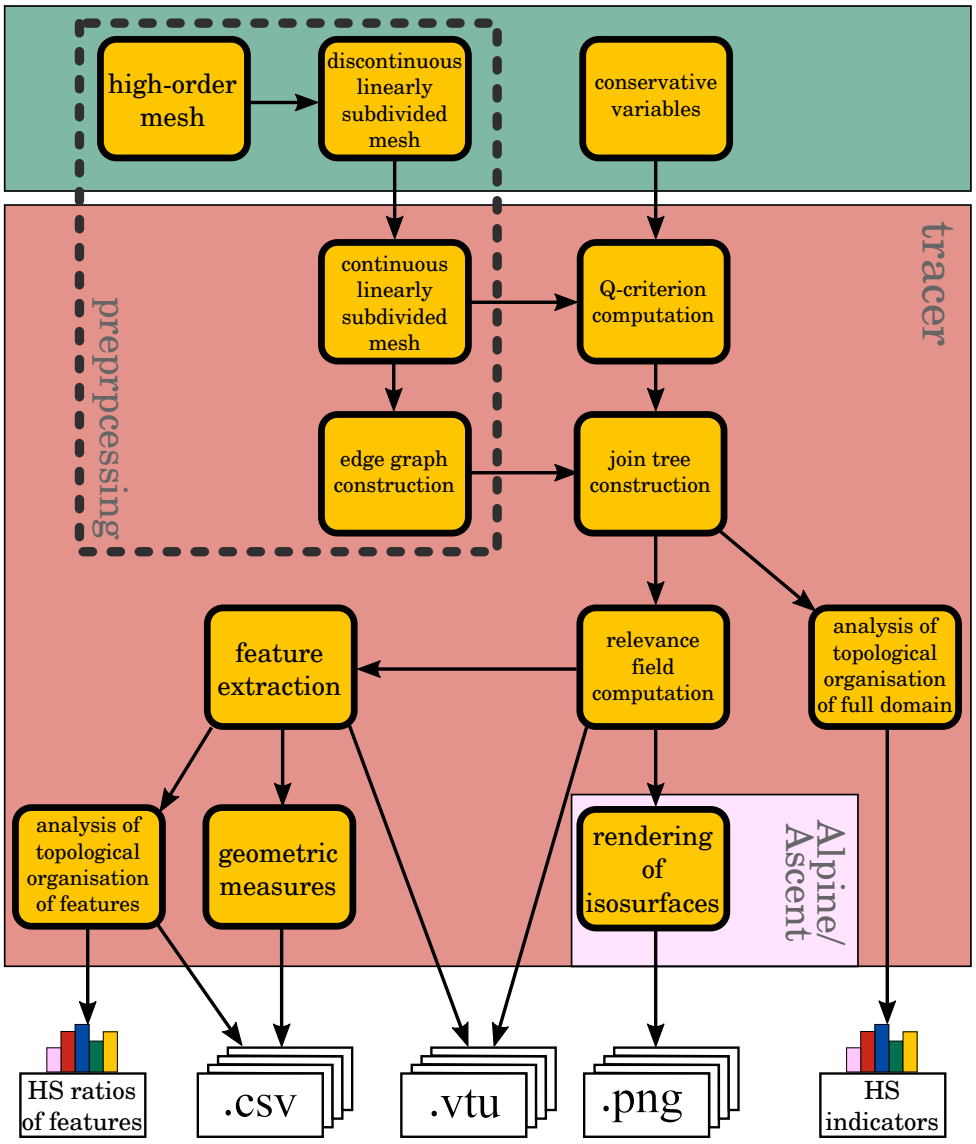



Figure 4.1: Flow diagram of the in-situ configuration of Tracer running alongside PyFR analysing the $Q$-criterion field to extract vortices. The colour of the background indicates the software which is carrying out the steps. Preprocessing steps are inside the dashed box, the remainder steps are executed every time Tracer is called by PyFR. The csv, vtu and png files are written to disk after every run of Tracer, the two histograms are accumulated over all time steps and written to disk in the end. More information on the output files is provided in Section 4.1.1.

`vtkCleanUnstructuredGrid` filter of VTK, which is also employed by the `CleanToGrid` filter in ParaView [5]. Periodic boundary conditions are handled analogously, vertices on both sides of periodic boundaries are paired up and in the connectivity array vertices of the *right* side of the periodic boundary are replaced with their partner from the *left* side of the periodic boundary. In case there are more than one periodic boundaries, they are dealt with one after another to correctly pair up and replace vertices that are located on intersections of multiple periodic boundaries.

Resolving ambiguities artificially amplifies gradients of the solution and derived quantities like the $Q$ criterion. Those amplifications scale with the magnitude of the discontinuities between

high-order elements, which in turn depend on the resolution of the flow. Since Tracer is applied to high-fidelity CFD data the discontinuities are expected to be small enough to not introduce a significant amount of artefacts. Additionally such artefacts will have sizes in the order of a few vertices, which can easily be filtered using the method from Section 4.2.5 once the domain segmentation is completed.

The edge graph is constructed from the continuous mesh and stored in the form of two arrays: edgeStart holding the smaller vertex ID and edgeEnd holding the higher vertex ID. The edges are sorted by (edgeStart, edgeEnd). This graph is copied to the device alongside vertex coordinates and the continuous connectivity array, the latter two of which are required for gradient computation.

After the edge graph is built Tracer allocates the required memory. The size of the memory is estimated based on the size of the input graph and a user defined estimate of which fraction of $f$ will be above a cutoff threshold $f_{min}$. If during the computation the allocated memory will turn out as not being big enough, Tracer will throw an error providing information how much memory was missing at which point.

When Tracer is applied to the solution data provided on the GPU by PyFR it converts the conservative into primitive variables so the $Q$-criterion field, which is used as $f$, can be computed from the velocity vector field. Within each cell of the continuous mesh the velocity gradient is computed at its vertices. Vertices which belong to more than one cell have their gradient averaged over all associated cells. Subsequently, $Q$ is computed at each vertex according to [44]. These steps are equivalent to applying VTK's `vtkGradientFilter`, which is also employed in ParaView's `GradientOfUnstructuredDataSet` filter. With the edge graph and $f$ being available Tracer has all information required to build the JT.

### 4.2.2 Preprocessing in the Standalone Version

The standalone version of Tracer reads the data from a `vtu` file on the disk as depicted in Figure 4.2. The mesh in this file is required to be continuous and the $f$ must be present. If for example the $Q$-criterion field of a `vtu` file exported from PyFR shall be processed one option would be preparing the data with ParaView's `CleanToGrid` and `GradientOfUnstructuredDataSet` filters. If present, Tracer will resolve any periodic boundary conditions before compiling the edge graph and allocating memory as outlined in Section 4.2.1. The graph consisting of vertices and edges is uploaded to the device to build the JT of $f$.

### 4.2.3 Implementation of Parallel Peak Pruning

Tracer relies on the JT construction algorithm from PPP [18, 23, 24] which was described in Section 2.3.2. To enable processing CFD data discretised on unstructured meshes PPP was implemented with the adaptions described in Section 2.4. The following paragraphs provide a detailed description of the implementation.

#### Monotone Path Construction

The aim of the monotone path construction is to assign a peak to each vertex in the relevant regions. A relevant region was defined in Section 2.4 as a connected region with $f > f_{min}$. In

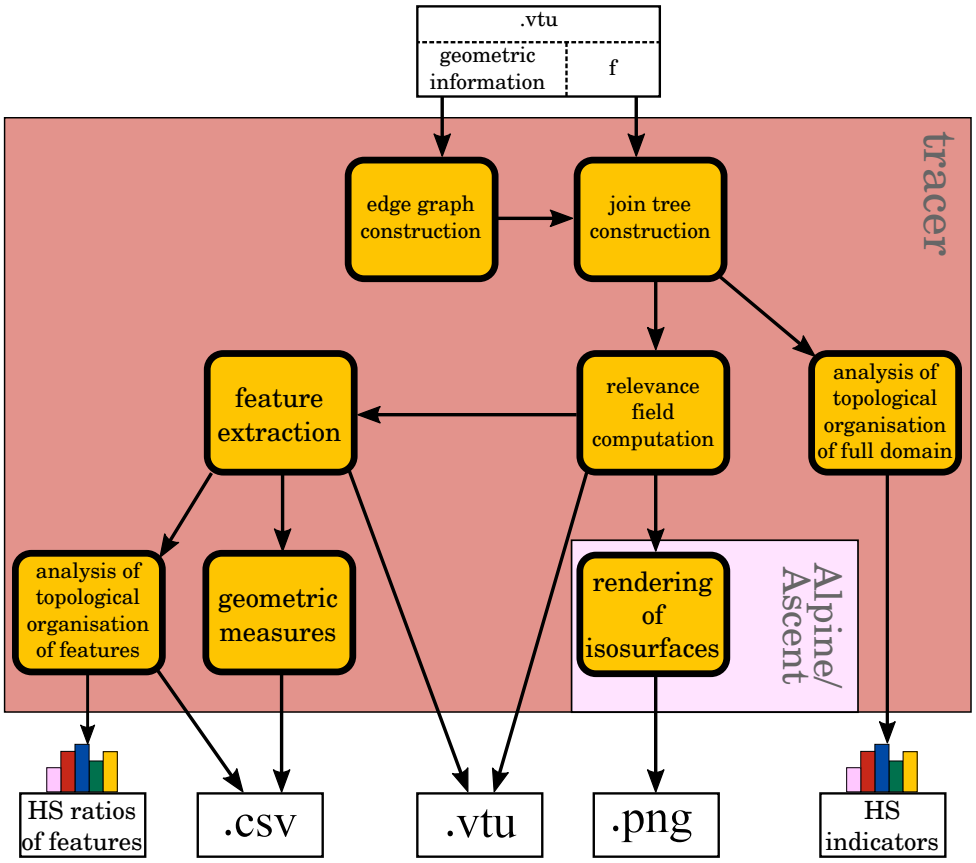



Figure 4.2: Flow diagram of the standalone configuration of Tracer, in which Tracer reads $f$ discretised on an unstructured mesh from disk. The colour of the background indicates which software is carrying out the steps. After analysing the scalar field Tracer writes the results in the form of the files from Section 4.1.1 to disk.

the context of the present edge graph the union of all relevant regions is comprised of the union of edges which start and end at vertices $i$ that have $f_i > f_{min}$ and their vertices. The ascending path $ap_i$ of each vertex $i$ is initialised with the respective vertex itself. Subsequently each edge is tested whether it is *relevant* and flagged accordingly. In case the edge is *relevant*, the scalar values $f_{start}$ and $f_{end}$ of both of its ends are compared. If $f_{start} \leq f_{end}$ the edge will be flagged as ascending and the ascending path of the edge start will be set to the edge end: $ap_{start} = end$. Otherwise the edge will be flagged as descending and the ascending path if the edge end will be set to the edge start: $ap_{end} = start$. Since $start < end$ is always fulfilled, this assignment of ascending and descending edges inherently follows the sorting according to (scalar value, vertex ID). Pseudocode for MonotonePathConstruction() is provided in Algorithm 4.1. A vertex generally belongs to more than one edge, thus assigning ascending paths in parallel will in practice lead to race conditions and non-deterministic results. However, it does not matter which of all possible ascending paths from that vertex are chosen. Hence a correct result is provided if writing $ap_i$ is guaranteed to be atomic. The only vertices the ascending path of which still point to themselves are true peaks and vertices with $f_i \leq f_{min}$. To improve efficiency Tracer constructs an active graph by storing all relevant edges in *activeEdges*, the active graph on which Tracer will henceforth operate.

**Algorithm 4.1:** MonotonePathConstruction() - Identifying relevant edges and ascending paths.

```
1  ap = {0, 1, ..., n_edges − 1}
2  for j ∈ {0, 1, ..., n_edges − 1} do in parallel
3      if f_start_j > f_min and f_end_j > f_min then
4          edge j is relevant
5          if f_start_j ≤ f_end_j then
6              edge j is ascending
7              ap_start_j = end_j
8          else
9              edge j is descending
10             ap_end_j = start_j
11     else
12         edge j is not relevant
13 end
14 store IDs of all relevant edges in activeEdges
15 return
```

#### Identifying Peaks in each Connected *Interesting* Region

Tracer tags at least one peak within each relevant region as being a supN. As pointed out in Section 2.4, adding this step to the original version of the algorithm is necessary since there can be connected relevant regions that are topologically identical to a sphere and include only a single peak. Actual critical points are identified in the trunk construction, during which peaks are paired with saddles. If, however, there is no saddle to be paired with such peaks would not be processed. The obvious way to identify all peaks in the relevant regions would be filtering all vertices $i$ which fulfill both $f_i > f_{min}$ and $ap_i = i$. But this would also identify isolated vertices with $f_i > f_{min}$ which are neither start nor end of a relevant edge and therefore are not included in the relevant region. In order to eliminate such isolated points Tracer adds as a third requirement that such peaks need to be pointed to by another vertex. This step can also eliminate peaks in interesting regions with more than one peak, but such peaks will be identified and paired with a saddle during first iteration of trunk construction. The details of this procedure are provided in Algorithm 4.2.

**Algorithm 4.2:** FindPeaksForAllSubregions() - Identifying at least one peak within each connected relevant region.

```
for i ∈ {0, 1, ..., n_verts − 1} do in parallel
    if ap_i ≠ i then
        vertex ap_i is relevant
end
for i ∈ {0, 1, ..., n_verts − 1} do in parallel
    if vertex i is relevant and ap_i = i then
        vertex i is a super node
    else
        vertex i is not a super node
end
return
```

### Critical Topology Graph Construction

The aim of this part of the program is to further compress the active graph to a critical topology graph consisting of vertices which are saddle candidates and edges which have saddle candidates as their lower end. The pseudocode for FindSaddleCandidateEdges() is provided in Algorithm 4.3.

The first step in identifying saddle candidates is assigning a peak to each vertex in the relevant regions which can be reached from the vertex via a monotonously ascending path. After performing a pointer jumping [51] on the *ap* array, *ap* holds the associated peak for all vertices. The pointer jumping is done on the full array rather than only the active ones, since otherwise those indices would need to be updated first, resulting in overall more work and memory requirements.

The saddle candidates (see Section 2.3) are identified in a second step: Consider a set of edges $g$ which is comprised of all edges having vertex $i$ as lower end. Tracer classifies $i$ as saddle candidate if any of the upper ends of $g$ is associated to a different peak than $ap_i$. The active graph is compressed by dismissing all edges the lower end of which is not a saddle candidate. Each edge of the active graph gets associated to the peak of its upper end, which is stored in the array *ep*.

**Algorithm 4.3:** FindSaddleCandidateEdges() - Identifying all saddle candidates and all edges having a saddle candidate as lower end.

```
1  PointerJumping(ap)                          // [51]
2  isSaddleCandidate = {false, false, ...}
3  for j ∈ activeEdges do in parallel
4  |  l = lower end of j
5  |  u = upper end of j
6  |  if ap_l ≠ ap_u then
7  |  |  isSaddleCandidate_l = true
8  end
9  isSaddleCandidateEdge = {false, false, ...}
10 for j ∈ activeEdges do in parallel
11 |  l = lower end of j
12 |  if isSaddleCandidate_l = true then
13 |  |  isSaddleCandidateEdge_j = true
14 end
15 return
```

### Trunk Construction Iteration

The objective of this part is to pair all peaks with their governing saddles, the highest join saddle from which a peak can be reached via a monotonous path. The peaks are pruned to their governing saddle by removing them from the active graph *activeEdges* and updating the list of saddle candidates and their edges accordingly. These steps are repeated until there are no saddles left. Each peak-saddle pair will be represented in the hypS as hyper node hypN and its governing saddle hypG respectively. A hypA starts at a hypN and terminates at its hypG. The counter of the iteration in which a pair is processed is provides the HS order of the pair's hypA. Additionally this identifies all supNs of relevant regions with two or more peaks. The pseudocode for FindHyperArcs() this iteration is provided in Algorithm 4.4.

**Algorithm 4.4:** FindHyperArcs() - Finding hypAs by iteratively pairing up peaks with their governing saddles and removing them from the active graph.

```
1  HS order = 0
2  while length(activeEdges) > 0 do
3  |  PairPeaksWithGoverningSaddles()        // Alg. 4.5
4  |  PrunePeaksFromActiveGraph()            // Alg. 4.6
5  |  RedirectAssociatePeaks()               // Alg. 4.7
6  |  CompressActiveGraph()                  // Alg. 4.8
7  |  HS order = HS order + 1
8  end
9  return
```

To identify governing saddles *activeEdges* are sorted by (associated peak ID, lower end scalar value, lower end ID). If after sorting the right neighbour of an edge has an associated peak different to its own, then the lower end of that edge is the governing saddle *govSad* of its associated peak. The peak and the saddle are tagged as supNs, the peak is additionally tagged as hypN. The number of identified pairs $n_{p,r}$ for this iteration $r$ is recorded cumulatively in *ovh*:

$$ovh_0 = 0, \tag{4.1}$$

$$ovh_{r+1} = ovh_r + n_{p,r}, \tag{4.2}$$

so the $r$th element of *ovh* holds the index of the first hypA of HS order $r$.

All edges that have the governing saddle as lower end and are associated to its peak are tagged to be removed from the active graph during the pruning step. Finally the associated peak of the governing saddle is redirected to point to itself. The pseudocode of this method is provided in Algorithm 4.5.

**Algorithm 4.5:** PairPeaksWithGoverningSaddles($r$) - Pairing Peaks of the active graph with their governing saddles.

```
1  sort activeEdges by (associated peak ID, lower end scalar value, lower end ID)
2  for j ∈ {0, 1, ..., length(activeEdges)} do in parallel
3  |   n_{p,r} = 0
4  |   if ep[activeEdges[j]] ≠ ep[activeEdges[j + 1]] then
5  |   |   govSad[ep[activeEdges[j]]] = lower end of activeEdges[j]
6  |   |   govSad[ep[activeEdges[j]]] is supN
7  |   |   ep[activeEdges[j]] is supN and hypN
8  |   |   n_{p,r} = n_{p,r} + 1
9  end
10 ovh_r = ovh_{r−1} + n_{p,r}
11 for j ∈ activeEdges do in parallel
12 |   l = lower end of j
13 |   if l is supN and ep[j] = ap[l] then
14 |   |   flag j to be removed from activeEdges
15 end
16 for i ∈ governing saddles do in parallel
17 |   ap[i] = i
18 end
19 return
```

After being paired with their governing saddles the peaks need to be pruned from the active graph. Firstly all edges which were flagged during the pairing step are removed from *activeEdges*. Secondly all edges left in *activeEdges* have the associated peaks for both of their ends redirected to their governing saddle. Ambiguities for vertices that belong to multiple edges won't affect the final JT. Vertices in pruned regions, that are upper ends of edges and which lie above the governing saddle of their associated (pruned) peak, are excepted from redirecting

their associated peaks. Keeping the associated peak of these vertices unchanged is important for assigning them to the correct hypA. The pseudocode of this method is provided in Algorithm 4.6.

**Algorithm 4.6:** PrunePeaksFromActiveGraph() - Prune peaks from the active graph.

```
remove all flagged edges from activeEdges
for j ∈ activeEdges do in parallel
    l = lower end of j
    u = upper end of j
    ap[l] = govSad[ap[l]]
    if u is located below govSad[ap[u]] then
        ap[l] = govSad[ap[l]]
end
return
```

After their peaks have been pruned, governing saddles are not necessarily peaks of the new scalar field but can also be regular nodes. If they turn into regular nodes they must not have themselves as associated peak. Since all edges from governing saddles to pruned peaks have been removed from the active graph, redirecting the associated peaks of the lower end of the edges to the associated peaks of the edge is ensuring that regular nodes do not have themselves as associated peaks. In a following step Tracer redirects the associated peaks of super nodes which have turned regular due to one of their associated peaks having been pruned in a previous iteration. Exceptions are again nodes that are within a region that has already been pruned. Once all associated peaks are redirected *ap* can be updated via another round of pointer jumping so that all vertices belonging to a relevant region which has not been pruned yet are associated to a peak of the active graph. The pseudocode of this method is provided in Algorithm 4.7. The associated peaks of vertices that belong to a region which has been pruned already are not affected and will stay associated to their hypN. All vertices in the relevant regions are guaranteed to be associated to a hypN of their subtree. This results in all vertices including the supNs to be associated to the hypN with the highest HS order which can be reached via a montone path. The associated peaks of the edges are set to the associated peaks of their upper ends.

**Algorithm 4.7:** RedirectAssociatePeaks() - Redirect associate peaks of vertices and edges.

```
for j ∈ activeEdges do in parallel
    l = lower end of j
    ap[l] = ep[j]
end
PointerJumping(ap)                          // [51]
for j ∈ activeEdges do in parallel
    u = upper end of j
    ep[j] = ap[u]
end
return
```

The active graph is compressed by removing all edges from *activeEdges* the lower ends of which are not saddle candidates of the pruned scalar field. Tracer identifies saddle candidates by checking if the lower end of an edge has a different peak assigned than the edge itself. If the lower end used to be a saddle but is now a regular point, the node will be added to the the array of former critical nodes so its associated peak keeps getting updated even though it no longer belongs to the active graph. Finally all edges the lower end of which is not a saddle candidate will be removed from *activeEdges*. The pseudocode of this method is provided in Algorithm 4.8. If the size of *activeEdges* is zero, all peaks have been pruned and the algorithm continues to build the hypS. Otherwise another iteration of trunk construction follows.

**Algorithm 4.8:** CompressActiveGraph() - Remove all edges from the active graph which do not have a saddle candidate as their lower end.

```
1  for j ∈ activeEdges do in parallel
2  |  l = lower end of j
3  |  if ap[l] ≠ ep[j] then
4  |  |  l is a saddle candidate
5  end
6  for j ∈ activeEdges do in parallel
7  |  l = lower end of j
8  |  if l is not a saddle candidate then
9  |  |  remove j from activeEdges
10 end
11 return
```

### Building the Hyper Structure

To build the hyper structure first a preliminary unaugmented JT is constructed. Traces uses this JT as a vehicle to extract hypN-hypG pairs, identify root hypNs, assign their HS order and add them to the hypN-hypG pairs from the trunk contstruction. The hypS is constructed from the hypN-hypG pairs using the information about their HS orders. The following paragraphs describe those steps in detail.

All supNs have been identified and were associated to a hyper parent. The hyper parent of a vertex is the hypN with the highest HS order which can be reached from the vertex via a montone path. Recall that hypNs are a subset of supNs and have themselves as hyper parent, hypNs from which root hypAs start have themselves assigned as hypG. The unaugmented JT is built in the form of a preliminary structure consisting of all supNs by sorting the supNs by (vertex ID of hyper parent [ascending], scalar value [descending], vertex ID [descending]). The resulting JT structure is a list of supNs in which the supG to each element is stored inherently as its right neighbour, unless the right neighbour is associated to a different hyper parent, in which case supG is hypG of the supN's hyper parent.

The preliminary structure bears the full information of the JT but is rather cumbersome to augment and analyse. Tracer therefore constructs the hypS and the supS described in Section 2.3.1 before augmenting the tree to obtain augS. The hypS built in this part of the code is

made up of hypAs in the form of two arrays `hypS.upper` and `hypS.lower` storing their upper ends hypN and their lower ends hypG respectively.

Tracer starts the hypS construction by extracting the $n_{sup}$ supN-supG pairs from the preliminary structure and stores them in `supS.upper` and `supS.lower`. If the supG of a supN is identical to its hyper parent, that hyper parent has not been paired with a saddle and is consequently a root hypN and flagged as such. The number of root hypNs provides the number of connected relevant regions $n_{rr}$. The root hypNs are added to `hypS.upper` and `hypS.lower` is filled with governing saddle from trunk construction. The elements of `hypS.lower` which correspond to root hypAs will store the same vertex ID as the corresponding elements in `hypS.upper`.

The root hypNs have to have their correct HS order being assigned to them via the method presented in Section 2.4, the implementation of which is shown in Algorithm 4.9: An array `hypS.HSorder` containing the HS order for each hypA is set up, root hypAs are initialised with 0 and a *counter* array `hypS.cnt` is allocated. In the following steps will be carried out iterating $r$ from 0 to $r_{max} - 2$:
The elements in `hypS.cnt` are initialised with 0. All hypNs of order $r$ atomically increase the `hypS.cnt` of their hypG by one, if their hypG is the upper end of a root hypA. All root hypAs that have a counter $\geq 2$ get assigned a HS order of $r + 1$. That method guarantees to assign the correct HS order also to hypNs at which more than two hypAs terminate.

**Algorithm 4.9:** AssigningHSOrdersToRoots() - Assigning HS-orders to root hypAs.

```
1  for r ∈ {0, 1, ..., r_max − 1} do
2  |  for i ∈ {ovh_r, ovh_r + 1, ..., ovh_{r+1} − 1} do in parallel
3  |  |  hypS.HSorder[i] = r
4  |  end
5  end
6  for i ∈ {0, 1, ..., n_rr − 1} do in parallel
7  |  hypS.HSorder[ovh_{r_max} + i] = 0
8  end
9  for r ∈ {0, 1, ..., r_max − 1} do
10 |  for i ∈ {0, 1, ..., n_rr − 1} do in parallel
11 |  |  hypS.cnt[ovh_{r_max} + i] = 0
12 |  end
13 |  for i ∈ {ovh_r, ovh_r + 1, ..., ovh_{r+1} − 1} do in parallel
14 |  |  if hypS.lower[i] is root then
15 |  |  |  hypS.cntr[hypS.lower[i]] += 1
16 |  end
17 |  for i ∈ {0, 1, ..., n_rr − 1} do in parallel
18 |  |  if hypS.cntr[ovh_{r_max} + i] ≥ 2 then
19 |  |  |  hypS.HSorder[ovh_{r_max} + i] = r + 1
20 |  end
21 end
22 return
```

After all iterations have finished the hypS arrays are sorted by their HS order and *ovh* is updated accordingly. To be able to link into the hypS the vertex IDs in *ap* are replaced with their associated hypS ID, which will be stored in the physical structure as `phyS.hypID`. Note that at this stage only supNs are guaranteed to have their correct `phyS.hypID` assigned to them. Regular nodes are assigned to a hypA of their subtree with $r = 0$.

#### Building the Super Structure

In the supS all supAs belonging to the same hypA shall occur subsequently in descending order of $f$ of their supN. Hence the supAs need to be arranged the order of (hypA ID [ascending], scalar value [descending], vertexID [descending]). Since the supAs are already stored in order of their supN's (vertex ID of their associated peak, scalar value [descending], vertexID [descending]) only a stable sort by hypA ID needs to be performed. To be able to link from phyS into supS, supNs have their supA ID assigned in `phyS.supID`.

#### Augmenting the Tree with Regular Nodes

The augS shall store all vertices of the relevant regions sorted by (hypA ID, supA ID, scalar value, vortex ID). Each node points to its right neighbour if the vertex belongs to the same hypA, otherwise it points to the hypG of its hypA.

First all $n_{relVerts}$ vertices in the relevant regions are identified as belonging to a relevant edge. During the trunk construction iterations only associated peaks of super nodes were updated, hence elements of regular nodes in `phyS.hypID` still have a hypA with $r = 0$ of their associated sub tree assigned to them, not necessarily their actual hyper parent. The correct hypA of a vertex $i$ is identified via a linear search by following the hypS down starting from `phyS.hypID[`$i$`]` until `hypS.lower[phyS.hypID[`$i$`]]` lies below $i$. The pseudocode for this procedure is provided in Algorithm 4.10. Since in every update $r$ of the hypA increases by one, the iteration finishes latest after $r_{max}$ steps.

**Algorithm 4.10:** AssignHyperArcsToVertices() - Assigning hypAs to all *relevant* vertices.

```
1 for i ∈ {0, 1, ..., n_relVerts − 1} do in parallel
2 |   while hypS.lower[phyS.hypID[i]] is above i do
3 |   |   phyS.hypID[i] = phyS.hypID[hypS.lower[phyS.hypID[i]]]
4 |   end
5 end
6 return
```

The associated supA for a vertex $i$ can be found analogously by iterating through all supAs of the vertice's associated hypA until `supS.lower[phyS.supID[`$i$`]]` lies below $i$. However, the number of supAs in a hypA can be very large and is generally a lot bigger than $r_{max}$ which is why Tracer is following [18] and finds the correct supA via a binary search.

To obtain the augmented structure the vertices have to be brought in the correct order. Since the supAs are already sorted by (hypA ID, scalar value of upper end [descending], vertexID of upper end [descending]) the vertices are sorted by (supA IDs [ascending], scalar values [descend-

ing], vertexIDs [descending]). Since `phyS.hypID` and `phyS.supID` have also been sorted with the vertices, they are now part of the augS and renamend to `augS.hypID` and `augS.supID`.

### Link Four Structures

`hypS.upper`, `hypS.lower`, `supS.upper` and `supS.lower` are populated with vertex IDs, hence linking into phyS. To avoid detours over the phyS Tracer will link the structures according to Figure 4.3: the `.upper` arrays of hypS and supS shall be populated with the IDs of the nodes in the augS, and the `.lower` arrays of hypS and supS will point into the supS. The augS already holds the ID of hypA ans supA of each node as well as their vertex ID which is the link into the phyS. The arrays `.lower` are redirected from pointing into the phyS to pointing into the supS via a sort according to Algorithm 4.11 before augS is finalised.

**Algorithm 4.11:** RedirectLowerEndsIntoSuperStructure() - Redirecting elements in `hypS.lower` and `supS.lower` from pointing into phyS to point into supS. This is done after all regular nodes have been assigned to a supA and before they are sorted to make up augS.

**for** $i \in \{0, 1, ..., n_{hyp}\}$ **do in parallel**
  `hypS.lower[`$i$`]` = `phyS.supID[hypS.lower[`$i$`]]`
**end**
**for** $i \in \{0, 1, ..., n_{sup}\}$ **do in parallel**
  `supS.lower[`$i$`]` = `phyS.supID[supS.lower[`$i$`]]`
**end**
**return**

After augS is finished Tracer is taking advantage from hypNs being the first node of their hypA to appear in augS. Since in augS the nodes are additionally sorted by hypAs, `hypS.upper` can be redirected into augS by filling `hypS.upper` with the indices $i$ of augS for which which `augS.hypID[`$i$`]` $\neq$ `augS.hypID[`$i-1$`]`. Analogously also `supS.upper` are redirected to point into augS.

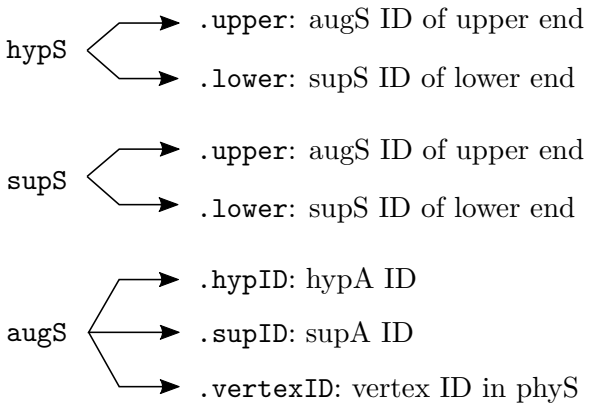



Figure 4.3: Links between the tree structures.

### 4.2.4 Obtaining Integral Quantities of Subtrees

With the JT structures integrals and maxima of a quantity $q$ over each node's subtree can be computed efficiently following the method by Carr et al. [21, 22] described in Section 2.2.2. The procedure is comparable to a prefix sum of a 1D array. The array `augS.q` is filled with the value of $q$ at each node in the augS. Tracer performs the operation *op*, which can either be plus min, or max, over $q$ along a hypA $a$ of order $r$ and passes the result to the lower end of $a$. The lower end of $a$ belongs to a hypA $b$ of order $o > r$, hence all hypAs of the same order can be processed in parallel. Since in augS all nodes of a hypA occur in consecutive order and are sorted by height, `augS.q` can be *summed up* along hyper arcs using the segmented prefix sum of the Thrust library [9]. Passing the result to the lower end has to be atomic, as more than one hypAs of $r$ can terminate at the same node. Tracer *sums up* $q$ via the functions AccumulateSubTreeQuantities(`augS.q`, *op*) and PassToHyperNeighbour(`augS.q`, *op*, $r$) which can be found in Algorithms 4.12 and 4.13.

Some applications like the computation of the relevance criterion or of HS ratios do not require summing over the augS but can be done on the supS. The procedure is the same, $q$ is summed up along the supN's of all hypAs of order $r$ and the result is passed to the hypA's lower end until a root is reached.

**Algorithm 4.12:** AccumulateSubTreeQuantities(`augS.q`, *op*) - Performing an in-place prefix sum on `augS.q` by accumulating a quantity $q$ from the leaves down to the root. As a result each node in augS holds the accumulated quantity of its subtree. *op*, the atomic operator used to *sum* the quantity, can either be plus, min, or max.

```
1 for r ∈ {0, 1, ..., r_max − 1} do
2 |   thrust::inclusive_scan_by_key(augS.hypIDs+ova_r, augS.hypIDs+ova_{r+1},
  |                                   augS.q,            augS.q,
  |                                   thrust::equal_to(), op) // [9]
3 |   PassToHyperNeighbour(augS.q, op, r)   // Alg. 4.13
4 end
5 return
```

**Algorithm 4.13:** PassToHyperNeighbour(`augS.q`, *op*, $r$) - Performs operator *op* on the lowest node of each hypA of order $r$ which is not a root and hypA's lower end (which belongs to a hypA of order $r + 1$).

```
1 for i ∈ {ova_r, ova_r + 1, ..., ova_{r+1} − 1} do in parallel
2 |   if augS.hypIDs[i] ≠ augS.hypIDs[i + 1] and i is not a root then
3 |   |   recipient = supS.upper[hypS.lower[augS.hypIDs[i]]]
4 |   |   augS.q[recipient] = op(augS.q[recipient], augS.q[i])
5 end
6 return
```

### 4.2.5 Feature Extraction via a Relevance Threshold

Tracer interprets features as connected point clouds of $\mathbb{G}$ for which the relevance criterion $R$ [73] is above a user-defined threshold $R_{cut}$. To extract such features, Tracer firstly computes the $R$ field, secondly finds the roots of each structure, thirdly filters artefacts and fourthly assigns a label to each feature and all of its vertices. The following paragraphs describe those steps in detail. For further information on feature extraction via $R$ see Section 2.1.

#### Computing the Relevance Field

To compute $R$ Tracer identifies the highest scalar value that can be reached from a supA via the method presented in Section 4.2.4. Since all nodes on a supA share their highest peak of their subtree, the relevance can be computed for every node in the augS according to Eq. (2.1). To improve smoothness of isocontours in case they get visualised, Tracer assigns a negative $R$ to all vertices outside the relevant regions. The result is stored in the phyS: `phyS.r`

#### Identifying Roots of Individual Features

The next step is identifying the roots of individual features and by doing so obtaining their individual $f$ threshold. The interval $[R_i, R_j[$ between a node $i$ in augS and its lower neighbour $j$ is assessed whether it includes $R_{cut}$. If both nodes belong to the same hypA $j = i + 1$, if they are located on different hypAs $j =$ `supS.upper[hypS.lower[augS.hypID[`$i$`]]]`. Root nodes do not have a lower neighbour, the lower bound of their associated intervals is set to $R_{min}$. If $R_j < R_{cut} \leq R_i$, $i$ is is marked as being root of a feature.

#### Filtering Small Features

There are many circumstances that can cause small local maxima. Examples are noise in the input data and discontinuities across boundaries of high order elements which locally amplify gradients resulting in maxima of $f$. Especially in regions in which $f$ has values close to $f_{min}$ small local maxima will be identified as individual features consisting of only a few vertices. Such artefacts are filtered by additionally requiring roots of features to have a subtree size that includes at least a user-defined amount of nodes, the minimal feature size *mfs*. The size of a node's subtree can be obtained via AccumulateSubTreeQuantities(`augS.q`, $+$) from Algorithm 4.11, where the input quantity vector is `augS.q` $= \{1, 1, ..., 1\}$. To prevent those artefacts from contaminating visualisations the relevance of their vertices $i$ is set to $R_i = -\varepsilon$.

#### Assigning Vortex Labels to all Vertices

Lastly each feature shall have an ID assigned with which its vertices shall be labeled. The feature IDs are obtained by sorting them by the number of vertices they include in descending order. Hence the feature including most vertices will have ID 0, the second largest ID 1 and so forth. Next all hypAs that include a feature root are identified. Since both $R$ and $f$ are monotonous along any monotonous path in the tree, each hypA can include at most feature root and thus belong to either a single feature or to no feature at all. An array `hypS.featureID` is allocated and each of its elements is initialised with $-1$. Elements of hypAs containing a feature root are

set to the ID of the associated feature. Processing all hypAs by HS order, starting from second to last HS order to HS order zero, each hypA gets assigned the structure ID of the hypA of its lower end, as long as $R$ of the lower end is above $R_{cut}$. Each vertex $i$ for which $R_i \geq R_{cut}$ is labelled with the feature ID of its hypA, all others with $-1$. The pseudocode for this procedure is provided in Algorithm 4.14.

**Algorithm 4.14:** PassLabelsFromRootsToSubtrees() - Passes the labels of the feature roots up to each node of its subtree.

```
for r ∈ {r_max − 2, r_max − 3, ..., 0} do
    for i ∈ {ovh_r, ovh_r + 1, ..., ovh_{r+1} − 1} do in parallel
        if physS.R[augS.node[supS.upper[hypS.lower[i]]]] ≥ R_cut then
            hypS.featureID[i]] = max(hypS.featureID[i]],
                hypS.featureID[augS.hypID[supS.upper[hypS.lower[i]]]])
    end
end
return
```

### 4.2.6 Visualising Features

Features can be visualised by rendering isosurfaces of $R$. Their intensity can be indicated by colouring the isosurfaces according to $f$. Examples in which extracted features are rendered and a discussion on the advantages over classic visualisation methods are provided in Section 5. Tracer can write the $R$ field and the vortex labels to a vtu file, which can be visualised using e.g. ParaView [5]. Via a custom filter the user can also render vortices with specific IDs only. Additionally Tracer has the capability of producing pngs in-transit via the Alpine/Ascent framework [64], which reduces time and memory requirements for visualisations significantly. To do so Tracer copies the $R$ and $f$ fields to the host, which also holds a copy of the mesh. Tracer passes the pointers to the scalar fields and mesh alongside user-defined camera positions and colour map configurations to Alpine/Ascent, which renders isosurfaces and writes pngs to disk.

### 4.2.7 Geometric Quantities of Individual Features

The following paragraphs describe the way geometric entities of individual features are obtained. Each of the subtree's nodes contributes the full entity which is assigned to it. Thus geometrical entities of subtrees and consequently of individual extracted features can be accumulated with Algorithm 4.12.

#### Volumes of Individual Features

The volume of a feature is approximated by adding up the volumes assigned to the nodes of the feature's subtree using Algorithm 4.11. Tracer computes the volume of each node during the initialisation by distributing the volume of each cell in equal parts to its eight vertices. With the augS IDs of feature roots, which were found in Section 4.2.5, the volumes of the features can be gathered from the augS.

### Centre of Individual Features

Tracer locates the centre of an individual feature by obtaining a weighted centroid ${}_j\mathbf{C}$ of the nodes of its subtree $j$. The coordinate ${}_i\mathbf{x}$ of each node $i$ of $j$ is weighted with the volume $V_i$ it occupies:

$$ {}_j\mathbf{C} = \frac{\sum_i {}_i\mathbf{x}\, V_i}{\sum_i V_i}. \tag{4.3} $$

The numerator of Eq. (4.3) is accumulated by summing the weighted coordinates using Algorithm 4.12. The denominator holds the volume of individual features, which have already been obtained. Note that a coordinate component $c$ of the centre is not correct for features that span across a period boundary, if the vector of that periodic boundary also has non-zero value in $c$.

### Assign Features to Regions

In the ini file the user can define one or more parallelepipeds, to which a feature is assigned to if its centre is located within the parallelepiped's body. A feature is assigned to exactly one region. In case its centre lies inside more than one regions it will be assigned to the region with lowest index. If its centre is not within any of the user defined regions it will be assigned to region 0. Each region is provided with a list of all its assigned features, allowing independent analysis of ensembles of features located in different regions. That way e.g. the topological organisation of features in a near wall region can be compared to the topological organisation of features in the wake.

### Bounding Boxes of Individual Features

Axis-aligned bounding boxes of features are specified by finding the minimum and the maximum of each coordinate component. Tracer employs AccumulateSubTreeQuantities(`augS.q`, *op*) from Algorithm 4.12 filling `augS.q` with the respective coordinate component of the nodes and using min and max as operator.

If bounding boxes shall be found in a domain that includes periodic boundaries, each periodic boundary has to be axis-aligned as well. Specifically their periodic vector is only allowed to have a single non-zero component each. Additionally the non-zero component is required to be positive. Features that span across a periodic boundary that is aligned with coordinate direction $x_c$ are detected via their maximum value in $x_c$.

In order to obtain the bounding box of such features, the process described above is repeated: `augS.q` is filled with $x_c$ of the nodes in augS, however before `augS.q` is passed to AccumulateSubTreeQuantities(`augS.q`, *op*), all of its elements $i$ which are smaller than $x_{c,cut}$ will be shifted to the right with the non-zero component periodic vector $p_c$:

$$ \texttt{augS.q[}i\texttt{]} = \texttt{augS.q[}i\texttt{]} + p_c \qquad \forall \qquad \texttt{augS.q[}i\texttt{]} < x_{c,cut}. \tag{4.4} $$

The bounding box of features spanning across periodic boundaries will consequently have its maximum outside the domain. The bounding box of features that extend over the major part of the domain might not be found, in that case the maximum value is set to a very small number so they can be detected in the feature csv file.

### 4.2.8 HS Orders and Tokunaga Indices of the Full Domain

This section describes the routines which compute the numbers required to obtain the Horton indicators and Tokunaga indices, which are explained in Section 2.2.1. For a given order $r$ these are the total number of hypAs $N_r$, the magnitude of hypAs $M_r$, the number of supAs within a hypA $C_r$ and the number of hypAs of order $r$ which join a hypA of order $j$ $N_{r,j}$. The numbers are computed interpreting the full scalar field above $f_{min}$, even though it might consist of multiple relevant regions. These regions however are a subset of a single connected domain, thus their individual trees are sections of a single global tree. The numbers are provided in the form of a histogram, in which the bins hold the different values that occurred, while the associated counts indicates how often a value has occurred. The pseudocode for making the histograms is provided in Algorithm 4.15. The in-situ version of Tracer will accumulate these histograms over multiple snapshots.

**Algorithm 4.15:** MakeHistograms(*stq*, *ov*) - Making histograms for quantities *stq* of order $r$. The offset vector *ov* provides information where the individual segments of *stq* assigned to $r$ start. Unique($q$) removes all duplicates of previous elements of $q$ so that no two elements are the same, Count($x$, $q$) returns the number of elements in $q$ that have value $x$.

```
for i ∈ {0, ..., r_max − 1} do
    bins = Unique({stq[ov_i], ..., stq[ov_{i+1}]})
    for j ∈ {0, ..., len(bins)} do in parallel
        counts_j = Count(bins_j, {stq[ov_i], ..., stq[ov_{i+1}]})
    end
    histos_i = bins, counts
end

return histos
```

#### $N_r$: The Number of Hyper Arcs per HS order

$N_r$ are obtained via the difference of consecutive elements in of the offset vector as shown in Algorithm 4.16. The result is provided as a single histogram in which the bins are associated with $r$ and their counts with $N_r$.

**Algorithm 4.16:** CountHyperArcsPerOrder() - Obtaining number of hypAs per $r$.

```
bins = {0, ..., r_max − 1}
for r ∈ bins do in parallel
    N_r = ovh_{r+1} − ovh_r
end
counts = {N_0, ..., N_{r_max−1}}
return bins, counts
```

### $M_r$: The Magnitude

The distribution of magnitudes $M_r$ of hypAs are provided as a histogram for each HS order $r$. The bins of the histogram hold occurring values of $M_r$, the associated counts specify how often that value has occurred. To obtain the magnitudes all leaves in the supS are tagged with 1, all other nodes with 0 before applying a prefix sum. Since all super nodes along a hypA occure consecutively in the supS and the ordering in the supS is consistent with the ordering of the hypS, the magnitudes of the hypS can be obtained by removing all elements associated with supAs which are not the lowest supA in their hypA. The method to obtain the magnitude can be found in Algorithm 4.17, the compaction algorithm is provided in Algorithm 4.18.

**Algorithm 4.17:** GetMagnitudesPerOrder() - Obtaining the magnitude of hypAs per $r$.

```
1  for i ∈ {0, ..., n_max − 1} do in parallel
2  |  magSup_i = 1
3  end
4  for i ∈ {n_max, ..., n_sup − 1} do in parallel
5  |  magSup_i = 0
6  end
7
8  AccumulateSubTreeQuantitySuper(magSup)
9  magHyp = SuperToHyperCompaction(magSup) // Alg. 4.18
10
11 return MakeHistograms(magHyp)          // Alg. 4.15
```

**Algorithm 4.18:** SuperToHyperCompaction($supSTQ$) - Obtaining subtree quantities of hypAs via subtree quantities of supAs.

```
1  j = 0
2  for i ∈ {0, ..., n_sup − 1} do in parallel
3  |  if hypID_i ≠ hypID_{i+1} then
4  |  |  hypSTQ_j = supSTQ_i
5  |  |  j++
6  end
7
8  return hypSTQ
```

### $C_r$: The Number of Super Arcs per Hyper Arc

$C_r$ are provided as a histogram per HS order $r$. The bins hold the numbers of supAs per hypA that occurred, the associated counts provide how often those numbers have occurred. To obtain $C_r$ Tracer again makes use of the fact that all supAs of a certain hypA appear consecutively in the supS. All super nodes are tagged with 1 and a linear prefix sum by key is performed on the vector, where the keys are the hypA IDs. The detailed algorithm can be found in Algorithm 4.19.

**Algorithm 4.19:** SuperArcsPerHyperArc(*hypIDs*, *ovh*) - Count number of supAs in hypAs.

```
1 for i ∈ {0, ..., n_sup − 1} do in parallel
2 |  sups_i = 1
3 end
4 PrefixSumByKey(sups, hypIDs)
5 hyps = SuperToHyperCompaction(sups)     // Alg. 4.18
6 return MakeHistograms(hyps, ovh)        // Alg. 4.15
```

#### Obtaining $N_{i,j}$: The Tokunaga Indices

$N_{i,j}$ are provided as a histogram per HS order $i$. The bins hold the order $j$ that is joined of a hypA of order $i$, the associated counts provide how often that combination has occurred. To obtain the correct bin, i.e. $j$, simply the hypG of the hypA needs to be taken from the hypS. In case the hypG is not a hypN itself, $j$ will be the HS order of the hypA that the hypG belongs. Should the hypG however be a hypN itself, $j$ will be one order below, since only non-terminal junctions are counted. The detailed algorithm can be found in Algorithm 4.20.

**Algorithm 4.20:** TokunagaIndices() - Obtain Tokunaga indices.

```
1 for i ∈ {0, ..., n_hyp − 1} do in parallel
2 |  tok_i = HS order of hypN_i
3 |  if hypN_i is hypN then
4 |  |  tok_i -= 1
5 end
6 return MakeHistograms(tok, ovh)         // Alg. 4.15
```

### 4.2.9 HS Ratios of Individual Features

The following paragraph describes the implementation of the method to get HS ratios, which were presented in Section 2.2.1, for individual features. The ratios can be added to the feature csv file and accumulated in histograms. There is furthermore the possibility of accumulating individual histograms for features the COM of which is located in a user-defined subregion of the domain. The histograms are subdivided by the topological complexity of the feature, which is defined by the HS order of their root. The bins of the histograms hold the values of the HS ratios, while their associated counts record how often a specific ratio has occurred.

As preparation Tracer first constructs an offset vector *ovf* for each region storing how many features of any given order there are in each subregion. Secondly the the ID of the root supA of each feature is found and stored in *supRoot*, so that the required numbers for the HS ratios can be gathered from supS. The average ratio of a feature is computed by averaging over all ratios built with consecutive orders. Hence Tracer iterates over all HS orders obtaining the HS numbers out of which the ratios can be computed and averaged. The details of this method are

provided in Algorithm 4.21.

**Algorithm 4.21:** GetHSRatiosOfFeatures() - Obtain HS ratios of extracted features.

```
for r ∈ {0, ..., r_max} do
    N_r = GetHypArcsPerOrder(r)                          // Alg. 4.22
    ⟨M_r⟩ = GetAverageMagnitudes(r)                      // Alg. 4.23
    ⟨C_r⟩ = GetSupArcsPerHypArc(r)                       // Alg. 4.24

    if r ≥ 1 then
        AccumulateAndAverageRatios(N_r, ⟨M_r⟩, ⟨C_r⟩, N_{r-1}, ⟨M_{r-1}⟩, ⟨C_{r-1}⟩, r) // Alg. 4.25
end

return
```

### Obtain BPO

To get the number of hypAs of a certain $r$, a component $c$ is added to the supS in which all hypNs of $r$ are initialised with 1, all other nodes with 0. After this component has been accumulated on the super structure, the ${}_iN_r$ for each feature can be gatherd from their root supA. The details of this method are provided in Algorithm 4.22.

**Algorithm 4.22:** GetHypArcsPerOrder($r$) - Count hypAs of order $r$.

```
for i ∈ {0, ..., n_sup − 1} do in parallel
    if hypID_i ≠ hypID_{i−1} and i ∈ {ovs_r, ..., ovs_{r+1} − 1} then
        c_i = 1
    else
        c_i = 0
end
AccumulateSubTreeQuantitySuper(c)
for i ∈ {ovf_r, ..., n_feat − 1} do in parallel
    ₍N_r = c_{supRoot_i}
end
return N_r
```

### Get Average Magnitude

In order to obtain $\langle M_r \rangle$ a component $c$ is added to the supS in which all maxima are initialised with 1, all other nodes with 0. After this component has been accumulated, the accumulated magnitudes for features of order $r$ are gathered. Subsequently all elements of $c$ which are not associated with a the lowest supA of a hypA of order $r$ are reset to 0 before being accumulated again. Now the accumulated magnitudes for features $j$ which have $r_{max,j} = r + 1$ can be gathered. $\langle M_r \rangle$ is gained by dividing the accumulated magnitude by the number of hypAs of order $r$. The details of this method are provided in Algorithm 4.23.

**Algorithm 4.23:** GetAverageMagnitudes($r$) - Obtain $\langle M_r \rangle$ for each extracted feature of order $r$ and above.

**for** $i \in \{0, ..., n_{sup} - 1\}$ **do in parallel**
  **if** $i \in \{0, ..., n_{max} - 1\}$ **then**
    $c_i = 1$
  **else**
    $c_i = 0$
**end**
AccumulateSubTreeQuantitySuper($c$)
**for** $i \in \{ovs_r, ..., ovs_{r+1} - 1\}$ **do in parallel**
  $\langle M_{r,i} \rangle = c_{supRoot_i}$
**end**
**for** $i \in \{0, ..., n_{sup} - 1\}$ **do in parallel**
  **if** $(hypID_i = hypID_{i+1})$ or $i \notin \{ovs_r, ..., ovs_{r+1} - 1\}$ **then**
    $c_i = 0$
**end**
AccumulateSubTreeQuantitySuper($c$)
**for** $i \in \{ovf_{r+1}, ..., n_{feat} - 1\}$ **do in parallel**
  $\langle M_{r,i} \rangle = \frac{M_i}{N_{r,i}}$
**end**
**return** $\langle M_r \rangle$

#### Get Average Number of supAs per hypA

To get the average number of supAs per hypAs of a certain $r$, a component $c$ is added to the supS in which all supNs of $r$ are initialised with 1, all other nodes with 0. After this component has been accumulated on the supS, the $C_r$ for each feature of order $r$ or above can be gatherd from their root supA and divided by their $N_r$ to obtain $\langle C_r \rangle$. Algorithm 4.24 provides the details of this mehtod.

**Algorithm 4.24:** GetSupArcsPerHypArc($r$) - Obtain average number of supAs for each hypA of order $r$.

**for** $i \in \{0, ..., n_{sup} - 1\}$ **do in parallel**
  **if** $i \in \{ovs_r, ..., ovs_{r+1} - 1\}$ **then**
    $c_i = 1$
  **else**
    $c_i = 0$
**end**
AccumulateSubTreeQuantitySuper($c$)
**for** $i \in \{ovf_r, ..., n_{sup} - 1\}$ **do in parallel**
  $\langle C_{r,i} \rangle = \frac{c_{supRoot_i}}{N_{r,i}}$
**end**
**return** $\langle C_r \rangle$

#### Accumulate and Average Ratios

The HS ratios provided in Eqs. (2.2-2.4) are obtained by dividing HS numbers of consecutive $r$ by each other and are accumulated over all HS orders. To get the averages, the ratios are divided by the the order of each features root. A detailed description is provided in Algorithm 4.25.

**Algorithm 4.25:** AccumulateAndAverageRatios($N_r, \langle M_r \rangle, \langle C_r \rangle, N_{r-1}, \langle M_{r-1} \rangle, \langle C_{r-1} \rangle, r$) - Accumulate and average HS ratios for extracted features.

**for** $i \in \{ovf_r, ..., n_{feat} - 1\}$ **do in parallel**
  $P_{N,i} = P_{N,i} + \frac{N_{r-1,i}}{N_{r,i}}$
  $P_{M,i} = P_{M,i} + \frac{\langle M_{r,i} \rangle}{\langle M_{r-1,i} \rangle}$
  $P_{C,i} = P_{C,i} + \frac{\langle C_{r,i} \rangle}{\langle C_{r-1,i} \rangle}$
  **if** $i < ovf_{r+1}$ **then**
    $P_{N,i} = \frac{P_{N,i}}{r}$
    $P_{M,i} = \frac{P_{M,i}}{r}$
    $P_{C,i} = \frac{P_{C,i}}{r}$
**end**
**return**

## 4.3 Testing

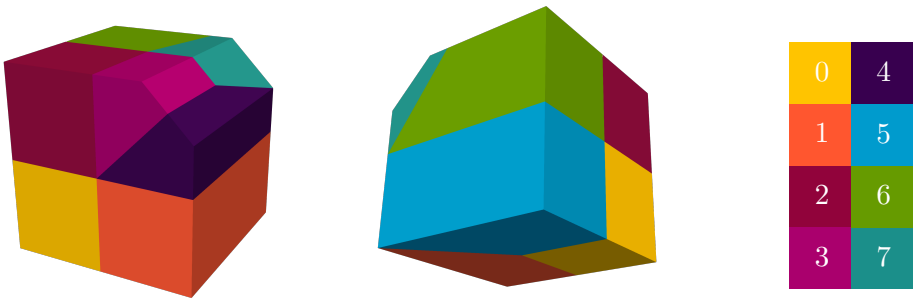
This section presents the major tests that have been used for the verification of Tracer. Section 4.3.1 introduces topologically simple test cases, the results of which can be compared to analytical solutions. Section 4.3.2 describes methods to check the consistency of results of topologically complex cases and performs them on the examples. Finally, Section 4.3.3 explains how the correctness of the software was assessed during development.

### 4.3.1 Topologically Simple Test Cases

This paragraph introduces two topologically simple test cases named *unstructured mesh* and *artiTree*, the results of which can be compared with analytical results. In detail this means checking the correctness of JT and checking the correctness of the tree analysis by hand. The correctness of the tree analysis e.g. includes correct HS ordering and ratios and correct number and size of identified features for a given relevance threshold. Note that topologically simple does not necessarily mean a small input graph, i.e. a small number of vertices and edges, but a small number of critical points, i.e. small supS and hypS.

#### Unstructured Mesh

The *unstructured mesh* case is made up of 27 vertices forming eight irregular hexahedral cells, which are depicted in Figure 4.4. Coordinates and scalar values of the vertices as well as the resulting edge graph are provided in Tables 4.1 and 4.2. The small mesh size allows comparison also of the augmented structure. The fully augmented tree is shown in Figure 4.5, for comparison Figure 4.6 provides hypS, supS and augS which were found by Tracer and are in agreement with the analytically obtained JT.




(a) (b) cell ID

Figure 4.4: (a) front view and (b) back view of the elements of the *unstructured mesh* case, the legend on the right assigns colours to cell ID.

Table 4.1: Vertex coordinates and scalar values for unstructured mesh case.

| | $V_0$ | $V_1$ | $V_2$ | $V_3$ | $V_4$ | $V_5$ | $V_6$ | $V_7$ | $V_8$ | $V_9$ | $V_{10}$ | $V_{11}$ | $V_{12}$ | $V_{13}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x$ | $-1$ | 0 | 1 | $-1$ | 0 | 1 | $-1$ | 0 | 0.5 | 0.75 | 1 | $-1$ | 0 | $-1$ |
| $y$ | $-1$ | $-1$ | $-1$ | 0 | 0 | 0 | 1 | 1 | 1 | 0.75 | 0.5 | $-1$ | $-1$ | 0 |
| $z$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| $f$ | 2.01 | 5.01 | 4.01 | 6.01 | 4.02 | 0.01 | 2.02 | 5.02 | 3.01 | $-0.99$ | 4.03 | 4.04 | 6.02 | 3.02 |

| | $V_{14}$ | $V_{15}$ | $V_{16}$ | $V_{17}$ | $V_{18}$ | $V_{19}$ | $V_{20}$ | $V_{21}$ | $V_{22}$ | $V_{23}$ | $V_{24}$ | $V_{25}$ | $V_{26}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x$ | 0 | $-1$ | 0 | 0.5 | 0.75 | $-1$ | 1 | $-1$ | 1 | $-1$ | $\frac{1}{4}$ | 0.5 | 1 |
| $y$ | 0 | 1 | 1 | 1 | 0.75 | $-1$ | $-1$ | 0 | 0 | 1 | 1 | 1 | 0.5 |
| $z$ | 0 | 0 | 0 | 0 | 0 | $-1$ | $-1$ | $-1$ | $-1$ | $-1$ | $-1$ | $-1$ | $-1$ |
| $f$ | 5.03 | 4.05 | 3.03 | 4.06 | 3.04 | 5.04 | 3.05 | 2.03 | 4.07 | 3.06 | 2.04 | 3.07 | 6.03 |

Table 4.2: Edge list of the unstructured grid test case.

| edge ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| edgeStart | $V_0$ | $V_0$ | $V_0$ | $V_1$ | $V_1$ | $V_1$ | $V_2$ | $V_2$ | $V_3$ | $V_3$ | $V_3$ | $V_4$ | $V_4$ | $V_4$ |
| edgeEnd | $V_1$ | $V_3$ | $V_{11}$ | $V_2$ | $V_4$ | $V_{12}$ | $V_5$ | $V_{20}$ | $V_4$ | $V_6$ | $V_{13}$ | $V_5$ | $V_7$ | $V_9$ |

| edge ID | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| edgeStart | $V_4$ | $V_5$ | $V_5$ | $V_6$ | $V_6$ | $V_7$ | $V_7$ | $V_8$ | $V_8$ | $V_9$ | $V_9$ | $V_{10}$ | $V_{11}$ | $V_{11}$ |
| edgeEnd | $V_{14}$ | $V_{10}$ | $V_{22}$ | $V_7$ | $V_{15}$ | $V_8$ | $V_{16}$ | $V_9$ | $V_{17}$ | $V_{10}$ | $V_{18}$ | $V_{26}$ | $V_{12}$ | $V_{13}$ |

| edge ID | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| edgeStart | $V_{11}$ | $V_{12}$ | $V_{12}$ | $V_{13}$ | $V_{13}$ | $V_{13}$ | $V_{14}$ | $V_{14}$ | $V_{14}$ | $V_{15}$ | $V_{15}$ | $V_{16}$ | $V_{16}$ |
| edgeEnd | $V_{19}$ | $V_{14}$ | $V_{20}$ | $V_{14}$ | $V_{15}$ | $V_{21}$ | $V_{16}$ | $V_{18}$ | $V_{22}$ | $V_{16}$ | $V_{23}$ | $V_{17}$ | $V_{24}$ |

| edge ID | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| edgeStart | $V_{17}$ | $V_{17}$ | $V_{18}$ | $V_{19}$ | $V_{19}$ | $V_{20}$ | $V_{21}$ | $V_{21}$ | $V_{22}$ | $V_{22}$ | $V_{23}$ | $V_{24}$ | $V_{25}$ |
| edgeEnd | $V_{18}$ | $V_{25}$ | $V_{26}$ | $V_{20}$ | $V_{21}$ | $V_{22}$ | $V_{22}$ | $V_{23}$ | $V_{24}$ | $V_{26}$ | $V_{24}$ | $V_{25}$ | $V_{26}$ |

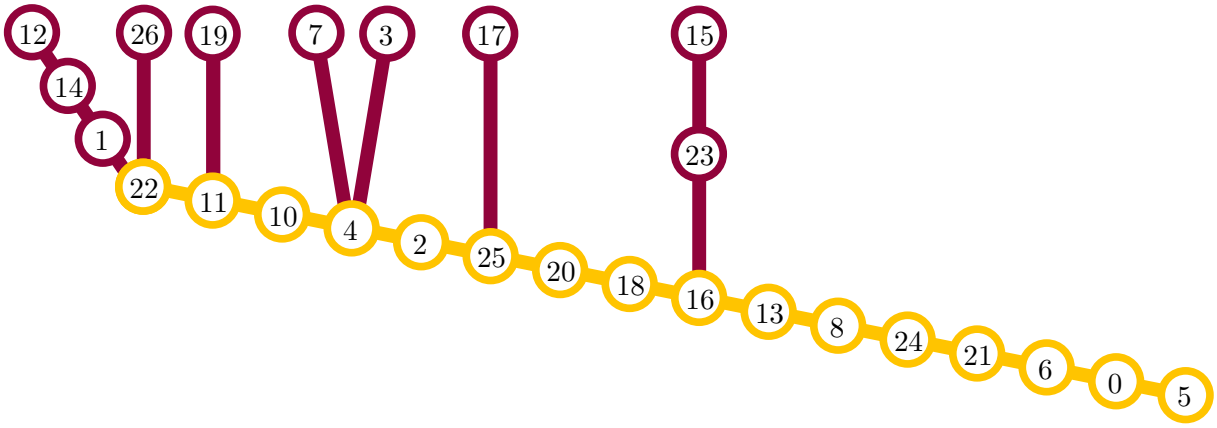



Figure 4.5: Fully augmented JT for the *unstructured mesh* case and $f_{min} = 0$. Red nodes and arcs are of HS order $r = 0$, yellow nodes and arcs of order $r = 1$.

### artiTree

The *artiTree* case defined on an axis-aligned cuboidal domain, which is shown in Figure 4.7, and discretised on a regularly spaced grid consisting of $128 \times 4 \times 4$ cells. The IDs of the vertices which are located on the $x$-axis are identical to their $x$-coordinate. The scalar value associated with the vertices is defined by:

$$f(x, y, z) = f(x, 0, 0) - y - z, \tag{4.5}$$

with $f(x, 0, 0)$ being provided in Figure 4.8. Since the domain only expands into the non-negative coordinate directions all maxima and join saddles are guaranteed to be located on the $x$-axis and hence on vertices 0-128. The JTs of $f$ for $f_{min} = 0$, which were obtained analytically, are

| hyper structure | |
|---|---|
| upper end | lower end |
| 3, | 4 |
| 7, | 4 |
| 12, | 22 |
| 15, | 16 |
| 17, | 25 |
| 19, | 11 |
| 26, | 22 |
| 22, | 22 |

| super structure | |
|---|---|
| upper end | lower end |
| 3, | 4 |
| 7, | 4 |
| 12, | 22 |
| 15, | 16 |
| 17, | 25 |
| 19, | 11 |
| 26, | 22 |
| 22, | 11 |
| 11, | 4 |
| 4, | 25 |
| 25, | 16 |
| 16, | 22 |

| augmented structure | | |
|---|---|---|
| hypP | supP | vertID |
| 3, | 3, | 3 |
| 7, | 7, | 7 |
| 12, | 12, | 12 |
| 12, | 12, | 14 |
| 12, | 12, | 1 |
| 15, | 15, | 15 |
| 15, | 15, | 23 |
| 17, | 17, | 17 |
| 19, | 19, | 19 |
| 26, | 26, | 26 |
| 22, | 22, | 22 |
| 22, | 11, | 11 |
| 22, | 11, | 10 |
| 22, | 4, | 4 |
| 22, | 4, | 2 |
| 22, | 25, | 25 |
| 22, | 25, | 20 |
| 22, | 25, | 18 |
| 22, | 16, | 16 |
| 22, | 16, | 13 |
| 22, | 16, | 8 |
| 22, | 16, | 24 |
| 22, | 16, | 21 |
| 22, | 16, | 6 |
| 22, | 16, | 0 |
| 22, | 16, | 5 |

Figure 4.6: hypS, supS and augS for the *unstructured mesh* case and $f_{min} = 0$. In the augS instead of the hypIDs and supIDs the vertex ID of the hypA or supA are provided.

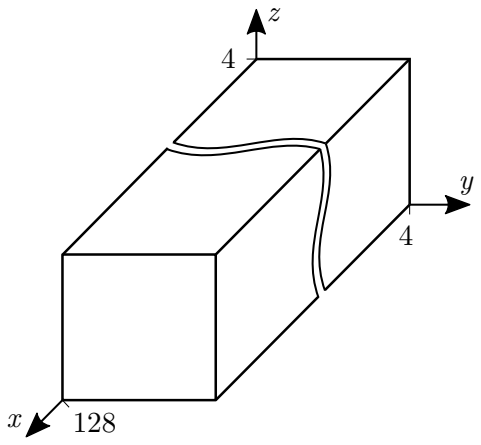



Figure 4.7: Domain of the artiTree case.

provided in Figure 4.9. The supS and hypS identified by Tracer are provided in Figure 4.10 and are in agreement with the analytically obtained JTs.

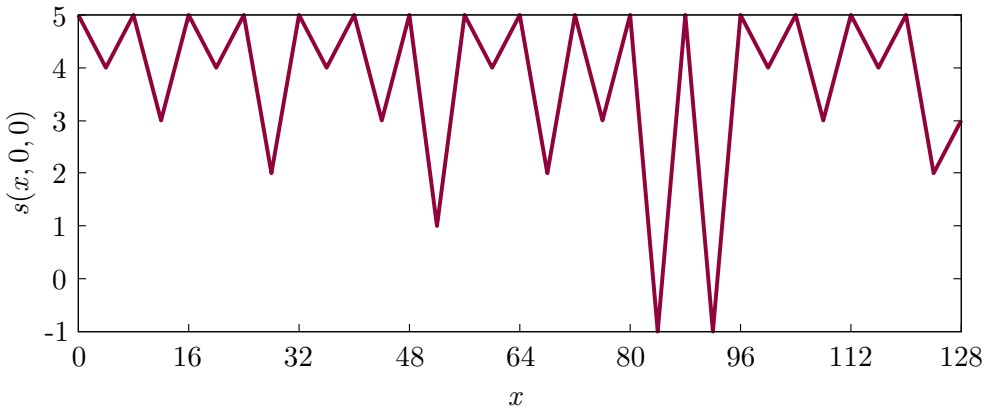



Figure 4.8: $f(x,0,0)$ for the artiTree case.

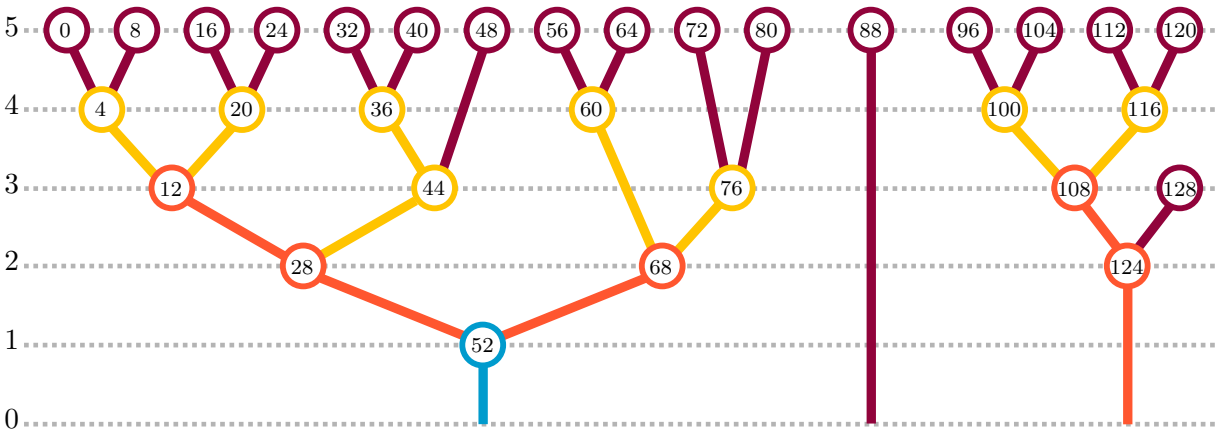



Figure 4.9: JTs for the artiTree case and $f_{min} = 0$. The nodes are vertically positioned according to their $f$. Red nodes and arcs are of HS order $r = 0$, for yellow $r = 1$, for orange $r = 2$ and for blue $r = 3$.

### 4.3.2 Topologically Complex Test Cases

Results of topologically complex cases, e.g. CFD data, are generally too large to generate and compare by hand. However, such results can be tested for consistency. One such method is comparing the numbers of extrema and saddles in a tree. Consider a JT $T$ with $n_{max,T}$ maxima. Every node in $T$ apart from the minimum is connected to exactly one node below. Join saddles are nodes of degree $\lambda > 2$. At such a saddle $\lambda - 1$ connections from upwards are joined to a singles connection downwards, hence reducing the number of links by $\lambda - 2$. Like all JTs $T$ has a single root, meaning that $n_{max,T}$ needs to be reduced to 1 by the saddles in $T$. Therefore a single JT $T$ satisfies:

$$n_{max,T} - \sum_{\lambda=3}^{\lambda_{max}} (\lambda - 2)\, n_{\lambda,T} = 1, \tag{4.6}$$

with $n_\lambda$ being the number of joins of degree $\lambda$. The number of JTs in a domain is equal to the number of relevant regions $n_{rr}$, consequently a full domain satisfies:

$$n_{max} - \sum_{\lambda=3}^{\lambda_{max}} (\lambda - 2)\, n_\lambda = n_{rr}. \tag{4.7}$$

| hyper structure | | super structure | |
|---|---|---|---|
| upper end | lower end | upper end | lower end |
| 0, | 4 | 0, | 4 |
| 8, | 4 | 8, | 4 |
| 16, | 20 | 16, | 20 |
| 24, | 20 | 24, | 20 |
| 32, | 36 | 32, | 36 |
| 40, | 36 | 40, | 36 |
| 48, | 44 | 48, | 44 |
| 56, | 60 | 56, | 60 |
| 64, | 60 | 64, | 60 |
| 72, | 76 | 72, | 76 |
| 80, | 76 | 80, | 76 |
| 96, | 100 | 96, | 100 |
| 104, | 100 | 104, | 100 |
| 112, | 116 | 112, | 116 |
| 120, | 116 | 120, | 116 |
| 128, | 124 | 128, | 124 |
| 88, | 88 | 88, | 88 |
| 4, | 12 | 4, | 12 |
| 20, | 12 | 20, | 12 |
| 36, | 28 | 36, | 44 |
| 60, | 68 | 44, | 28 |
| 76, | 68 | 60, | 68 |
| 100, | 108 | 76, | 68 |
| 116, | 108 | 100, | 108 |
| 12, | 52 | 116, | 108 |
| 68, | 52 | 12, | 28 |
| 108, | 108 | 28, | 52 |
| 52, | 52 | 68, | 52 |
| | | 108, | 124 |
| | | 124, | 108 |
| | | 52, | 52 |

Figure 4.10: hypS and supS of the artiTree case identified by Tracer. The lower end of root arcs is set to the upper end of their hypA.

This test is exemplarily shown on the the flow field around an aerofoil from Section 5 and on the TGV case from Section 6. In both cases $f_{min}$ was set to 0. In a snapshot of the SD7003 case at 40 convective time units Tracer has identified 84 615 maxima, 72 651 saddles of degree 3, 2 271 saddles of degree 4, 19 saddles of degree 5 and 1 saddle of degree 6, which are balanced by 7 361 relevant regions. In a snapshot of the TGV case at 15 convective time units Tracer has identified 81 240 maxima, 76 914 saddles of degree 3 and 1 591 saddles of degree 4, which are balanced by 1 144 relevant regions.

Another way to test the consistency of results can be achieved via thresholding. The most obvious being that:

$$n_{regions} = 1 \qquad \forall \qquad f_{min} \leq f_{min,glob}. \tag{4.8}$$

Furthermore without filtering small features if $R_{cut} = 0$ the number of extracted features $n_{ef}$ has to be equal to $n_{rr}$:

$$n_{ef} = n_{rr} \qquad \text{if} \qquad R_{cut} = 0, \tag{4.9}$$

if $R_{cut} \to 1$ without filtering small features:

$$\lim_{R_{cut} \to 1} n_{ef} = n_{max}, \tag{4.10}$$

and with filtering small features:

$$\lim_{R_{cut} \to 1} n_{ef} = 0. \tag{4.11}$$

Note that theoretically Eq. (4.11) can be not true in the unlikely case of a number of vertices larger than *mfs* around the maximum have the same scalar value as the maximum. An analysis of the number of identified features depending on $R_{cut}$ and *mfs* is of the TGV case from Section 6 is provided in Figures 6.2 (a, b). Without filtering small features Tracer identifies 1 144 features with a relevance threshold $R_{cut} = 0$, which is equal to $n_{rr}$. With $R_{cut} = 1$ Tracer extracts 81 240 features, which is equal to $n_{max}$. From $R_{cut} = 0$ to $R_{cut} = 1$ the $n_{ef}$ is monotonically ascending since features can split up but not disappear. If *mfs* is set to a positive value Eq. (4.11) is satisfied, as the number of extracted features reaches 0 for $R_{cut} = 1$.

### 4.3.3 Verification Cases for Software Development

For software development a selection of topologically simple cases and topologically complex cases, which were derived from real-world examples, were implemented in an automated testing routine. The correctness of results has been verified by hand for topologically simple test cases, or checked for consistency according to Section 4.3. Once approved certain output files like tree structures or the features csv file were saved to be compared with future outputs.

## 4.4 Performance Analysis for the In-Situ Version

The main objective of Tracer is the in-situ analysis of flow field data generated by a CFD solver. Consequently the impact on the performance of a CFD simulation when Tracer is added in-situ is of higher interest than how Tracer compares to other JT construction software. In order to analyse Tracer's performance the increase in memory usage and runtime were assessed when applying Tracer in-situ to a turbulent flow simulation conducted with PyFR [107]. To interpolate the solution at the vertices of the linearly subdivided mesh PyFR employs a module called vis plugin. The contribution of the vis plugin to additional memory usage and runtime shall also be part of this assessment. The performance analysis was done on a Taylor-Green vortex (TGV) at $Re = 1\,600$ using the setup for the flow simulation by Witherden and Jameson [106], which is presented in detail in Section 6.1. The simulation was conducted with single precision floating point numbers and the analysed time interval was $t = [0, 20]$ using a constant time step of $\Delta t = 7.5 \cdot 10^{-4}$ resulting in $n_{\mathrm{PyFR}} = 225\,375$ time steps. Tracer identified and counted features based on $R_{cut} = 0.4$ and $mfs = 27$. All runs were conducted on a Nvidia V100 GPU, the versions of operating system, compilers and driver are specified in Table 4.3.

Carrying out the performance analysis on a TGV simulation using single precision will return conservative results on Tracer's performance relative to the performance of PyFR. Single precision floating point numbers minimise both the amount of system memory occupied by PyFR and the time required by PyFR to advance the flow field one step in time. Additionally no

Table 4.3: Versions of operating system, compilers and driver used for the performance analysis.

| | |
|---|---|
| operating system: | CENTOS Linux 7.4.1708 |
| C/C++ compiler: | gcc 4.8.5 |
| CUDA compiler: | Nvidia Cuda compiler V10.0.130 |
| GPU driver: | Nvidia driver version 410.48 |

anti-aliasing method was applied in PyFR, which also minimises runtime.

### 4.4.1 Memory Consumption

To analyse the memory consumption the TGV was run using PyFR, the vis plugin and Tracer in the three configurations provided in Table 4.4. In configuration A $m_A = m_{\text{PyFR}} = 4\,903\,\text{MiB}$ of memory were used by PyFR. The memory usage of configurations B $m_\text{B}$ and C $m_\text{C}$, which depend on the levels of subdivision of the mesh, are provided in Table 4.5. The level of subdivision is the number of linear cells each high-order element is split up in each direction. If the level of subdivisions is e.g. 4, every high-order hexahedron of the computational mesh is subdivided into $4 \times 4 \times 4 = 64$ linear hexahedra.

The memory $m_{\text{visPlugin}}$ occupied by the vis plugin is obtained by subtracting $m_{\text{PyFR}}$ from $m_\text{B}$:

$$m_{\text{visPlugin}} = m_\text{B} - m_{\text{PyFR}}, \tag{4.12}$$

Table 4.4: Configurations which were compared in the performance analysis. A green tick indicates the module was used in the configuration, a red cross means it was not.

| configuration | PyFR | vis plugin | Tracer |
|---|---|---|---|
| A | ✓ | ✗ | ✗ |
| B | ✓ | ✓ | ✗ |
| C | ✓ | ✓ | ✓ |

Table 4.5: Memory usage of configurations A $m_\text{A}$, B $m_\text{B}$ and C $m_\text{C}$ and memory requirements for the vis plugin $m_{\text{visPlugin}}$ and Tracer $m_{\text{Tracer}}$ normalised with the memory used by PyFR $m_{\text{PyFR}}$. All are provided for a range of levels of subdivision.

| level of subdivision | 3 | 4 | 5 | 6 |
|---|---|---|---|---|
| $m_{\text{PyFR}} = m_\text{A}$ | | 4 903 MiB | | |
| $m_\text{B}$ | 5 065 MiB | 5 221 MiB | 5 451 MiB | 5 771 MiB |
| $m_\text{C}$ | 5 531 MiB | 6 287 MiB | 7 524 MiB | 9 309 MiB |
| $\frac{m_{\text{visPlugin}}}{m_{\text{PyFR}}}$ | 0.0330 | 0.0649 | 0.112 | 0.177 |
| $\frac{m_{\text{Tracer}}}{m_{\text{PyFR}}}$ | 0.0950 | 0.217 | 0.423 | 0.722 |
| $\frac{m_{\text{visPlugin}}+m_{\text{Tracer}}}{m_{\text{PyFR}}}$ | 0.128 | 0.282 | 0.535 | 0.899 |

analogously the memory $m_{\text{Tracer}}$ used by Tracer is the difference

$$m_{\text{Tracer}} = m_{\text{C}} - m_{\text{B}}. \tag{4.13}$$

$m_{\text{visPlugin}}$ and $m_{\text{Tracer}}$ normalised by $m_{\text{PyFR}}$ are also provided in Table 4.5. The memory overhead for the highest analysed level of subdivision is 89.9 % and only 53.5 % for the second highest. Using double precision floating point number instead of single precision increases $m_{\text{PyFR}}$ to 9103 MiB. Since Tracer only analyses the data and does not use it to advance the flow field in time it always operates on single precision. The expected memory overhead for a double precision simulation analysed at highest assessed level of subdivision is therefore expected to be in the order of 50 %. It can hence be concluded that applying Tracer in-situ comes at reasonable costs in system memory.

### 4.4.2 Run Time

The total runtimes $\tau_{\text{total}}$ are compared between configurations A, B and C at various levels of subdivision. In configurations B and C the vis plugin is applied at every time step, hence $n_{\text{visPlugin}} = n_{\text{PyFR}}$, in configuration C Tracer was applied $n_{\text{Tracer}} = 201$ times. Table 4.6 provides $\tau_{\text{total}}$ for all three configurations and for various levels of subdivision.

$\tau_{\text{total}}$ is comprised of:

$$\tau_{\text{total}} = n_{\text{PyFR}}\tau_{\text{PyFR}} + n_{\text{visPlugin}}\tau_{\text{visPlugin}} + n_{\text{Tracer}}\tau_{\text{Tracer}}, \tag{4.14}$$

with the average time $\tau_{\text{PyFR}}$ required by PyFR to advance the flow field by one time step, the time $\tau_{\text{visPlugin}}$ required by PyFR's vis plugin to interpolate the solution at the vertices of the linearly subdivided mesh and the time $\tau_{\text{Tracer}}$ that Tracer takes to compute $Q$ from the velocity vector field, build the JT, compute $R$ and identify and count features. Using Eq. (4.14) and the runtimes from Table 4.6 $\tau_{\text{PyFR}}$, $\tau_{\text{visPlugin}}$ and $\tau_{\text{Tracer}}$ result in the values provided in Table 4.7.

By dividing Eq. (4.14) by $n_{\text{PyFR}}\tau_{\text{PyFR}}$ the normalised increase in runtime $\frac{\tau_{\text{total, C}}}{\tau_{\text{total, A}}}$ is obtained parametrised by $\frac{n_{\text{PyFR}}}{n_{\text{Tracer}}}$, the average number of PyFR time steps between two evaluations of the flow field by Tracer:

$$\frac{\tau_{\text{total, C}}}{\tau_{\text{total, A}}} = 1 + \frac{\tau_{\text{visPlugin}}}{\tau_{\text{PyFR}}} + \frac{\tau_{\text{Tracer}}}{\tau_{\text{PyFR}}} \frac{1}{\left(\frac{n_{\text{PyFR}}}{n_{\text{Tracer}}}\right)}. \tag{4.15}$$

The values of $\frac{\tau_{\text{visPlugin}}}{\tau_{\text{PyFR}}}$ and $\frac{\tau_{\text{Tracer}}}{\tau_{\text{PyFR}}}$ for different levels of subdivision are provided in Table 4.7. Table 4.8 provides total normalised runtime $\frac{\tau_{\text{total, C}}}{\tau_{\text{total, A}}}$ for selected values of $\frac{n_{\text{PyFR}}}{n_{\text{Tracer}}}$ and various

Table 4.6: Number of times each module was applied for each configuration and their total runtime $\tau_{\text{total}}$ for various levels of subdivision.

| configuration | $n_{\text{PyFR}}$ | $n_{\text{visPlugin}}$ | $n_{\text{Tracer}}$ | $\tau_{\text{total, 3}}$ | $\tau_{\text{total, 4}}$ | $\tau_{\text{total, 5}}$ | $\tau_{\text{total, 6}}$ |
|---|---|---|---|---|---|---|---|
| A | 225 375 | 0 | 0 | | 44 071 s | | |
| B | 225 375 | 225 375 | 0 | 44 252 s | 44 548 s | 44 957 s | 45 275 s |
| C | 225 375 | 225 375 | 201 | 44 374 s | 44 734 s | 45 174 s | 45 876 s |

Table 4.7: Time $\tau_{\text{PyFR}}$ required by PyFR to advance the flow field by a time step, the time $\tau_{\text{visPlugin}}$ required by the vis plugin to interpolate the high-order solution at vertices of the subdivided mesh and the time $\tau_{\text{Tracer}}$ required by Tracer to compute $Q$ from the velocity vector field, build the JT, compute $R$ and identify and count features. Additionally provided are $\tau_{\text{visPlugin}}$ and $\tau_{\text{Tracer}}$ normalised with $\tau_{\text{PyFR}}$. All times are averages based on a comparison of the runtimes of configurations A, B and C.

| level of subdivision | 3 | 4 | 5 | 6 |
|---|---|---|---|---|
| $\tau_{\text{PyFR}}$ | $1.955 \cdot 10^{-1}\,\text{s}$ | | | |
| $\tau_{\text{visPlugin}}$ | $8.031 \cdot 10^{-4}\,\text{s}$ | $1.983 \cdot 10^{-3}\,\text{s}$ | $3.931 \cdot 10^{-3}\,\text{s}$ | $5.342 \cdot 10^{-3}\,\text{s}$ |
| $\tau_{\text{Tracer}}$ | $6.070 \cdot 10^{-1}\,\text{s}$ | $9.254 \cdot 10^{-1}\,\text{s}$ | $1.080\,\text{s}$ | $2.990\,\text{s}$ |
| $\frac{\tau_{\text{visPlugin}}}{\tau_{\text{PyFR}}}$ | $4.108 \cdot 10^{-3}$ | $1.015 \cdot 10^{-2}$ | $2.011 \cdot 10^{-2}$ | $2.732 \cdot 10^{-2}$ |
| $\frac{\tau_{\text{Tracer}}}{\tau_{\text{PyFR}}}$ | 3.105 | 4.733 | 5.522 | 15.29 |

Table 4.8: Total runtime $\tau_{\text{total, C}}$ of configuration C normalised by the total runtime $\tau_{\text{total, A}}$ according to Eq. (4.15) using $\frac{\tau_{\text{visPlugin}}}{\tau_{\text{PyFR}}}$ and $\frac{\tau_{\text{Tracer}}}{\tau_{\text{PyFR}}}$ from Table 4.7 for selected values of $\frac{n_{\text{PyFR}}}{n_{\text{Tracer}}}$ and various levels of subdivision.

| $\frac{n_{\text{PyFR}}}{n_{\text{Tracer}}}$ | 1 | 10 | 25 | 50 | 100 | 250 | 500 | 1 000 | 10 000 |
|---|---|---|---|---|---|---|---|---|---|
| $\frac{\tau_{\text{total, C, 3}}}{\tau_{\text{total, A}}}$ | 4.109 | 1.315 | 1.128 | 1.066 | 1.035 | 1.017 | 1.010 | 1.007 | 1.004 |
| $\frac{\tau_{\text{total, C, 4}}}{\tau_{\text{total, A}}}$ | 5.744 | 1.483 | 1.199 | 1.105 | 1.057 | 1.029 | 1.020 | 1.015 | 1.011 |
| $\frac{\tau_{\text{total, C, 5}}}{\tau_{\text{total, A}}}$ | 6.542 | 1.572 | 1.241 | 1.131 | 1.075 | 1.042 | 1.031 | 1.026 | 1.021 |
| $\frac{\tau_{\text{total, C, 6}}}{\tau_{\text{total, A}}}$ | 16.31 | 2.556 | 1.639 | 1.333 | 1.180 | 1.088 | 1.058 | 1.043 | 1.029 |

levels of subdivision as a result of Eq. (4.15) and the values for $\frac{\tau_{\text{visPlugin}}}{\tau_{\text{PyFR}}}$ and $\frac{\tau_{\text{Tracer}}}{\tau_{\text{PyFR}}}$ from Table 4.7. Defining the normalised increase in runtime caused by Tracer as

$$\alpha = \frac{\tau_{\text{Tracer}}}{\tau_{\text{PyFR}}} \frac{1}{\left(\frac{n_{\text{PyFR}}}{n_{\text{Tracer}}}\right)}, \tag{4.16}$$

Figure 4.11 shows $\alpha$ as a function of $\frac{n_{\text{PyFR}}}{n_{\text{Tracer}}}$.

The size of a time step of PyFR is limited by the time information is transported between adjacent solution points. The most restrictive time step limit for Tracer is when the identified features shall be tracked. Those features move only with advection velocity and extend over multiple solution points in each direction. Consequently $n_{\text{PyFR}}$ is expected to be at least an order of magnitude above $n_{\text{Tracer}}$. To provide a rough estimate, the investigation in Section 6.3 was carried out using six levels of subdivision and $\frac{n_{\text{PyFR}}}{n_{\text{Tracer}}} \approx 225$ resulting in 9.2 % of additional runtime. Renderings from that configuration were used to make smooth videos of the TGV, providing evidence that $\Delta t_{\text{Tracer}}$ was below the threshold for tracking features. Additionally $\tau_{\text{PyFR}}$ will be significantly bigger for simulations run with double precision, while $\tau_{\text{Tracer}}$ will be unaffected as it is always operating on single precision floating point numbers. The additional runtime when adding Tracer in-situ to a simulation conducted by PyFR is thus expected to be well within the single-digit percentage range.

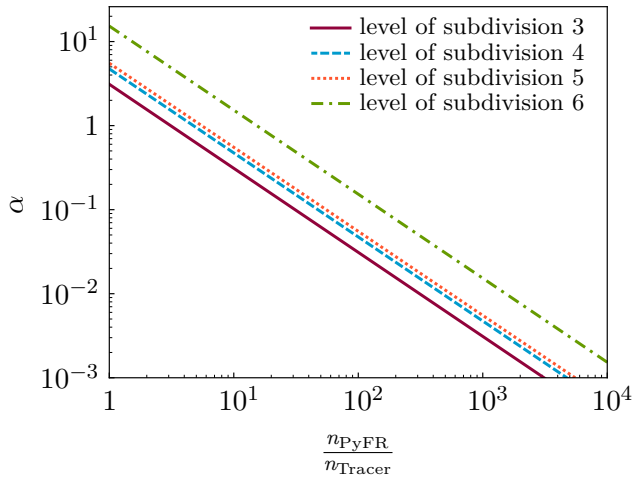



Figure 4.11: The normalised increase in runtime $\alpha$ caused by Tracer as a function of $\frac{n_{\mathrm{PyFR}}}{n_{\mathrm{Tracer}}}$ for various levels of subdivisions. $\alpha$ is plotted as defined in Eq. (4.16) with $\frac{\tau_{\mathrm{Tracer}}}{\tau_{\mathrm{PyFR}}}$ taken from Table 4.7.

# 5 Visual Comparison between Vortex Identification via a Global Threshold and Topology Based Vortex Identification

Let us visually compare vortex identification via the classic approach using a global $Q$ threshold with the topology based approach. The comparison is done on a flow around an SD7003 aerofoil produced by Vermeire et al. [103]. This flow field is well suited for such a comparison, as it includes features over a wide range of intensities and sizes. The setups of the flow simulation and the topological analysis using Tracer are provided in Section 5.1. Section 5.2 presents renderings of isosurfaces of constant $Q$ and constant relevance $R$ as defined in Eq. (2.1) and discusses the differences between them.

## 5.1 Setup

The data used in this chapter is from a flow around a SD7003 aerofoil at a Reynolds number of $Re = 60\,000$ and a Mach number of $Ma = 0.2$ conducted by Vermeire et al. [103]. A summary of the simulation setup is provided in Table 5.1. A structured mesh of high-order hexahedra elements was used in the boundary layer region, with a fully unstructured and refined wake region behind the aerofoil capturing the turbulent wake. For the topological analysis each high-order hexahedron of the mesh was subdivided into $4 \times 4 \times 4$ linear hexahedra. Tracer computed $R$ based on the $Q$-criterion field with an $f_{min}$ of $Q = 0$. To remove visual clutter $R$ was set to $-\varepsilon$ in features the sizes of which were below $mfs = 27$ based on $R_{cut} = 0.4$. A summary of the setup for Tracer is provided in Table 5.2.

Table 5.1: Setup of the simulation [103] of the flow around a SD7003 aerofoil from which the $Q$-criterion field was obtained.

| | |
|---|---|
| solver: | PyFR |
| equation: | compressible Navier-Stokes |
| solution basis order: | 4 |
| Reynolds number: | 60 000 |
| Mach number: | 0.2 |
| angle of attack: | 8° |

Table 5.2: Setup of Tracer for generating the $R$ field of the flow around the SD7003 profile.

| | |
|---|---|
| level of subdivision: | 4 |
| mesh: | 8 833 536 hexahedra |
| analysed scalar field $f$: | $Q$-criterion |
| $f_{min}$: | 0 |
| feature detection threshold $R_{cut}$: | 0.4 |
| minimal feature size *mfs*: | 27 vertices |

## 5.2 Comparison between Isosurfaces of $Q$ and $R$ in the Flow around an SD7003 Aerofoil

Let us visually compare isosurfaces of $Q$ with isosurfaces of $R$, which was generated by Tracer based on $Q$. Depicted features are hence considered to identify vortices. The focus lies on analysing in which part of the flow vortices are detected and how well they are resolved. Furthermore it shall be assessed how these observations are affected by changing $Q_{iso}$ or $R_{iso}$.

Figure 5.1 shows a side view of isosurfaces of the $Q$ field for three isovalues $Q_{iso} \in \{1, 10, 100\}$, Figure 5.2 shows the same isosurfaces on the suction side of the aerofoil. For an isovalue of $Q_{iso} = 1$ the full suction side and a major fraction of the wake are densely populated with an interconnected complex structure which has been described as *sponge-like* by [77]. Only on the downstream end of the wake individually resolved vortices or vortex clusters can be found. With an increase to $Q_{iso} = 10$ and $Q_{iso} = 100$ the resolved region moves further upstream. Upstream of this region remains a sponge-like structure while less and less features are detected further downstream as the intensity of the vortices declines. For $Q_{iso} = 1$ and $Q_{iso} = 10$ the region from the leading edge until turbulent transition is obstructed by an isosurface caused by strong shear. For $Q_{iso} = 100$ this area is not obstructed and an abrupt start of the sponge-like structure can be seen.

Figure 5.3 shows a side view of isosurfaces of the $R$ field for three isovalues $R_{iso} \in \{0.1, 0.4, 0.7\}$, Figure 5.4 shows the same isosurfaces on the suction side of the aerofoil. The full wake is visible for all three values of $R_{iso}$. Furthermore individual features are resolved from the area where the transition happens on the suction side of the aerofoil until the downstream end of the wake. An increase in $R_{iso}$ changes the size of the features, but not the overall picture of the flow. The area immediately downstream of the leading edge is not obstructed exhibiting roll-up and breakdown of spanwise vortices. Turbulent transition is not happening abruptly, but one can see the formation of strong vortices which further downstream grow in number while declining in intensity.

If vortices are extracted via a global $R_{iso}$ each vortex is identified via an individual threshold in $Q$. That way vortices are resolved in the full domain and phenomena such as roll up of vortices, transition of the flow from a laminar to a turbulent state and the full wake can be observed using a single threshold only. Changing the value of $R_{iso}$ only has a minor impact on the depiction of these phenomena. In contrast, extracting vortices using $Q_{iso}$ fails to illustrate all phenomena at once as the view on intense vortices is obstructed by artefacts and vortices of low intensities are not identified. The result has in addition a strong dependence on the user-defined value of $Q_{iso}$.

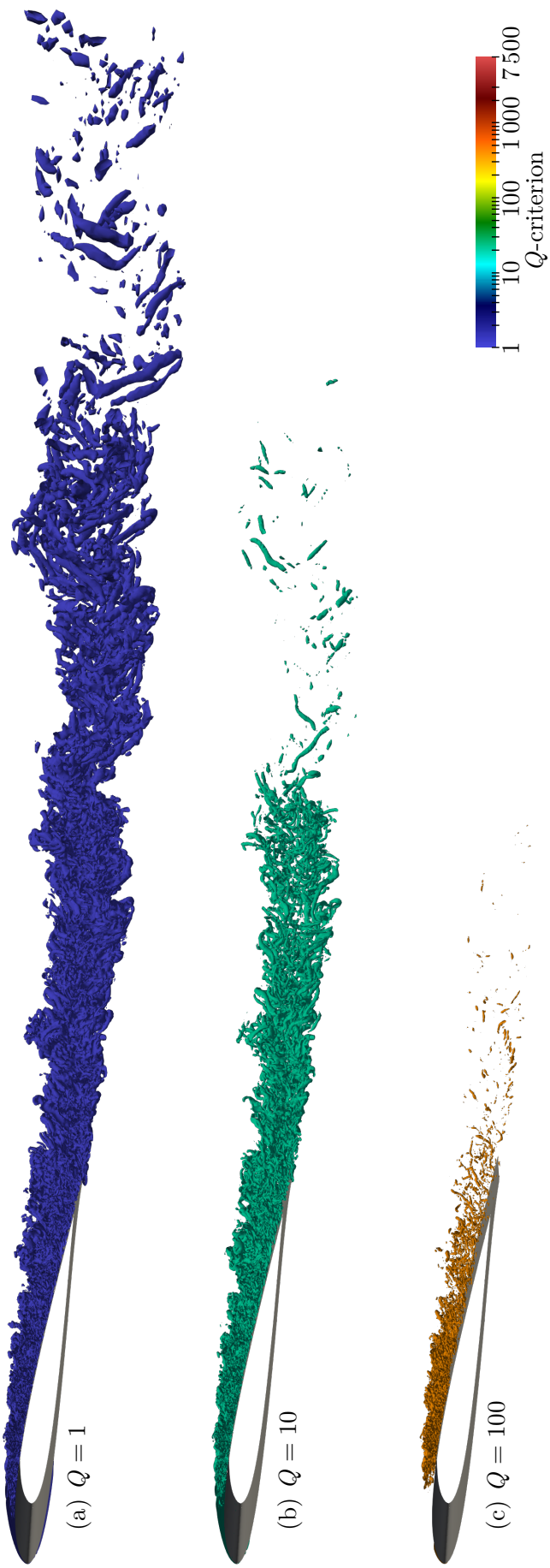



Figure 5.1: Side view of the SD7003 aerofoil and its wake including isosurfaces of (a) $Q_{iso} = 1$, (b) $Q_{iso} = 10$ and (c) $Q_{iso} = 100$ coloured by the $Q$-criterion.

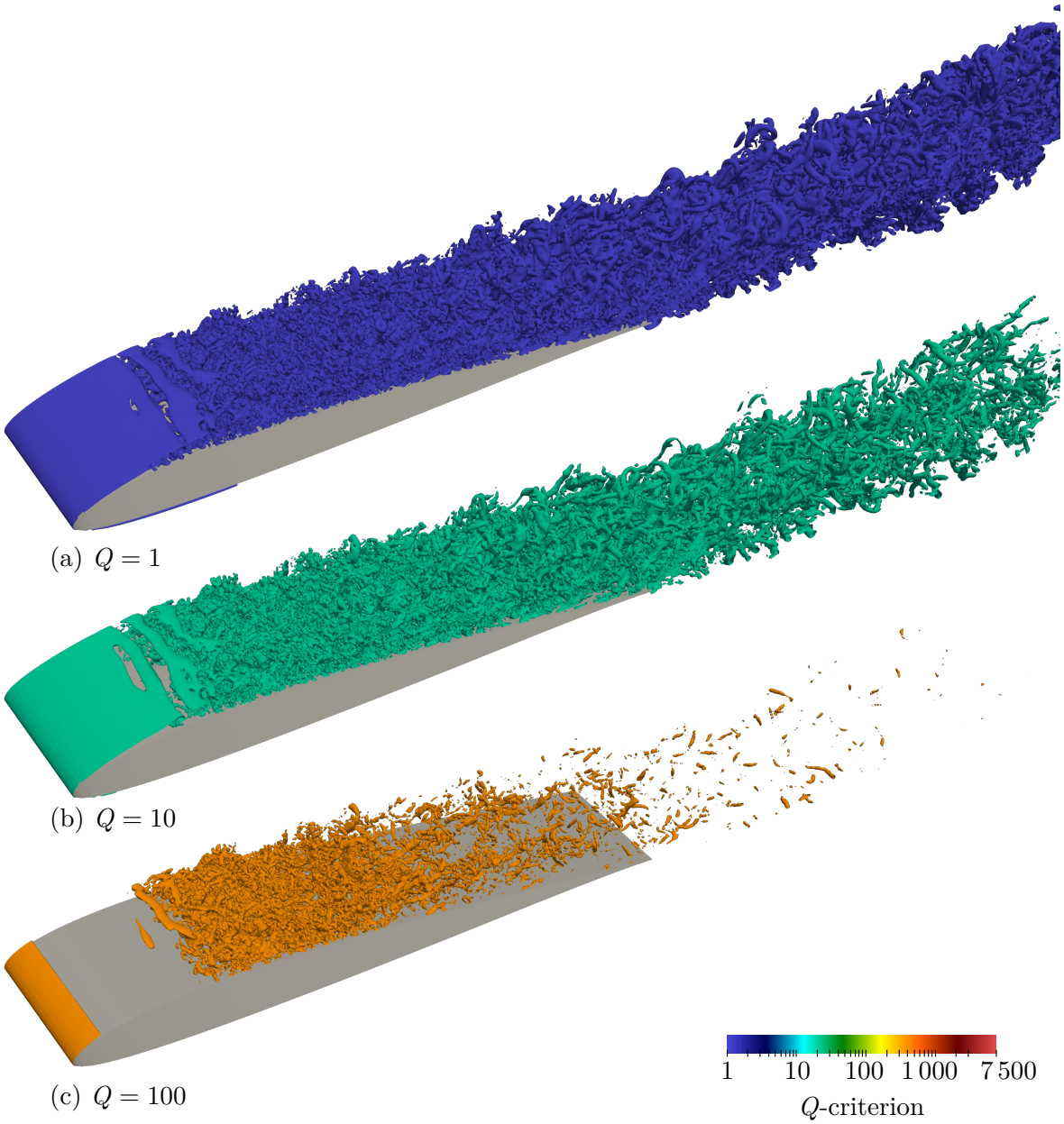



Figure 5.2: Isosurfaces of (a) $Q = 1$, (b) $Q = 10$ and (c) $Q = 100$ on the suction side of the SD7003 aerofoil coloured by the $Q$-criterion.

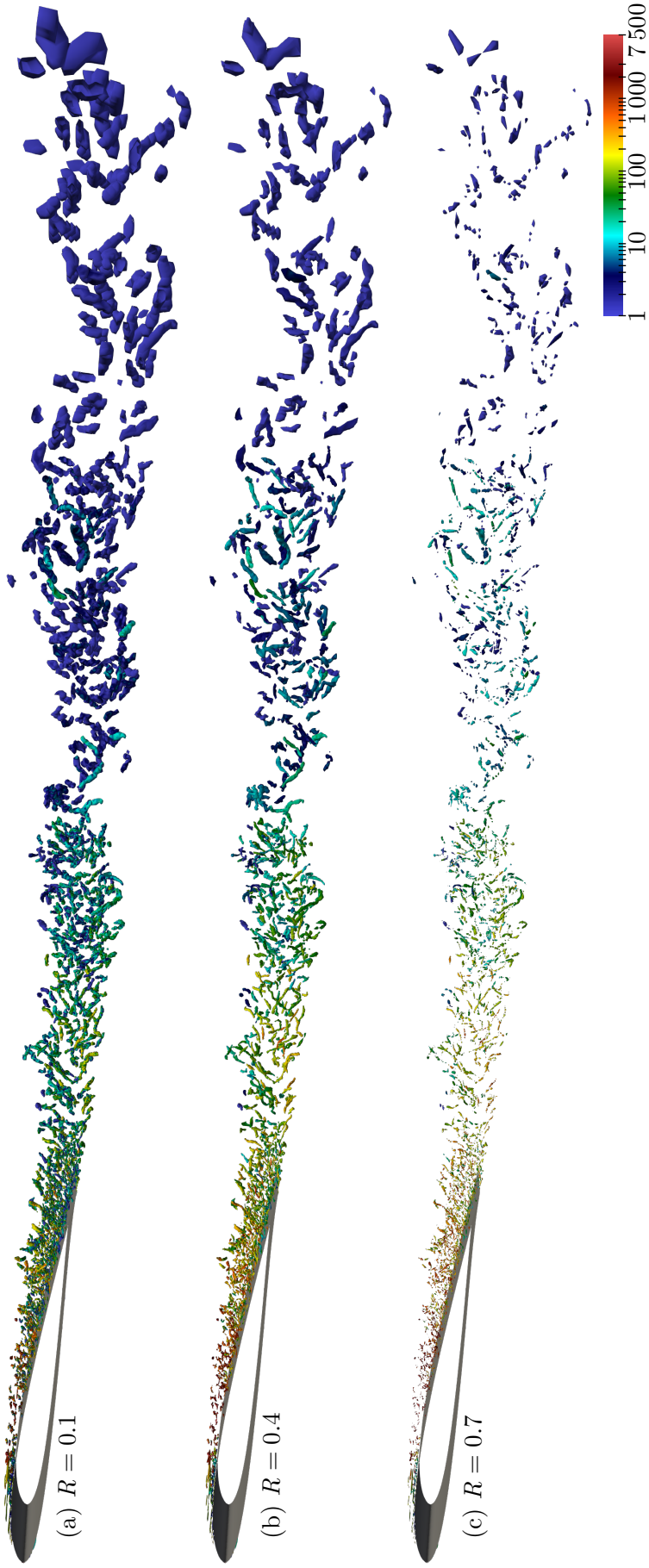
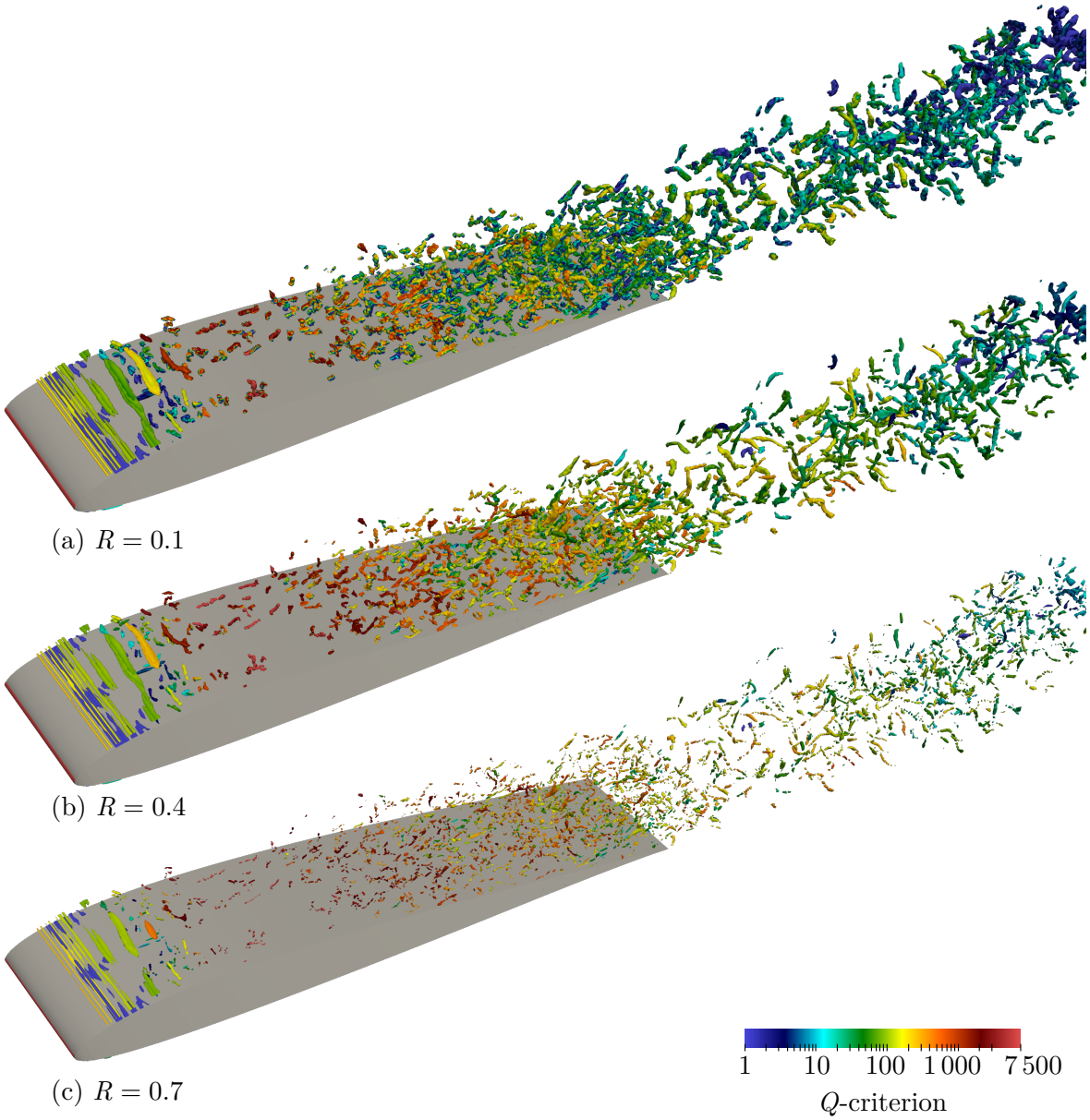



Figure 5.3: Side view of the SD7003 aerofoil and its wake including isosurfaces of (a) $R = 0.1$, (b) $R = 0.4$ and (c) $R = 0.7$ coloured by the $Q$-criterion.




(a) $R = 0.1$

(b) $R = 0.4$

(c) $R = 0.7$

Figure 5.4: Isosurfaces of (a) $R_{iso} = 0.1$, (b) $R_{iso} = 0.4$ and (c) $R_{iso} = 0.7$ on the suction side of the SD7003 aerofoil coloured by the $Q$-criterion.

# 6 Identifying and Counting Vortex Clusters in a Taylor-Green Vortex

To show its capabilities Tracer was applied in-situ to extract vortices of a Tayor-Green vortex (TGV) [97] at $Re = 1\,600$, the setup of which is provided in Section 6.1. The focus of the analysis lies on quantitatively highlighting the advantages of feature extraction via $R_{cut}$ over using a global threshold $Q_{cut}$. Section 6.2 examines the dependence of the number of extracted features on the threshold value using either $Q_{cut}$ or $R_{cut}$. It is furthermore explained how such an assessment helps finding an appropriate value for $R_{cut}$. Section 6.3 analyses the correlation between turbulent breakdown and decay and the cardinality of vortices which are extracted on the basis of $R_{cut}$. Note that in this chapter all renderings of isosurfaces of $R$ were made using Tracer's in-transit capability while tracer was running in-situ alongside PyFR.

## 6.1 Setup

### 6.1.1 Simulation Setup

The simulation of the TGV was conducted with the compressible Navier-Stokes solver of PyFR [107] using the setup by Witherden and Jameson [106] at $Re = 1\,600$ and an effectively incompressible Mach number of $Ma = 0.1$. A summary of the setup is provided in Table 6.1. The computational domain was a box with of size $2\pi^3$ with periodic boundaries in all directions discretised with a grid of $52^3$ hexahedra, in each of which the solution was approximated using a fourth-order polynomial.

Table 6.1: Setup of the Taylor-Green Vortex simulation [106] from which the $Q$-criterion field was obtained.

| | |
|---|---|
| solver: | PyFR |
| solution basis order: | 4 |
| mesh: | $52^3$ hexahedra |
| Reynolds number: | 1 600 |
| Mach number: | 0.1 |
| $t_{\text{start}}$ | 0 |
| $t_{\text{end}}$ | 100 |
| time integration | four-stage Runge-Kutta |

### 6.1.2 Setup for the Topological Analysis

Tracer was employed in-situ every $\Delta t = 0.02$ to extract vortices from the $Q$-criterion field of the TGV as the flow field was produced by PyFR. For the topological analysis each high-order element of the computational mesh was subdivided into $6^3$ hexhedral cells with trilinear interpolation, resulting in a grid of $312^3$ cells. The level of subdivision was chosen following the reasoning of Jallepalli et al. [52] which was described in Section 4.2.1. In order to filter noise extracted vortices were required to contain at least 8, 27 or 64 vertices. Aiming at maximising the number of extracted features, the relevance threshold was set to $R_{cut} = 0.4$ based on the results from Section 6.2. A summary of the setup can be taken from Table 6.2.

## 6.2 Counting Features via a Global Threshold vs Counting Vortex Clusters via Relevance

A single snapshot of the TGV flow field at $t = 15$ was processed by Tracer to investigate the percolation behaviour of identified features on the chosen threshold. Figure 6.1 shows isosurfaces of $Q$ and $R$ of the flow field. Figure 6.2 provides the number of identified features $n_Q$ extracted via $Q_{cut}$ as a function of $\frac{Q_{cut}}{Q_{max}}$ and the number of identified features $n_R$ extracted via $R_{cut}$ as a function of $R_{cut}$. Features of all sizes were counted to produce the graphs in Figure 6.2 (a), for the graphs in Figure 6.2 (b) features were required to contain at least 8, 27 or 64 vertices. Without filter $n_R$ increases monotonically with $R_{cut}$. The rising threshold causes the features to split up until at $R_{cut} = 1$ all 81 240 maxima of the $Q$-criterion field are identified as individual features. $n_Q$ has a strong dependence on the chosen threshold. It reaches a sharp peak at $\frac{Q_{cut}}{Q_{max}} \approx 0.025$, which is an order of magnitude below $n_R$. After the peak $n_Q$ declines steeply.

Filtering small features by setting *mfs* to a positive value changes the course of $n_R$ as depicted in Figure 6.2 (b). After an initial ascent $n_R$ reaches a peak around which there is only a weak dependence on $R_{cut}$. At $R_{cut} = 1$ all features contain a single vertex only and are therefore filtered resulting in $n_R = 0$. With increasing *mfs* the height of the peak decreases and moves towards lower values of $R_{cut}$. Changing *mfs* for vortex extraction via $Q_{cut}$ scales the graph of $n_Q$, but does not change its overall behaviour. The number of identified features stays well below the numbers identified with $R_{cut}$. In addition there remains a strong dependence of $n_Q$ on $Q_{cut}$. Figure 6.2 (c) shows the volume $V_{max}$ occupied by the largest feature normalised with the volume $V_{tot}$ of all features extracted with that threshold value. $Q_{cut}$ exhibits again a strong

Table 6.2: Setup of Tracer for analysing the TGV.

| | |
|---|---|
| linear subdivisions : | 6 |
| mesh: | $312^3$ hexahedra |
| time between snapshots $\Delta t$: | 0.02 |
| analysed scalar field $f$: | $Q$-criterion |
| $f_{min}$: | 0 |
| feature detection threshold: | $Q_{cut}(t) = 0.086\,Q_{max}(t), R_{cut} = 0.4$ |
| minimal feature size *mfs*: | $8, 27, 64$ vertices |

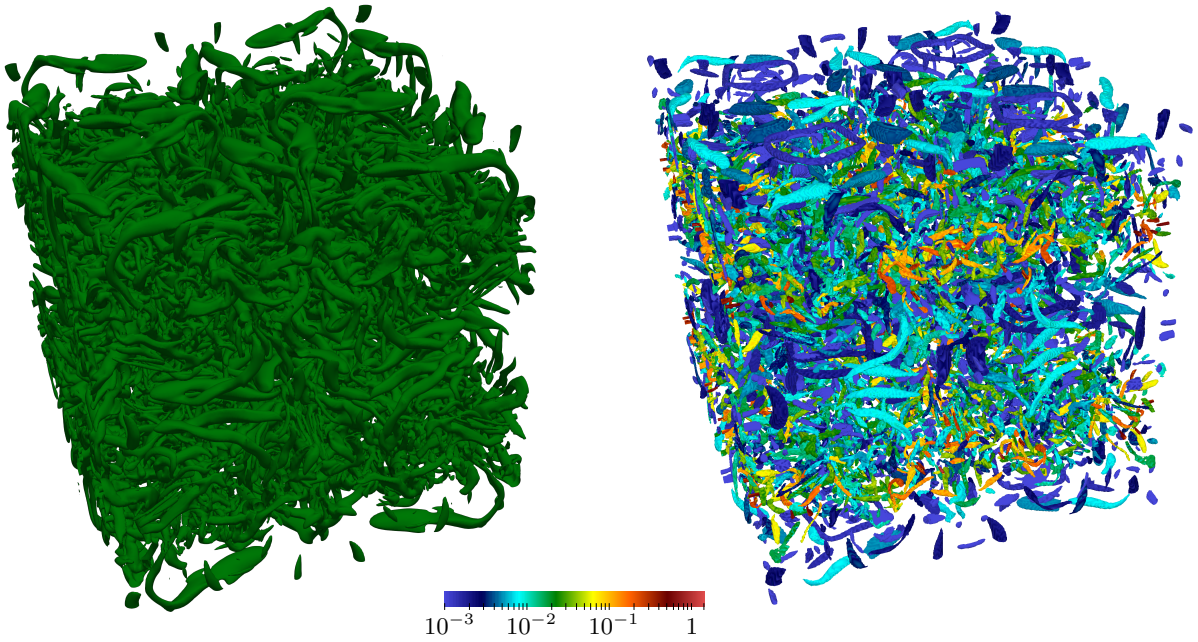



Figure 6.1: Snapshot of the TGV at $t = 15$ including isosurfaces of (a) $Q = 0.021 = 0.086\,Q_{max}$ and (b) $R = 0.4$ coloured by the $Q$-criterion.

dependence on the chosen value with a sharp minimum at $Q_{cut} = 0.086\,Q_{max}$, which is more than three times higher than the value of $Q_{cut}$ for which $n_Q$ maximises. The volume ratio drops to considerably lower values when features are identified via $R_{cut}$, meaning the total volume of features is less dominated by the largest features. Additionally the dependence on the value of $R_{cut}$ is comparably weak for $R_{cut} = [0.1, 0.8]$.

Based on these findings a threshold of $R_{cut} = 0.4$ is a good choice for identifying individual vortices, as it maximises $n_R$ and the total volume of all extracted features is not dominated by the largest features.

## 6.3 Turbulent Breakdown Reflected in Number of Vortices

This section investigates how the turbulent breakdown is reflected in the vortex cardinality. As Tracer was applied 5 001 times alongside the simulation, this investigation would have been virtually impossible without Tracer's capability extract vortices on the fly, as it would have required to store all 5 001 flow fields. Note that the analysis was done before periodic boundary conditions were implemented in Tracer.

Figure 6.3 (a) shows the number of vortices identfied by Tracer with a time dependend threshold $Q_{cut}(t) = 0.086\,Q_{max}(t)$ and a constant threshold of $R_{cut} = 0.4$. Vortices extracted with $Q_{cut}(t)$ were required to contain at least eight vertices, while for vortices extracted with $R_{cut}$ three different filters were applied: $mfs \in \{8, 27, 64\}$. Directly after the initialisation the $Q$-criterion field is extremely smooth. Due to the nature of the relevance criterion weak perturbations which are small in amplitude but can have a significant sizes are identified as features. Because of this the number of extracted features was not recorded before $t = 0.5$.

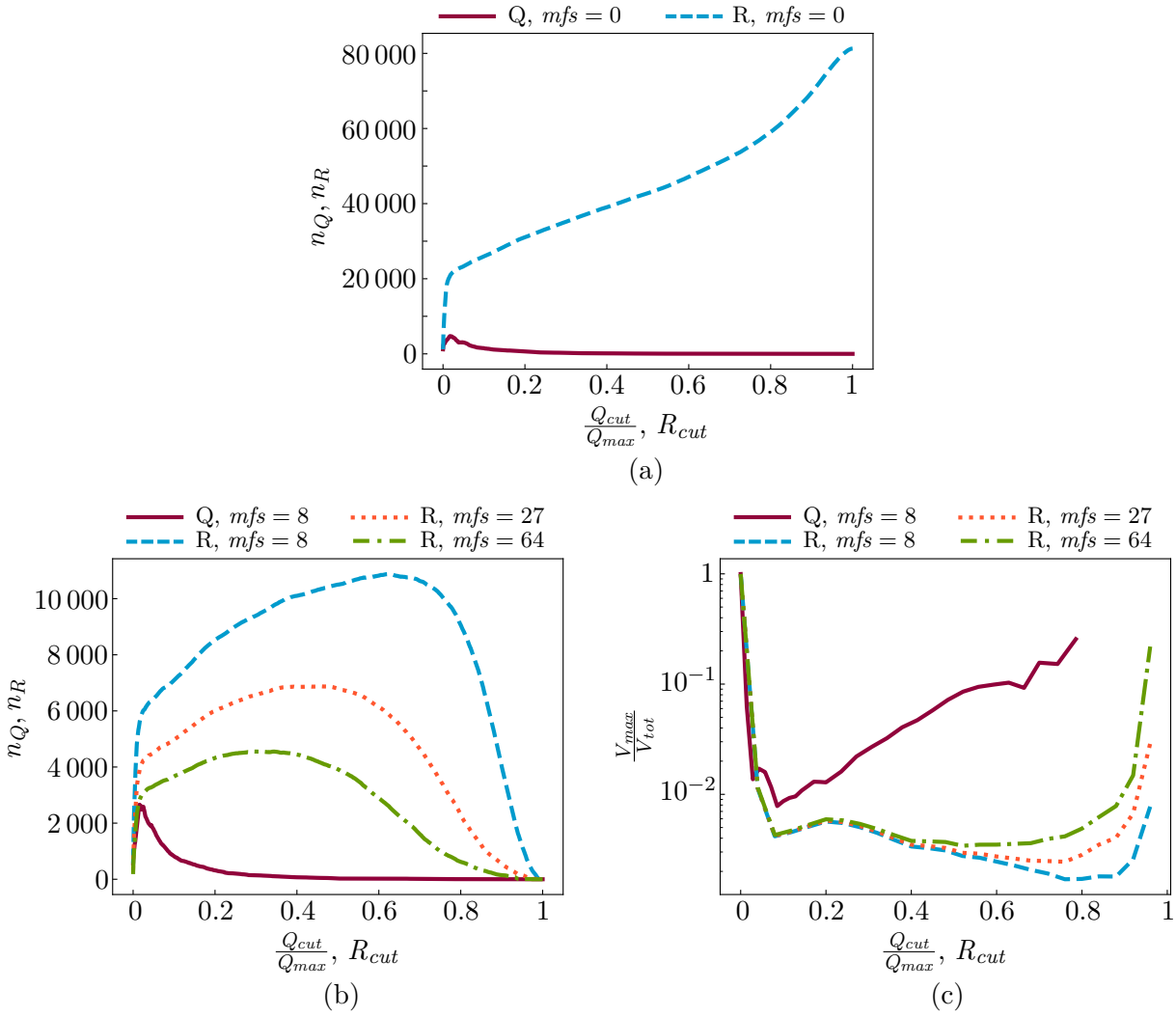



Figure 6.2: (a) $n_Q$ over $\frac{Q_{cut}}{Q_{max}}$ and $n_R$ over $R$ for $mfs = 0$ (b) $n_Q$ over $\frac{Q_{cut}}{Q_{max}}$ and $n_R$ over $R$ for $mfs \in \{8, 27, 64\}$ and (c) $\frac{V_{max}}{V_{tot}}$ over $\frac{Q_{cut}}{Q_{max}}$ or $R_{cut}$ for $mfs \in \{8, 27, 64\}$ of a snapshot of the TGV case at $t = 15$.

Figures 6.3 (b)-(f) show extracted vortices at given times as rendered isosurfaces of $R = 0.4$, which were produced in-transit by the implementation of Alpine/Ascent [64] in Tracer. The isosurfaces are coloured by the individual $Q$-thresholds of their vortices. In Figure 6.3 (b) one can find twelve features, four in three layers. The four features in each layer merge as can be seen in Figure 6.3 (c) forming three features which split up into 48 features in Figure 6.3 (d). The graphs in Figure 6.3 (a) which are associated with $R_{cut}$ are in agreement with the observations from the renderings. The detection via $Q_{cut}(t)$ however fails to identify those structures. In Figure 6.3 (e) at $t = 3.5$ two types of features can be seen, one being weaker than the other. While $R_{cut}$ detects all 96 features, $Q_{cut}(t)$ fails to properly resolve all flow features. Turbulent breakdown occurs around $3.5 \lesssim t \lesssim 5$ [13]. The number of features jumps for $R_{cut}$ and keeps increasing afterwards, as the flow transitions to a chaotic state which can be seen in Figure 6.3 (f). The exact time at which the jump happens depends on the filter size, but stays within the interval. If features are counted using $Q_{cut}(t)$ the turbulent breakdown is not reflected in the feature cardinality. Instead $n_Q(t)$ starts slowly increasing at $t \approx 4.5$ but drops back down at $t \approx 5.5$ For $t \gtrsim 5.5$ $n_Q$

gradually increases again while being dominated by oscillations.

Figure 6.4 shows the course of $n_Q(t)$ and $n_R(t)$ until $t = 100$ with a linear vertical axis. The graph of $n_Q(t)$ is dominated by oscillation and bears little information only. The course of $n_R(t)$ however reflects the turbulent transition with a steep increase in the beginning. After reaching a peak at $t \approx 15$ $n_R(t)$ declines as the turbulence decays.

## 6.4 Conclusion

Feature extraction via $R_{cut}$ detects vortices over the full spectrum of intesity. $n_R$ is generally larger than $n_Q$, as a global $Q_{cut}$ is at the same time too high to detect weak features and low enough so that intense features are amalgamated. Small features have to be filtered as they otherwise account for the majority of identified features. Plotting $n_R$ and $\frac{V_{max}}{V_{tot}}$ as a function of $R_{cut}$ one can observe a comparably flat maximum $n_R$ at which $\frac{V_{max}}{V_{tot}}$ takes a stable low value. The maximum of $n_Q$ and the minimum of $\frac{V_{max}}{V_{tot}}$ are both sharp extrema, meaning that there is a strong dependence on $Q_{cut}$. Furthemore is the location of the minimum of $\frac{V_{max}}{V_{tot}}$ at a $Q_{cut}$ more than three times higher than the $Q_{cut}$ for the maximum of $n_Q$. Counting features which were extracted with a constant $R_{cut}$ one can observe individual vortices merge and split in the start-up phase of the TGV, their increase during turbulent breakdown at a time which is consistent with [13] and decrease during the decay of turbulence. A time dependent $Q_{cut}$, however, fails to deliver comparable results over the full simulation time.

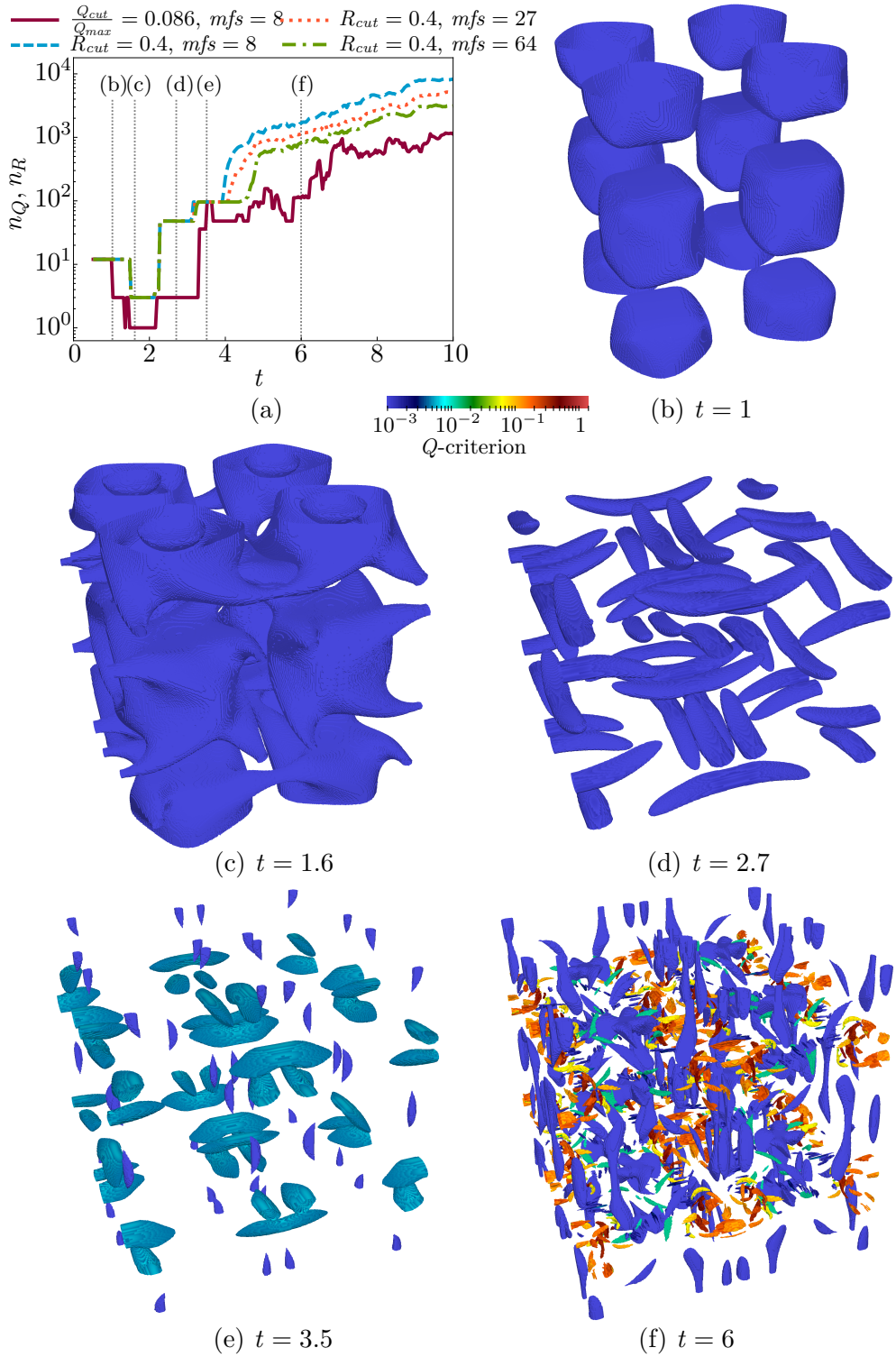



Figure 6.3: (a) Number of identified flow features over time using either a time-dependent $Q_{cut}$ and $mfs = 8$ or a constant $R_{cut}$ and $mfs \in \{8, 27, 64\}$. (b)-(f) Isosurface sets for $R = 0.4$ of snapshots of the TGV at various times. The number of identified features matches up with those visible. Note that the topological analysis was conducted with version of Tracer which did not include periodic boundary conditions. The renderings of isosurfaces of $R$ were produced with Tracer's in-transit capability of generating png files via Alpine/Ascent [64].

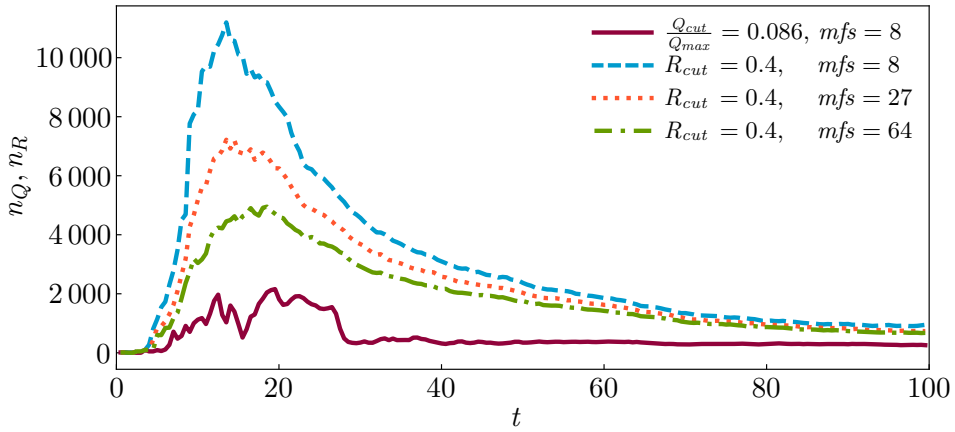



Figure 6.4: Number of identified flow features over time using either a time-dependent $Q_{cut}$ and $mfs = 8$ or a constant $R_{cut}$ and $mfs \in \{8, 27, 64\}$.

# 7 Analysing the Topology and the Geometry of Vortices in a Channel Flow

The attached eddy hypothesis (AEH) by Townsend [101, 102] has advanced the understanding and wall-bounded turbulence and the modelling thereof via the attached eddy model [72]. The underlying assumption of the AEH is that wall-bounded turbulent flows are governed by self-similar wall-attached eddies, the geometry of which scales with the distance of their centre from the wall. With increasing availability of high-fidelity data of turbulent flow fields from experiments and CFD the predictions of the AEH have been confirmed [28, 48, 65, 75, 92] and the model has been refined. While there have been some studies on individual features close to the wall [53], the majority of self-similarity was found in clusters [6, 29, 47]. The criteria that were used to extract clusters using a threshold that depends on the distance from the wall. This is an improvement to setting a global threshold enabling the detection of the most intense clusters [29]. These identification methods however lack the ability to extract individual features over the full scale of intensity.

Tracer's capability of extracting individual features over the whole range of the domain facilitates the analysis of their topology and geometry. In Section 7.1 Tracer was applied to a channel flow at $Re_\tau = 180$ to identify vortices and asses similarity in their topology. In Section 7.2 the geometrical self-similarity of individual vortices in a channel flow at $Re_\tau = 550$ was investigated. $Re_\tau$ is the friction Reynolds number based on the channel half-height $\delta$ and the friction velocity

$$u_\tau = \sqrt{\frac{\tau_w}{\rho}}. \tag{7.1}$$

Throughout this chapter normalisation in wall units is achieved via $u_\tau$ and the kinematic viscosity $\nu$ and indicated by the superscript "+". Specifically length scales like the wall-distance are normalised by:

$$y^+ = \frac{u_\tau}{\nu} y, \tag{7.2}$$

and time scales by

$$t^+ = \frac{u_\tau^2}{\nu} t. \tag{7.3}$$

## 7.1 Topological Similarity of Vortices in near-wall and Central Region of a Channel

In a channel flow one can find quasi-streamwise vortices close to the wall [53], which become more and more unorganised towards the channel centre. This poses the question if this is reflected in the topology of the vortices. To answer this question the vortices were assigned to either the

near-wall region or the channel centre and the topological parameters of vortices in both regions were compared. Sections 7.1.1 and 7.1.2 present the setup of the simulation of the channel flow at $Re_\tau = 180$ and the setup for the feature extraction respectively. In Section 7.1.3 the topological complexity of vortices in both regions is compared on the basis of the Horton number of their root $r_{max}$. A comparison of the topological organisation of the vortices in near-wall and central region parametrised by the HS ratios is provided in Section 7.1.4. The Section finishes with a summary of the findings.

### 7.1.1 Simulation of a Turbulent Channel Flow at $Re_\tau = 180$

#### Setup

The topological similarity of vortices was analysed using DNS data of a turbulent channel flow conducted by Iyer et al. [50]. A summary of the simulation setup is provided in Table 7.1. The computational domain was a box with of size $8\pi \times 2 \times 4\pi$ as depicted in Figure 7.1. The $x$-, $y$- and $z$-directions were chosen to be streamwise, wall-normal and spanwise directions respectively. Periodic boundary conditions were applied in the $x$- and $z$-directions and adiabatic no-slip conditions in the $y$-direction. The Reynolds number based on the bulk velocity and the channel half-height $\delta$ is $Re = 2\,767$, the friction Reynolds number is $Re_\tau = 180$ and the Mach number is $Ma = 0.1$.

Table 7.1: Setup of the channel flow simulation at $Re_\tau = 180$ by Iyer et al. [50].

| | |
|---|---|
| solver: | PyFR |
| solution basis order: | 4 |
| domain size $\lambda_x \times 2\delta \times \lambda_z$: | $8\pi \times 2 \times 4\pi$ |
| mesh: | $62 \times 19 \times 60$ hexahedra |
| friction Reynolds number $Re_\tau$: | 180 |
| bulk Reynolds number $Re$: | 2 767 |
| Mach number: | 0.1 |

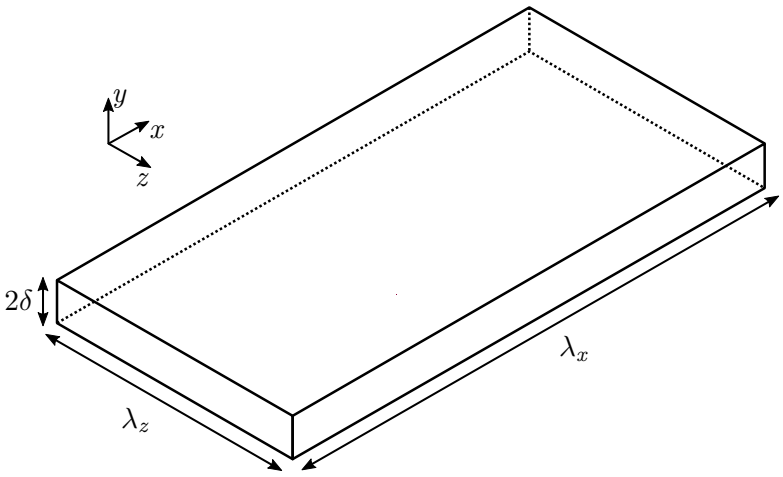



Figure 7.1: Domain of the channel flow.

#### Mesh

The domain was discretised on a structured mesh consisting of $62 \times 19 \times 60$ hexahedral high-order elements, in each of which the solution was represented with a fourth order polynomial. The mesh had uniform spacing in the $x$- and $z$-directions, while the $y$-direction the resolution was increasing towards the wall.

### 7.1.2 Setup for the Topological Analysis

Tracer was employed to extract vortices from the $Q$-criterion field of the channel flow. For the topological analysis each high-order element of the computational mesh was subdivided into $6 \times 6 \times 6$ hexhedral cells with trilinear interpolation, resulting in a grid of $372 \times 114 \times 360$ cells. The level of subdivision was chosen following the reasoning of Jallepalli et al. [52] which was described in Section 4.2.1. In order to filter noise extracted vortices were required to contain at least 27 vertices. $R_{cut}$ was chosen via a percolation analysis of a single snapshot of the $Q$-criterion field. Figure 7.2 provides the number of identified clusters and the volume of the largest vortex $V_{max}$ normalised with the total volume $V_{tot}$ of all vortices as a function of $R_{cut}$. For the extraction of individual vortices the number of identified features shall be maximised and $\frac{V_{max}}{V_{tot}}$ shall be very small and remain constant. Based on these criteria the relevance threshold was set to $R_{cut} = 0.4$. In addition features were extracted with $R_{cut} = 0.1$. While the number of identified features is only slightly smaller compared to $R_{cut} = 0.4$, $\frac{V_{max}}{V_{tot}}$ is three times larger, indicating that in areas densely populated with vortices individual features have merged to clusters. Below $R_{cut} = 0.1$ the volume ratio is ascending steeply, as the vortices merge to very large features. Renderings of isosurfaces of $R = 0.1$ are provided in Figure 7.3, renderings of isosurfaces of $R = 0.4$ are provided in Figure 7.4. A total of 87 snapshots, which are $\Delta t^+ = 585.5$ apart, were processed resulting in the identification of 755 116 vortices for $R_{cut} = 0.1$ and of 910 132 vortices for $R_{cut} = 0.4$. A summary of the setup can be taken from Table 7.2.

The channel was split up in a near-wall region which extends up to 90 wall units away from the wall and a central region, as depicted in Figure 7.3 (a). The vortices were assigned to the region in which their centre was located in.

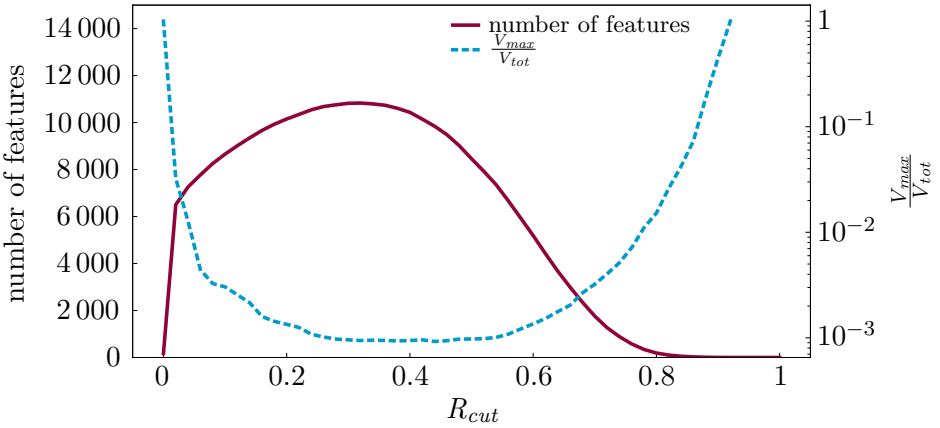



Figure 7.2: Percolation behaviour of the flow features in the channel at $Re_\tau = 180$.

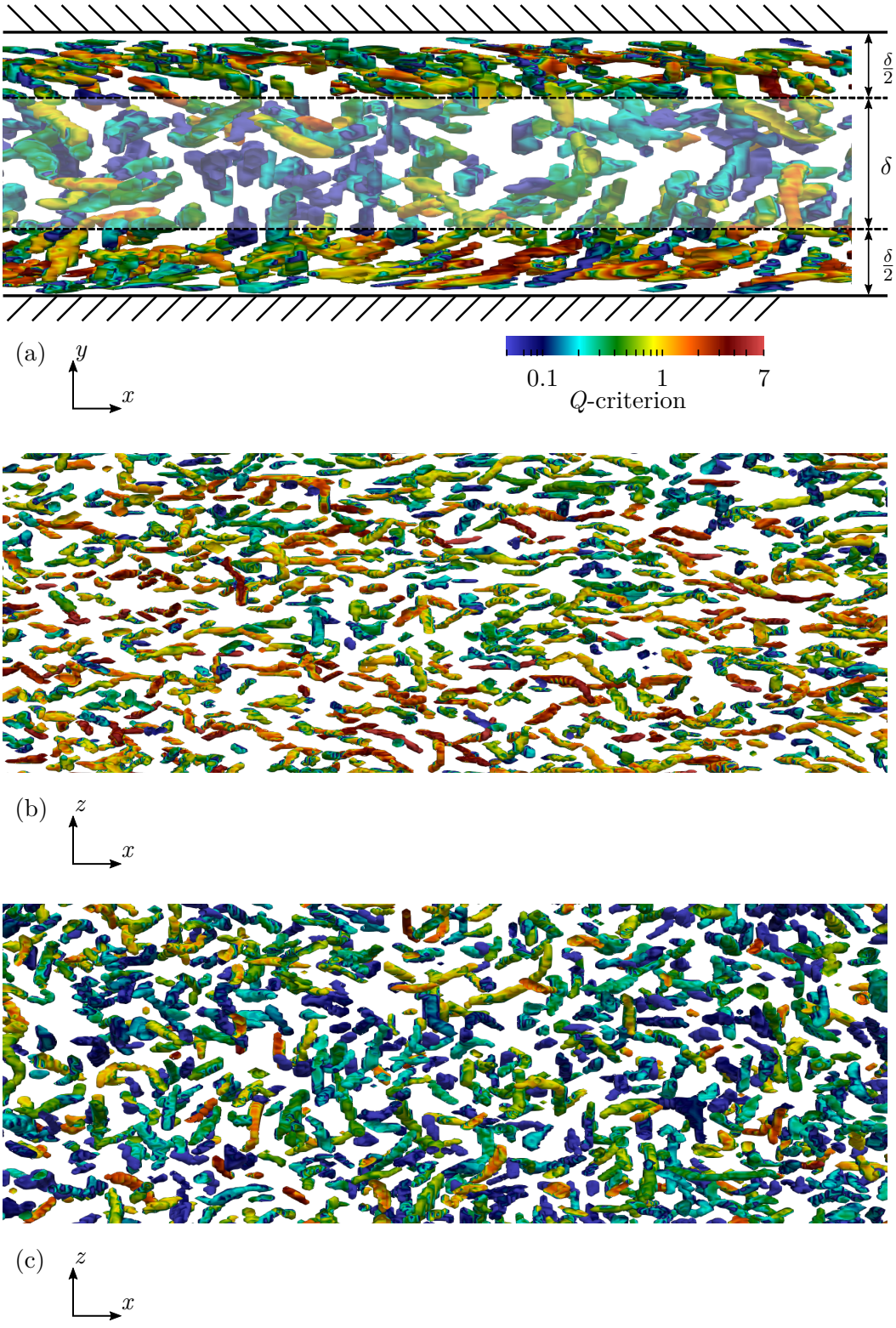



Figure 7.3: Isosurfaces of $R = 0.1$ coloured by $Q$ in a snapshot of the channel flow at $Re_\tau = 180$. (a) shows the side view and the partition into near-wall and central regions. Bottom view of (b) the near wall region and (c) the central region. To remove visual clutter $R$ was set to $-\varepsilon$ in features the sizes of which were below $mfs = 27$. A change in colour of the isosurface of a given feature is caused by discontinuities in $R$ and interpolation of the visualisation software.

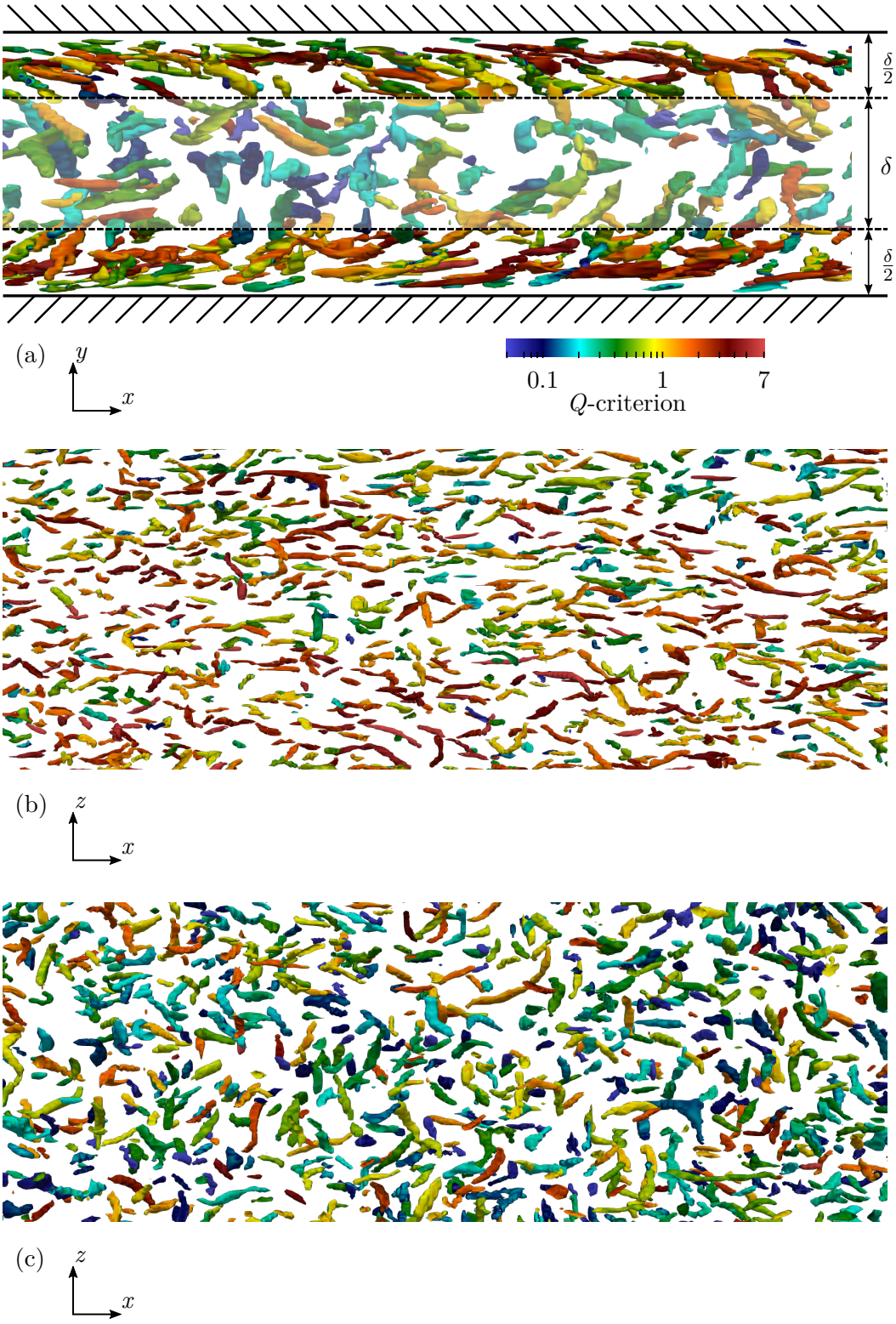



Figure 7.4: Isosurfaces of $R = 0.4$ coloured by $Q$ in a snapshot of the channel flow at $Re_\tau = 180$. (a) shows the side view and the partition into near-wall and central regions. Bottom view of (b) the near wall region and (c) the central region. To remove visual clutter $R$ was set to $-\varepsilon$ in features the sizes of which were below $mfs = 27$. A change in colour of the isosurface of a given feature is caused by discontinuities in $R$ and interpolation of the visualisation software.

Table 7.2: Setup of Tracer for identifying features in the channel flow at $Re_\tau = 180$.

| | |
|---|---|
| linear subdivisions : | 6 |
| mesh: | $372 \times 114 \times 360$ hexahedra |
| number of snapshots: | 87 |
| time between snapshots $\Delta t^+$: | 585.5 |
| analysed scalar field $f$: | $Q$-criterion |
| $f_{min}$: | 0 |
| feature detection threshold $R_{cut}$: | $0.1, 0.4$ |
| minimal feature size $mfs$: | 27 vertices |
| near-wall region: | $y^+ \leq 90$ |
| central region: | $y^+ > 90$ |

### 7.1.3 Distribution of Horton Numbers

Figure 7.5 shows histograms of distribution in wall-normal direction of vortices of a given $r_{max}$ extracted with $R_{cut} = 0.1$. The histograms are normalised by the height of their highest bar. For $r_{max} \geq 1$ all histograms have a peak at each of the two walls at wall and a trough in channel centre. The histogram for $r_{max} = 0$ includes an additional local maximum in the channel centre, which is a lot lower than the peaks close to the wall. With increasing $r_{max}$ the peaks move away from the wall. Topologically more complex features are more likely to occupy a larger volume, resulting in their centre being further away from the wall. The numbers of vortices of a given $r_{max}$ split up by near-wall and central region are provided in Table 7.3. Vortices of $r_{max} = 1$ and $r_{max} = 2$ account for 87.25 % of all identified vortices. The ratio of number of vortices in the near-wall region over the number of vortices in the channel centre is increasing with $r_{max}$.

For $R_{cut} = 0.4$ Figure 7.6 shows normalized histograms of distribution in wall-normal direction of vortices of a given $r_{max}$, the numbers of vortices of a given $r_{max}$ split up by near-wall and central region are provided in Table 7.3. For $r_{max} = 1$ to $r_{max} = 3$ the histograms have two peaks close to the wall while the histogram for $r_{max} = 0$ has an additional smaller peak in the channel

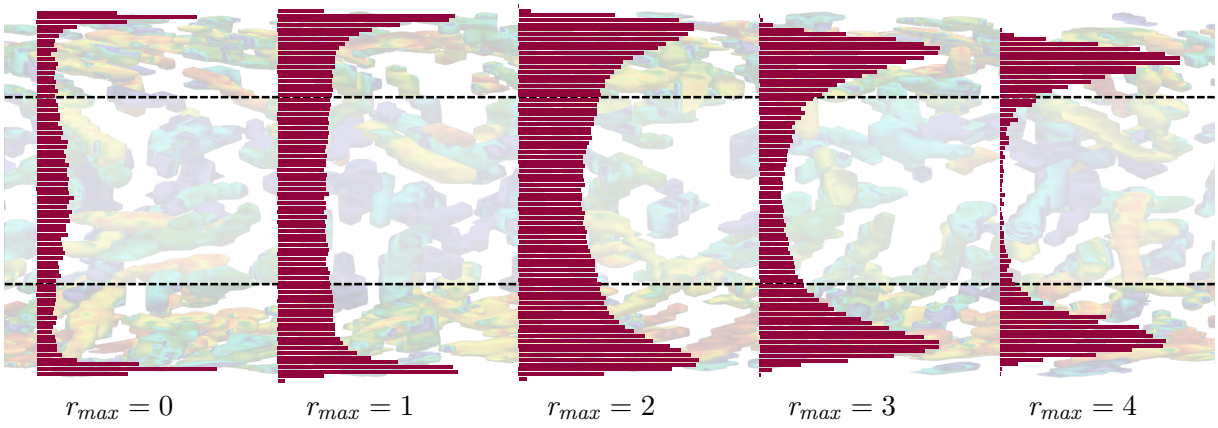
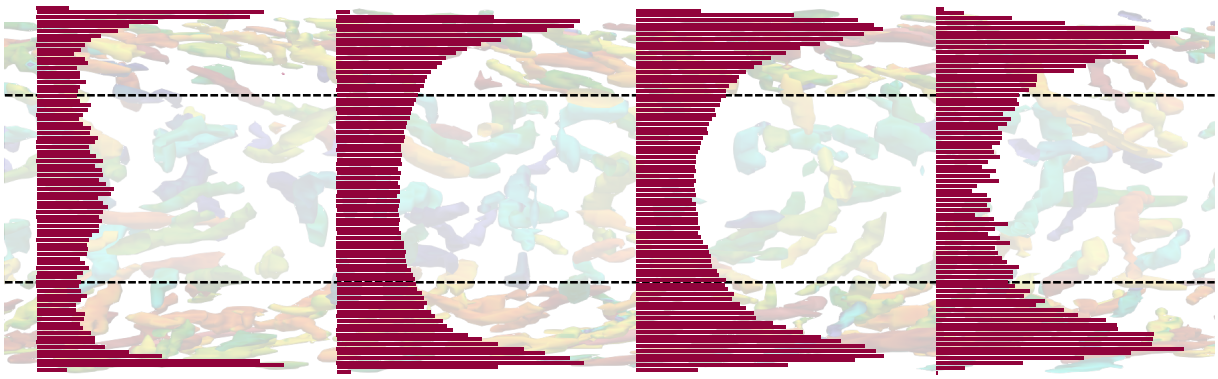



Figure 7.5: Normalised histograms of the $y$ component of the centre of vortices of a given topological complexity at $R_{cut} = 0.1$. The histogram are normalised so that the highest bar of each histogram has the same height.

Table 7.3: Number of extracted vortices of a given topological complexity $r_{max}$ split up by near-wall and central region for $R_{cut} = 0.1$ and $R_{cut} = 0.4$.

| | $R_{cut} = 0.1$ | | | $R_{cut} = 0.4$ | | |
|---|---|---|---|---|---|---|
| $r_{max}$ | # near-wall | # central | $\frac{\#\text{ near-wall}}{\#\text{ centre}}$ | # near-wall | # central | $\frac{\#\text{ near-wall}}{\#\text{ centre}}$ |
| 0 | 11 614 | 9 237 | 1.2573 | 15 531 | 12 454 | 1.2471 |
| 1 | 212 662 | 142 044 | 1.4972 | 357 636 | 193 832 | 1.8451 |
| 2 | 187 267 | 116 928 | 1.6016 | 215 280 | 105 707 | 2.0366 |
| 3 | 52 338 | 17 406 | 3.0069 | 6 511 | 3 128 | 2.0815 |
| 4 | 5 117 | 470 | 10.887 | 2 | 1 | — |
| 5 | 33 | — | — | — | — | — |



$r_{max} = 0$ $r_{max} = 1$ $r_{max} = 2$ $r_{max} = 3$

Figure 7.6: Normalised histograms of the $y$ component of the centre of vortices of a given topological complexity at $R_{cut} = 0.4$. The histogram are normalised so that the highest bar of each histogram has the same height.

centre. With increasing $r_{max}$ the peaks move away from the wall. However for $R_{cut} = 0.4$ this effect is smaller than for $R_{cut} = 0.1$. Vortices of $r_{max} = 1$ and $r_{max} = 2$ account for 95.87 % of all identified vortices. The ratio of number of vortices in the near-wall region over the number of vortices in the channel centre is increasing with $r_{max}$, with a minimal increase from $r_{max} = 2$ to $r_{max} = 3$.

### 7.1.4 Distribution of HS-Ratios in Vortices

The topological organisation of a feature is parameterised by the HS ratios, which were presented in Section 2.2.1. As seen in Section 7.1.3 the numbers of and the ratios of those numbers between near-wall and central region varies significantly for different $r_{max}$. To avoid measuring the differences in the topological organisation of vortices of different $r_{max}$, the analysis was done for all sets of vortices of a given $r_{max}$ separately.

Figures 7.7 (a), (c), (e) show histograms of the means of HS ratios of vortices of $r_{max} = 3$ extracted with $R_{cut} = 0.1$. The means were built within each vortex using Algorithm 4.21. The dark red bars indicate the numbers for the central region, the blue bars for the near-wall region. Scaling the dark red bars such that their total length is identical to the total length of the blue bars results in the bright red bars. The mean of all three HS ratios of vortices in

the near-wall region is larger those of vortices in the channel centre. Figures 7.7 (b), (d), (f) show histograms of the means of HS ratios of vortices of $r_{max} = 3$ extracted with $R_{cut} = 0.4$. The means were built within each vortex using Algorithm 4.21. Unlike for vortices extracted with $R_{cut} = 0.1$, the differences of the distribution for near-wall and the scaled distribution of the central region are marginal. Different means for $R_{cut} = 0.1$ and almost equal means for $R_{cut} = 0.4$ as shown on the example of vortices with $r_{max} = 3$ was the case for all sets of vortices of a given $r_{max}$. The HS ratios of trees of comparably small orders can be infuenced by effects of the root branch. Hence the analysis was repeated without taking statistics related to the root branches into account. While this changes the absolute numbers of the HS ratios, the relation between the regions remained the same. Figure 7.8 shows averages of the three HS ratios taken over all vortices of a given $r_{max}$ with and without statistics related to roots over $r_{max}$ for $R_{cut} = 0.1$ and $R_{cut} = 0.4$. While for $R_{cut} = 0.1$ the means of the vortices in near-wall region are consistently higher than the means of the central region, there is hardly a difference in the means for vortices with $r_{max} \geq 2$ extracted with $R_{cut} = 0.4$. Only for vortices of $r_{max} = 1$ the means of vortices in the near-wall region is clearly larger than the mean of the vortices in the central region.

### 7.1.5 Conclusion

The near wall region of a channel flow at $Re_\tau = 180$ is populated more densely with vortices than the central region. This trend increases with the topological complexity of the vortices. While the topological organisation of individual vortices extracted via $R_{cut} = 0.4$ in the near-wall and in the central region are indistinguishable, there is a difference when vortex clusters are extracted with $R_{cut} = 0.1$: the means of all HS ratios in the near-wall region is higher than the mean of vortices in the central region. A higher number in any of these ratios generally means an increased difference from the minimal tree of a given $r_{max}$. This could mean that in the near-wall region individual vortices of higher $r_{max}$ are surrounded by and merge with more vortices of smaller $r_{max}$ than in the central region. An explanation which is in agreement with the near-wall region being more densely populated with features than the central region.

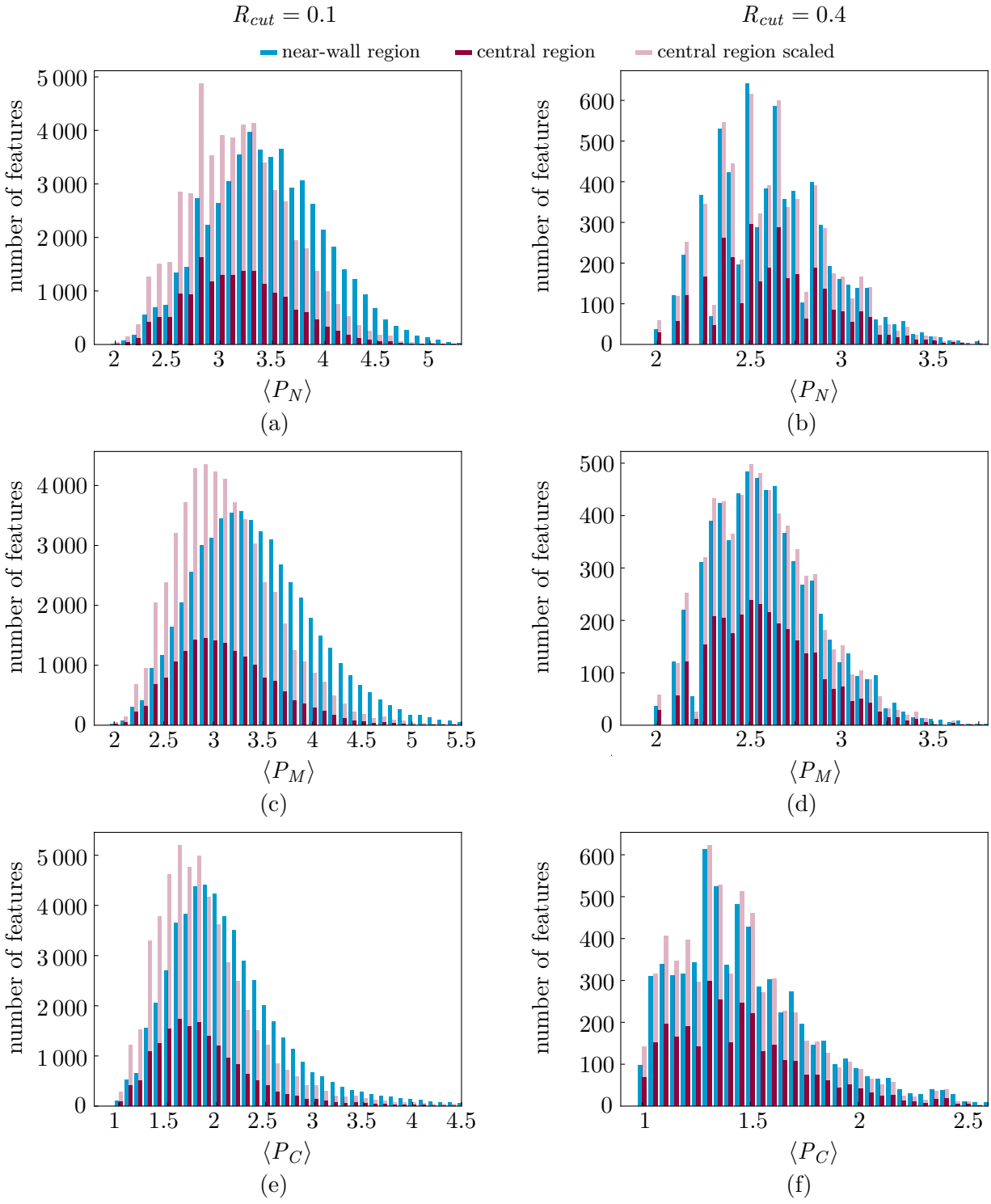



Figure 7.7: Histograms of HS ratios (a), (b) $\langle P_N \rangle$, (c), (d) $\langle P_M \rangle$ and (e), (f) $\langle P_C \rangle$ of vortices of $r_{max} = 3$ extracted with (a), (c), (e) $R_{cut} = 0.1$ and (b), (d), (f) $R_{cut} = 0.4$. The blue histograms are associated with vortices in the near-wall region, the red bars are associated with vortices in the central region. For the pale red bars the histograms of the central region were scaled so the total length of all bars is identical to the total length of all blue bars.

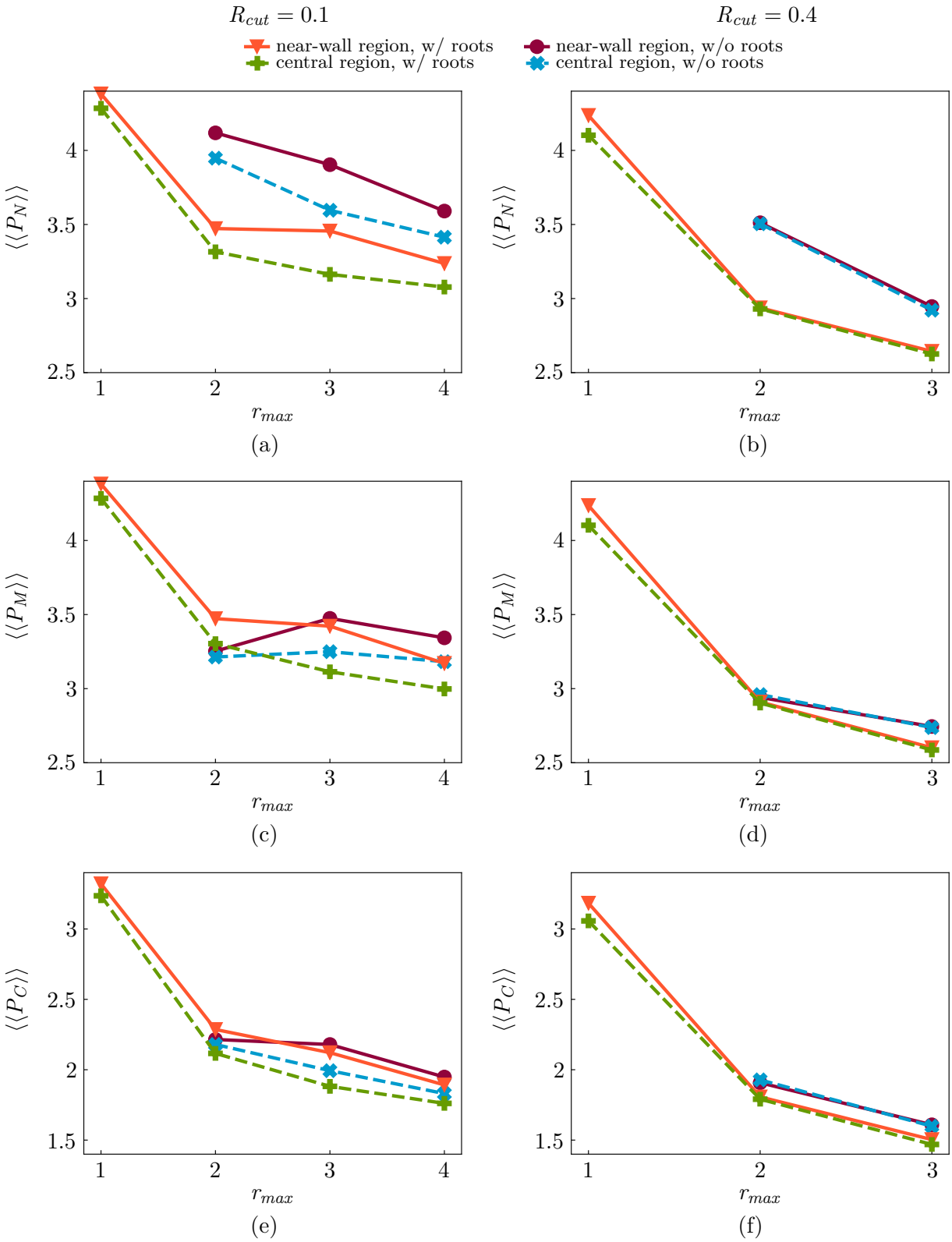



Figure 7.8: Averages HS ratios taken over all vortices of a given $r_{max}$ in the near-wall region and taken over all vortices of a given $r_{max}$ in the central region of the channel as a function of their topological complexity $r_{max}$ with and without taking statistics related to the root arc into account. The graphs in (a) and (b) show $\langle\langle P_N \rangle\rangle$, the graphs in (c) and (d) show $\langle\langle P_M \rangle\rangle$ and the graphs in (e) and (f) show $\langle\langle P_C \rangle\rangle$. The graphs in (a), (c) and (d) are associated with vortices extracted with $R_{cut} = 0.1$, the graphs in (b), (d) and (f) are associated with vortices extracted with $R_{cut} = 0.4$.

## 7.2 Assessing Geometrical Self-Similarity of Vortices in a Channel Flow of $Re_\tau = 550$

del Álamo et al. [29] have investigated self-similarity vortex clusters of the logarithmic region of a channel flows at a regime up to $Re_\tau = 1\,900$. The vortex clusters were extracted with the discriminant criterion [27] using a threshold which is a function of the wall distance. Such a varying threshold is an improvement to a global threshold, however as the authors point out it extracts only the most intense clusters. Analysing the volume distribution of the extracted clusters in a channel at $Re_\tau = 550$ as a function of their minimum $y^+_{min}$ and maximum $y^+_{max}$ wall distances, del Álamo et al. found that the vortex clusters can be grouped into attached and detached clusters. While the detached clusters were isotropic, del Álamo et al. detected self-similarity in attached clusters. In this section individual vortices of all intensities in a turbulent channel flow at $Re_\tau = 550$ were extracted by Tracer and their geometry was investigated for self-similarity. Section 7.2.1 provides the setup for the simulation to produce the channel flow fields, Section 7.2.2 provides the setup for the vortex extraction. The results are presented in Section 7.2.3.

### 7.2.1 Simulation of a Turbulent Channel Flow at $Re_\tau = 550$

#### Setup

The simulation in which the geometrical self-similarity of vortices was assessed was conducted with the compressible Navier-Stokes solver of PyFR [107]. A summary of the setup is provided in Table 7.4. The computational domain was a box with of size $2\pi \times 2 \times \pi$ as depicted in Figure 7.1. The $x$-, $y$- and $z$-directions were chosen to be streamwise, wall-normal and spanwise directions respectively. Periodic boundary conditions were applied in the $x$- and $z$-directions and adiabatic no-slip conditions in the $y$-direction. The flow was driven by a constant pressure gradient. The density, the viscosity and the pressure gradient were set to achieve a Mach number of $Ma = 0.1$, a Reynolds number based on the bulk velocity and the channel half-height $\delta$ of $Re = 10\,000$ and a friction Reynolds number of $Re_\tau = 550$. The simulation resulted in an actual friction Reynolds number of $Re_\tau = 551.36$.

Table 7.4: Computational setup of the channel flow at $Re_\tau = 550$.

| | |
|---|---|
| solver: | PyFR |
| system: | compressible Navier-Stokes |
| domain size $\lambda_x \times 2\delta \times \lambda_z$: | $2\pi \times 2 \times \pi$ |
| mesh: | $80 \times 55 \times 80$ hexahedra |
| solution basis order: | 4 |
| friction Reynolds number $Re_\tau$: | 551.36 |
| bulk Reynolds number $Re$: | 10 000 |
| Mach number: | 0.1 |

#### Mesh

The domain was discretised using a structured mesh consisting of $80 \times 55 \times 80$ hexahedral high-order elements, in each of which the solution was represented with a fourth order polynomial. The mesh had uniform spacing in the $x$- and $z$-directions, while the $y$-direction the resolution was increasing towards the wall. The two layers of elements closest to the wall hold the first ten solution points in wall-normal direction. These two layers are located within 6.8 wall units distance from the wall.

#### Comparison with Published DNS Data

Statistics of the velocity were compared with DNS data obtained by Lee and Moser [65]. Figure 7.9 shows a comparisons of the mean streamwise velocity $\langle u \rangle^+$ and the variances of the streamwise $\langle u'^2 \rangle^+$, wall-normal $\langle v'^2 \rangle^+$ and spanwise $\langle w'^2 \rangle^+$ velocity components as a function of $y^+$. All analysed statistics are consistent across all $y^+$, providing evidence that the channel flow simulation effectively resolves the flow physics.

### 7.2.2 Setup for the Topological Analysis

Tracer was employed to extract vortices based on the topology of the $Q$-criterion field of the channel flow. For the topological analysis each high-order element of the computational mesh was subdivided into $6 \times 6 \times 6$ hexhedral cells with trilinear interpolation, resulting in a grid of $480 \times 330 \times 480$ cells. The level of subdivision was chosen following the reasoning of Jallepalli et al. [52] which was described in Section 4.2.1. In order to filter noise extracted vortices were required to contain at least 64 vertices. $R_{cut}$ was chosen via a percolation analysis of a single snapshot of the $Q$-criterion field. Figure 7.10 provides the number of identified clusters and the volume of the largest vortex $V_{max}$ normalised with the total volume of all vortices as a function of $R_{cut}$. Aiming at maximising the number of extracted features, the relevance threshold was

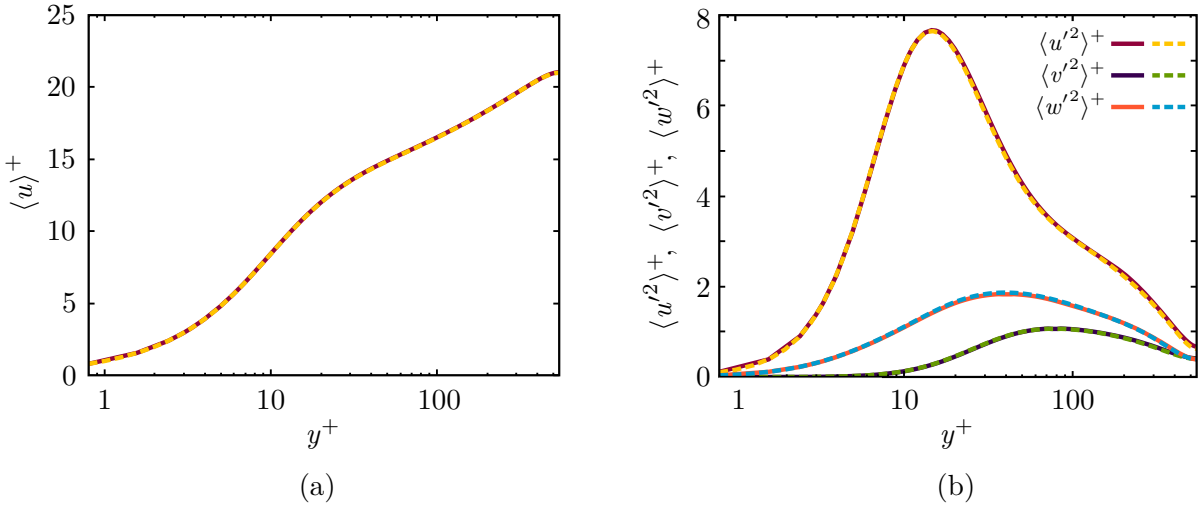



Figure 7.9: Comparison of (a) the mean streamwise velocity $\langle u \rangle^+$ and (b) the variances of the streamwise $\langle u'^2 \rangle^+$, wall-normal $\langle v'^2 \rangle^+$ and spanwise $\langle w'^2 \rangle^+$ velocity components of the channel flow simulation at $Re_\tau = 550$ (solid lines) with data of the channel flow at the same $Re_\tau$ by Lee and Moser [65] (dashed lines).

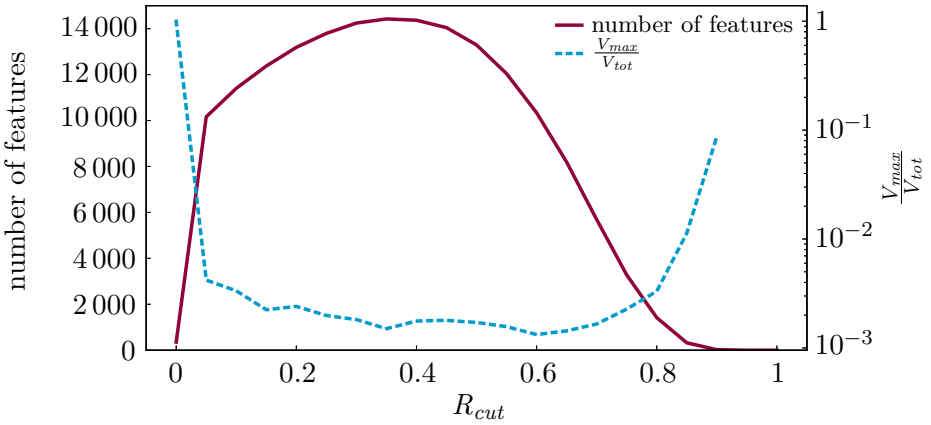



Figure 7.10: Percolation behaviour of the flow features in the channel at $Re_\tau = 550$.

set to $R_{cut} = 0.4$. Renderings of isosurfaces of $R = 0.4$ coloured by the resulting $Q$-criterion threshold are provided in Figure 7.11.

A total of 200 snapshots, which are $\Delta t^+ = 95$ apart, were processed resulting in the identification of 2 884 968 vortices. A summary of the setup can be taken from Table 7.5. The size of 200 files containing the solution of the flow field is 352 GB, converted to vtu files that size increases to 2.2 TB. For comparison the size of the csv file containing the information of all 2 884 968 vortices was only 293 MB. Storing a list of vortices instead of the full flow field hence reduced the size of data by three orders of magnitude.

### 7.2.3 Assessing Geometric Self-Similarity

In accordance with del Álamo et al. [29] only vortices which occupy a volume of $V^+ \geq (30 \text{ wall units})^3$ were considered in this analysis. Figure 7.12 shows the volume distribution of individual vortices extracted by Tracer as a function of $y^+_{min}$ and $y^+_{max}$. While del Álamo et al. found tall attached vortex clusters in their analysis, such features are missing in the present analysis of individual vortices. As a consequence the geometries of all vortices were analysed together.

Figures 7.13 (a) - (c) show the probability density functions (pdf) of the lenghts $l^+_x$, $l^+_y$ and $l^+_z$ of the vortices in the streamwise, wall-normal and spanwise direction and the distance $y^+_{com}$ of their centre from the wall. For $y^+_{com} \gtrsim 70$ all three distribution are isotropic. Vortices which have $y^+_{com} \lesssim 70$ tend to increase in $\langle l^+_x \rangle$ while decreasing in $\langle l^+_y \rangle$ and $\langle l^+_z \rangle$. Those vortices are

Table 7.5: Setup of Tracer for identifying features in the channel flow at $Re_\tau = 550$.

| | |
|---|---|
| linear subdivisions : | 6 |
| mesh: | $480 \times 330 \times 480$ hexahedra |
| number of snapshots: | 200 |
| time between snapshots $\Delta t^+$: | 95.0 |
| analysed scalar field $f$: | $Q$-criterion |
| $f_{min}$: | 0 |
| feature detection threshold $R_{cut}$: | 0.4 |
| minimal feature size $mfs$: | 64 vertices |

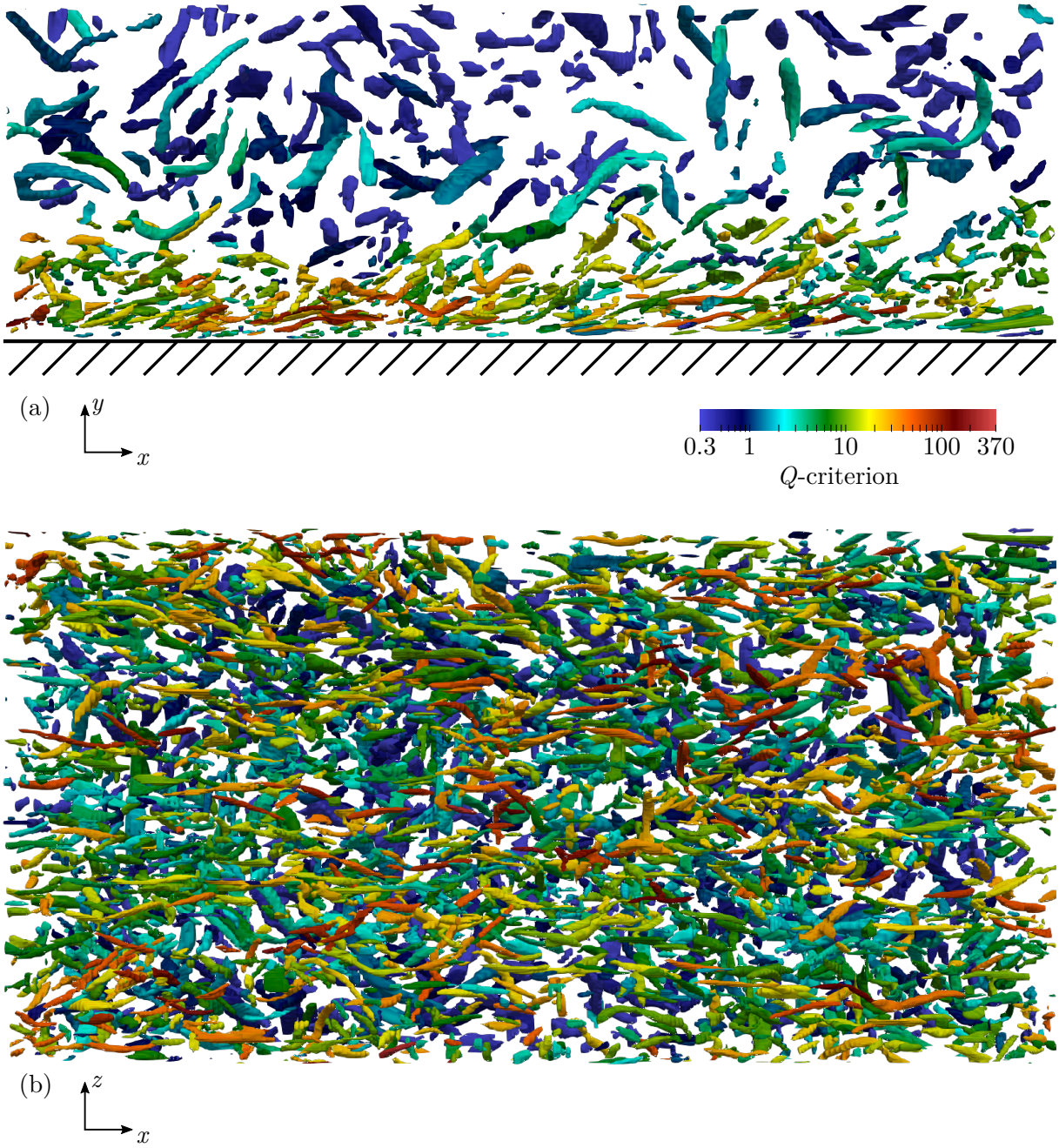



Figure 7.11: (a) Side and (b) bottom view of vortices in the channel flow at $Re_\tau = 550$. Vortices are depicted as isosurfaces of $R = 0.4$ and are coloured by the resulting $Q$-threshold

hence aligning in the streamwise direction. Figures 7.13 (d) - (f) show the pdfs of combinations of $l_x^+$, $l_y^+$ and $l_z^+$. From theses pdfs no correlation between the lenghts can be observed.

Figure 7.14 shows the pdf of the volume occupied by the vortices and $y_{com}^+$. While there is no significant change with wall-distance in the volume with highest probability, vortices with higher volume become present with increasing $y_{com}^+$.

Finally, the correlation between diameter $d^+$ of vortices and their $y_{com}^+$ shall be examined. To get a better estimate only elongated vortices were considered for this analysis. Instead of dismissing vortices with $V^+ < (30\,\text{wall units})^3$, the vortices for this analysis were required to have $l_x^+ \geq 55$ and $\sqrt{(l_x^+)^2 + (l_y^+)^2} \geq 3l_z^+$. In Figure 7.11 one can see that such elongated vortices

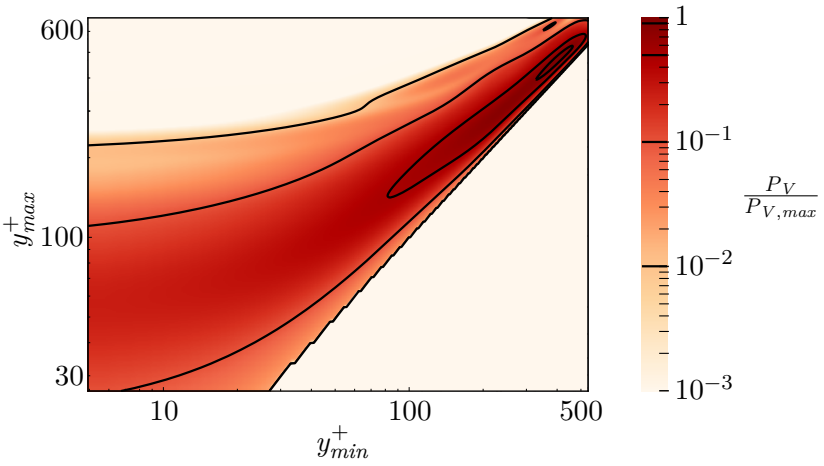



Figure 7.12: Volume distribution $P_V$ of the vortices as a function of $y^+_{min}$ and $y^+_{max}$. The contours are at 1 %, 10 %, 50 % and 90 % of $P_{V,max}$.

do not meander much, hence their shape can be approximated with a cylinder. The height $h^+$ of the cylinders was estimated to be the diagonally of their bounding box:

$$h^+ \approx \sqrt{(l_x^+)^2 + (l_y^+)^2 + (l_z^+)^2}. \tag{7.4}$$

The estimate for $d^+$ becomes:

$$d^+ \approx 2\sqrt{\frac{V^+}{\pi h^+}}. \tag{7.5}$$

Figure 7.15 shows the pdf of $d^+$ and $y^+_{com}$. The diameter increases with wall distance and is in good agreement with the grey dashed line. This line depicts a scaling with $(y^+)^{\frac{1}{4}}$, with which the Kolmogrov lenght scales [46].

### 7.2.4 Conclusion

With Tracer's capabilities individual vortices of all intensities were extracted from a channel at $Re_\tau = 550$ and analysed for geometrical self-similarity. Below $y^+ \approx 70$ the extend of vortices in the streamwise direction increases, while the length in the spanwise and the wall-normal direction decreases. This is in agreement with the description of vortices aligned in the streamwise direction by Jeong et al. [53]. Towards the channel centre vortices become istropic. Elongated vortices were extracted and their diameter was estimated by approximating the shape of such vortices with a cylinder. The analysis supports the assumption that the diameter of elongated vortices scales with the Kolmogorov length, which is proportional to $(y^+)^{\frac{1}{4}}$ [46].

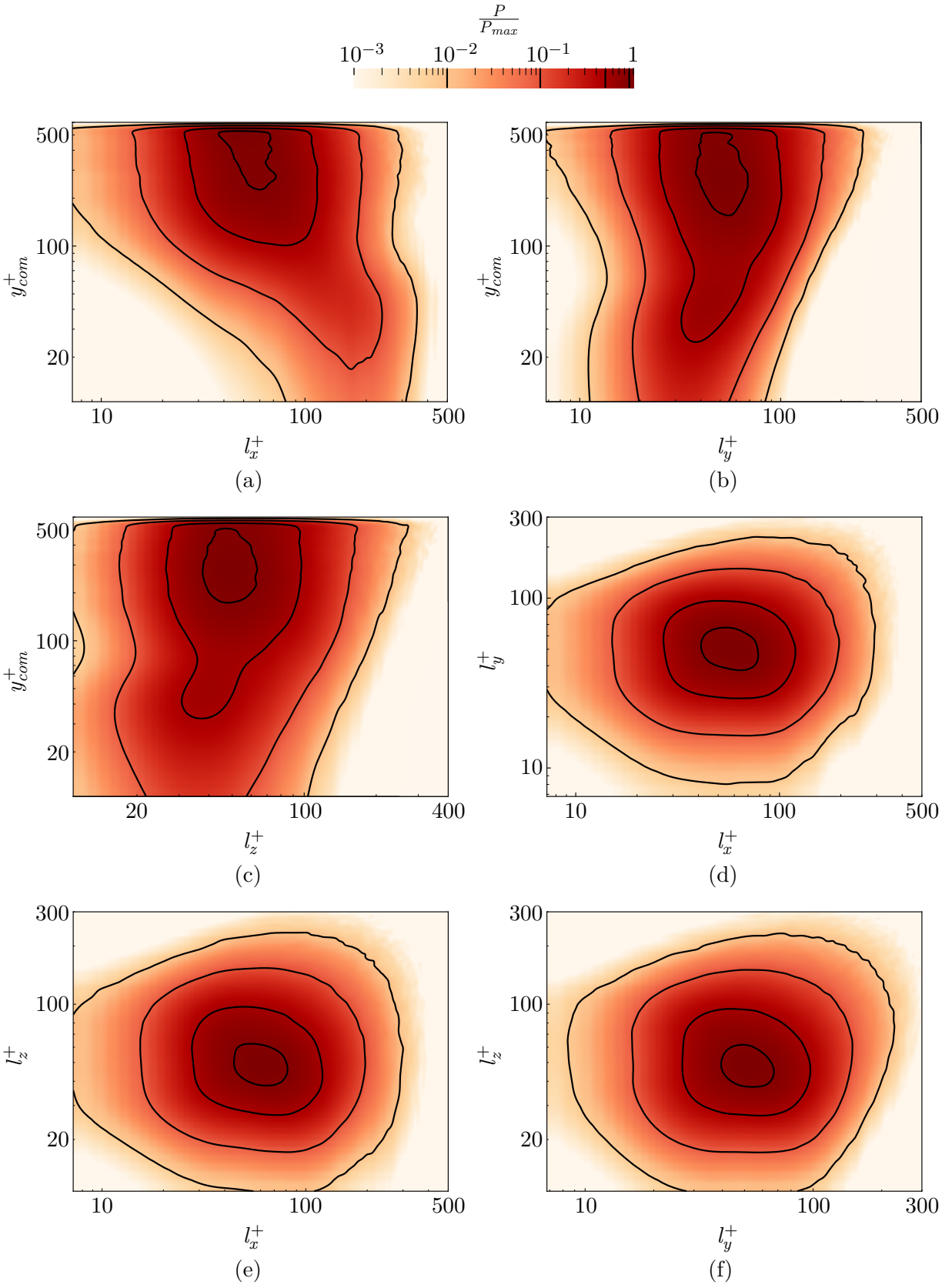



Figure 7.13: Probability density functions $P$ of the vortices as a function of (a) their length in the $x$-direction $l_x^+$ and their centre's distance from the wall $y_{com}^+$, (b) $l_y^+$ and $y_{com}^+$, (c) $l_z^+$ and $y_{com}^+$, (d) $l_x^+$ and $l_y^+$, (e) $l_x^+$ and $l_z^+$ and (f) $l_y^+$ and $l_z^+$. The contours are at 1 %, 10 %, 50 % and 90 % of $P_{max}$.

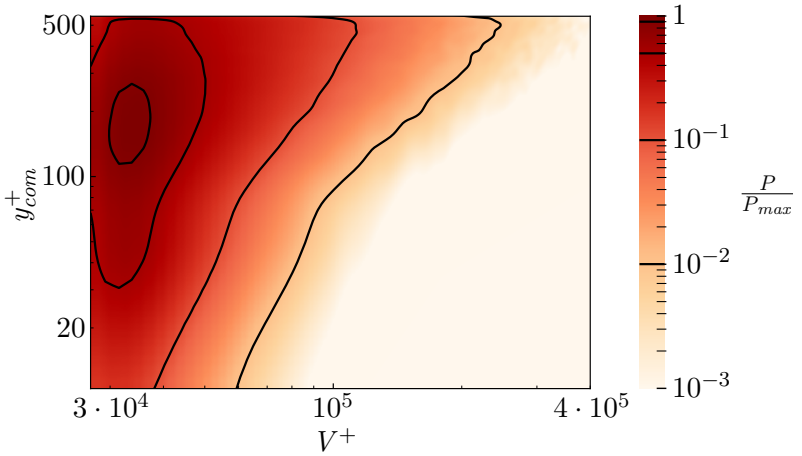



Figure 7.14: Probability density functions $P$ of the vortices as a function of their volume $V^+$ and their centre's distance from the wall $y^+_{com}$. The contours are at 1 %, 10 %, 50 % and 90 % of $P_{max}$.

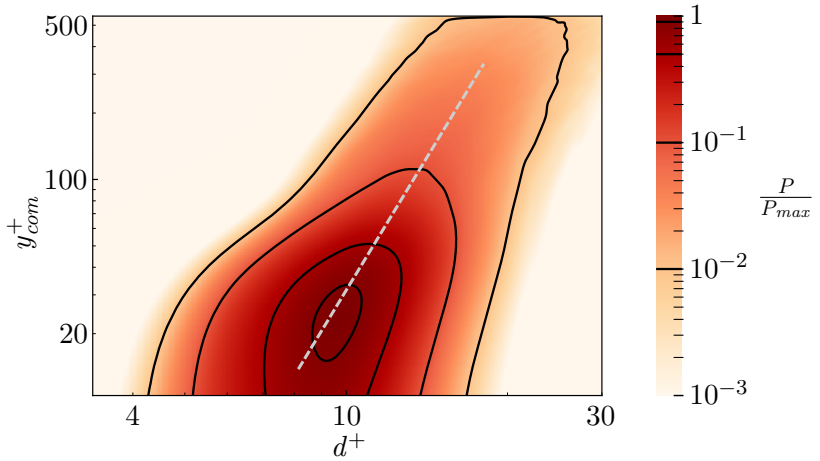



Figure 7.15: Probability density functions $P$ of elongated vortices, which fulfil $l_x^+ \geq 55$ and $\sqrt{(l_x^+)^2 + (l_y^+)^2} \geq 3l_z^+$, as a function of their diameter $d^+$ and their centre's distance from the wall $y^+_{com}$. $d^+$ was estimated with Eqs. (7.4, 7.5). The contours are at 1 %, 10 %, 50 % and 90 % of $P_{max}$. The dashed line depicts a scaling with $(y^+)^{\frac{1}{4}}$.

# 8 Summary and Future Work

Fully turbulent flow fields are populated with features such as vortices or eddies. A feature based analysis of such flows could hence help address a range of open questions in the physics of unsteady turbulent flows. The vast abundance of features in fully turbulent flow fields requires feature based analyses of such flows to be automated. To address this issue Tracer was developed, an in-situ software framework to extract flow features in data of unsteady CFD simulations conducted on GPU systems. Section 8.1 summarises Tracer's capabilities and reevaluates the results obtained with Tracer in the context of the benefits of a feature based analysis as suggested by Silver [93]. Additionally a feature based analysis of a turbulent flow is shown on the example of the turbulent channel flow. Section 8.2 is split up in two parts: the first part proposes a range of research topics in turbulent flow physics to which a feature based analysis is beneficial. Various capabilities have to be added to Tracer in order to enable the analysis of flow fields related to these research topics. The second part specifies theses missing capabilities and proposes methods that can provide Tracer with the required functionality.

## 8.1 Summary

### 8.1.1 Tracer

Tracer is an in-situ software framework to extract flow features in data of unsteady CFD simulations conducted on GPU systems. To avoid the restrictions related to moving and storing data, Tracer is able to run concurrently with the simulation, analysing the data while it is still in system memory. Contrary to the classic approach of extracting features by defining a global threshold of a scalar identifier $f$ for the whole domain, Tracer identifies an individual threshold for each feature, facilitating the identification of features over a wide range of scales and intensities. The individual threshold for each feature is identified using a JT, which represents the topology of $f$. Tracer constructs the JT using the PPP algorithm, a method which was developed to perform well on shared memory platforms. It was shown that the additional system memory required by Tracer is well below the memory used by the CFD solver. The increase in run time depends on the number of time steps of the CFD solver between two evaluations of the flow field by Tracer. Even for a number small enough to enable tracking of features the increase in runtime when applying Tracer is expected to be within the single-digit percentage range.

### 8.1.2 Reevaluation of Tracer's Results against the Benefits of a Feature Based Analysis as Suggested by Silver [93]

The advantages of a topology based feature identification using Tracer over the classic approach of using a global threshold were demonstrated qualitatively in Chapter 5 by visualising vortices

in the flow around an SD7003 aerofoil and quantitatively in Chapter 6 by counting vortices in the simulation of a TGV from start-up until the decay of the turbulence. Additionally a feature based analysis of turbulent channel flows was conducted, the results of which are presented in Chapter 7. All three applications extracted features based on the $Q$-criterion field which are considered to identify vortices. Tracer can also extract other features when being supplied with the field of an associated scalar identifier. Let us reevaluate some of the results of those three applications in the context of the benefits of a feature based analysis, which were suggested by Silver [93]:

- **Reduction of visual clutter.**
  Visualising the flow field around the SD7003 aerofoil via the classic approach of rendering isosurfaces associated with a global isovalue has a number of drawbacks: the region on the suction side of the aerofoil behind the leading edge was obstructed by intense shear, after turbulent transition has happened the isosurface formed a complex interconnected structure and depending on the chose isovalue only a certain part of the wake was visible. If vortices, which were extracted by Tracer, are visualised instead the area immediately downstream of the leading edge is not obstructed exhibiting role-up and breakdown of spanwise vortices. Downstream of the spanwise vortices turbulent transition takes place, which is reflected the formation of small and intense vortices. Moving further downstream the vortices grow in number and size while their intensity declines. Individual vortices are visible throughout the full wake. Removing all vortices below a certain size from the visualisation also provides a much clearer view of the flow field.

- **Measuring features individually.**
  Tracer can produce a csv file which includes geometric and topological parameters of all extracted features. The feature based analysis of the channel flow was carried out post-processing such csv files. That way e.g. the scaling of the diameter of vortices with wall distance could be confirmed.

- **Feature cardinality.**
  The turbulent breakdown of the TGV is indicated by a sudden increase in the number of vortices. During decay of turbulence the number of vortices is leveling off.

- **Classification of features.**
  Throughout all applications features the size of which was below *mfs* were classified as noisy features and disregarded. The the channel flow vortices were classified by e.g. by their topological complexity or distance from the wall. For the pdf of Figure 7.15 only elongated vortices were considered. Such vortices were classified by posing suitable requirements to their bounding boxes. Waters [104] successfully classified vortices, which were extracted by Tracer, by shape. An implementation of the methods he was using into Tracer is proposed in Section 8.2.2.

- **Tracking.**
  Identifying features is a prerequisite for tracking them. To date however Tracer cannot track features over multiple time steps. In Section 8.2.2 a method which has been developed to track features in turbulent flows is proposed to be added to Tracer.

- **Juxtaposition of features.**
  In Section 7.1 the topological parameters of vortices in the near-wall region of the channel flow were juxtaposed to the topological parameters of vortices in the central region of the channel.

- **Data reduction.**
  The feature based analysis of the channel flow at $Re_\tau = 550$ was done by post-processing the csv files produced by Tracer which include geometric parameters of the vortices. The size of these files was three orders of magintude smaller than the size of all evaluated snaphots of the flow field.

### 8.1.3 Analysis of Vortices in a Turbulent Channel Flow

Tracer was applied to extract vortices from a turbulent channel flow at $Re_\tau = 180$ and from a turbulent channel flow at $Re_\tau = 550$. In the channel flow at $Re_\tau = 180$ the topology of vortices in the channel centre was compared with the topology of vortices in the near-wall region. While the topology differed when vortex clusters were extracted, no such difference could be found in the topology of individual vortices. Extracting individual vortices, however, is easier as one maximises number of features. Percolation analysis have repeatedly shown that in a comfortable range around such a maximum the dependence of the number of extracted vortices and the distribution of their volume on the threshold value is small. This dependence however becomes strong when choosing smaller threshold values to extract clusters. In the channel flow at $Re_\tau = 550$ individual vortices were analysed regarding their geometric self-similarity. In such a channel flow del Álamo et al. [29] found that intense vortex clusters can be grouped into attached and detached clusters. While the detached clusters were isotropic, the authors detected self-similarity in attached clusters. A clear distinction between attached and detached could not be established for the individual vortices identified by Tracer. While vortices in the channel centre were isotropic, vortices which had their centre below $y^+ \approx 70$ aligned in the streamwise direction. This is in agreement with the description of vortices aligned in the streamwise direction by Jeong et al. [53]. When analysing elongated vortices only, Tracer's results support the assumption that the diameter of elongated vortices scales with the Kolmogorov length, which is proportional to $(y^+)^{\frac{1}{4}}$ [46].

## 8.2 Future Work

Section 8.2.1 presents a selection future applications of the current technology in the context of turbulent flow physics. In order to process data from such flow fields in the desired way, certain additional capabilities need to be added to Tracer, which are presented in Section 8.2.2.

### 8.2.1 Applying Tracer

In the work presented in Chapter 7 mainly individual vortices were extracted and analysed. Vortex clusters could be extracted by lowering $R_{cut}$ and therefore merging individual features. The analysis could then be repeated with the aim of identifying detached and self-similar attached

vortex clusters like were found by Bae and Lee [6]. Following Hwang and Sung [47] clusters of velocity fluctuation could be analysed as well. The detection methods in both [6] and [47] extracted the most intense clusters only, extraction using a topological method can furthermore provide insight into less intense features. Studies in the current work have been carried out at minimal domain sizes and at moderate $Re$ regimes mainly due to limitations in available system memory. Increasing the domain size while maintaining $Re$ would resolve the very large scale motions which reach into the outer layer of wall bounded turbulence. The investigation of flows at higher $Re$ regimes is desirable as such flows exhibit a larger range of scales. In addition further understanding of the dependence of the life time of eddies on their volume, intensity and height and how life times differ between attached and detached features can be obtained. Marusic and Monty [72] have recently formulated a range of open questions in wall-bounded turbulence, specifically in relation to the attached eddy model. Amongst them is assessing what fraction of eddies of a given scale are attached and how many are detached. They furthermore point out that current eddies used in the attached eddy method do not stem from velocity fields satisfying the Navier-Stokes equations, hence identifying candidate eddies which satisfy Navier-Stokes equation would be valuable. Whether attached self-similar features are still dominant in high $Re$ regime also is yet to be found out. There is especially a debate on whether hairpin vortices are a persistent feature in fully developed turbulence [31].

Similarly there is an ongoing debate whether Lundgren's spiral [69] exists in isotropic turbulence. While it was found by [42], Goto at al. [37] could not see it in their investigation of anti-parallel pairs of vortex tubes. In their study Goto at al. showed that isotropic turbulence at sufficiently high $Re$ contains a self-similar hierarchy of anti-parallel pairs of vortex tubes with varying lengths and diameters. They have also investigated the spatial organisation of these vortex tubes. Their extraction method for intense vortices however uses a constant threshold for each snapshot of the flow field and a low pressure method to identify centre lines of vortices of lower intensity. Topology based vortex identification would allow to extract the volume of all vortices using a single method. In their concluding remarks Goto at al. [37] proposed investigating how the hierarchy of vortices and the mechanism of energy cascade are related to the Kolmogorov scaling in the inertial range. In relation to the latter Dong et al. [30] have identified coherent structures related to energy transfer in a turbulent shear flow: energy transfer is happening in the shear layer and saddle region between hairpin-shaped features. Identifying similar features in the inertial range of isotropic turbulence can provide further insight in the mechanism of the energy cascade. Turbulence is characterised by extreme events in which the velocity gradient is multiple orders of magnitude above its average value [109]. Buaria et al. [16] qualitatively described the flow features around such extreme events: there are intense vortex tubes and less intense regions of strain. The spacing between vortex tubes varies widely from vortex tubes being isolated to a strong interaction of tubes with their surrounding. Using Tracer one could identify such features and carry out a quantitative analysis of their geometry, the distance between them and their relative orientation. A possible correlation of these results with the the position and intensity of the extreme gradient event could also be investigated.

### 8.2.2 Extending the Capabilities of Tracer

To carry out such investigations proposed in Section 8.2.1 the following capabilities need to be added to Tracer:

- **feature identification on distributed memory** to enable the topological analysis of flow fields in higher $Re$ regimes. With increasing $Re$ more scales appear in the flow requiring a finer resolution of the domain and increasing the size of the mesh. As available system memory on GPUs is limited, distributed memory parallelism is required to conduct CFD simulations in the high $Re$ regime. Tracer's analysis of the channel flow for example was limited by the the system memory on a Nvidia V100 GPU to $Re_\tau = 550$. The $Re$ regime at which many engineering applications of wall-bounded flows take place is at least one order of magnitude higher [95]. This is furthermore the regime in which certain phenomena of such flows, like a distinct log-layer or very large-scale motions, start to appear [65].
- **shape classification of features** to detect features with a characteristic shape like hair-pins,
- **tracking features over time** to study their evolution.

The following paragraphs propose methods which provide Tracer with the required capabilities.

#### Feature Identification on Distributed Memory

Tracer identifies features based on the topology of the surrounding scalar field, making this procedure intrinsically non-local. To extract features in distributed data settings, each process needs topological information of the data residing on other processes. For large simulations the size of a global JT can exceed the available memory of a single processor, prohibiting the approach of building the global JT on a single processor. Additionally having one processor combining trees from all partitions negatively impacts performance. The distributed merge tree introduced by Morozov and Weber [78] provides a local-global representation of the JT, in which each process only holds global information necessary to describe its local topology. As each processor computes its own local-global representation of the JT, the workload is also distributed over all processors. Instead of communicating the full JT between the processors, it is enough to exchange the super structure augmented with maxima in the frame of partition interfaces. Tracer constructs the JT only on connected regions above $f_{min}$. In practices a significant amount of such connected regions might reside on a single process and therefor do not need to be communicated. Morozov and Weber also provide an algorithm to merge two trees on a shared memory platform. However, one could also use the super arcs of both trees as input edge graph for PPP, which would avoid having to implement and maintain another algorithm.

#### Shape Classification of Features

Currently Tracer can only obtain basic geometric properties of extracted features like volume or centre. An automated classification of features by shape requires their shape to be parametrised. Angelidakis et al. [4] have provided a descriptor space for parameterising the shape of particles

using simple geometric properties, most of which are already implemented in Tracer. Waters [104] has investigated more sophisticated shape classifiers and came to the conclusion that Zernike-Canterakis moments [17, 57] and Voxel Variational Auto-Encoders [15, 59] are suitable descriptors. Using clustering algorithms to partition the different description spaces, Waters succeeded in establishing a correlation between the shape and the wall distance of the vortices extracted by Tracer in the channel flow presented in Section 7.1.

### Tracking Features over Time

Lozano-Durán and Jiménez [68] have described a tracking algorithm which organises the evolution of features in a graph that spans along the time axis. The nodes of the graph represent features and the edges between them represent their connection. Features of consecutive snaphots are considered connected if they have spatial overlap. Nodes with no connection to the negative direction of time are associated with the birth of features, nodes with no connection to the positive direction of time are associated with the death of features. If a node has more than one connection in the negative direction multiple features are merging, if a node has more than one connection in the positive direction the feature splits up.

# Bibliography


[1] M. Abbassi, W. Baars, N. Hutchins, and I. Marusic. Skin-friction drag reduction in a high-Reynolds-number turbulent boundary layer via real-time control of large-scale structures. *International Journal of Heat and Fluid Flow*, 67:30–41, 2017.

[2] L. Agostini, E. Touber, and M. A. Leschziner. Spanwise oscillatory wall motion in channel flow: drag-reduction mechanisms inferred from DNS-predicted phase-wise property variations at. *Journal of Fluid Mechanics*, 743:606–635, 2014.

[3] B.-K. Ahn, W. Graham, and S. Rizzi. A structure-based model for turbulent-boundary-layer wall pressures. *Journal of Fluid Mechanics*, 650:443–478, 2010.

[4] V. Angelidakis, S. Nadimi, and S. Utili. SHape Analyser for Particle Engineering (SHAPE): Seamless characterisation and simplification of particle morphology from imaging data. *Computer Physics Communications*, 265:107983, 2021.

[5] U. Ayachit. *The paraview guide: a parallel visualization application*. Kitware, Inc., 2015.

[6] H. J. Bae and M. Lee. Life cycle of streaks in the buffer layer of wall-bounded turbulence. *Physical Review Fluids*, 6(6):064603, 2021.

[7] H. Bai, Y. Zhou, W. Zhang, S. Xu, Y. Wang, and R. Antonia. Active control of a turbulent boundary layer based on local surface perturbation. *Journal of Fluid Mechanics*, 750:316, 2014.

[8] D. Bechert, M. Bruse, W. v. Hage, J. T. Van der Hoeven, and G. Hoppe. Experiments on drag-reducing surfaces and their optimization with an adjustable geometry. *Journal of Fluid Mechanics*, 338:59–87, 1997.

[9] N. Bell and J. Hoberock. Thrust: A productivity-oriented library for CUDA. In *GPU computing gems Jade edition*, pages 359–371. Elsevier, 2012.

[10] J. C. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, et al. Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–9. IEEE, 2012.

[11] J. C. Bennett, V. Krishnamoorthy, S. Liu, R. W. Grout, E. R. Hawkes, J. H. Chen, J. Shepherd, V. Pascucci, and P.-T. Bremer. Feature-based statistical analysis of combustion simulation data. *IEEE transactions on visualization and computer graphics*, 17(12):1822–1831, 2011.

[12] P. S. Bernard, J. M. Thomas, and R. A. Handler. Vortex dynamics and the production of reynolds stress. *Journal of Fluid Mechanics*, 253:385–419, 1993.

[13] M. E. Brachet, D. I. Meiron, S. A. Orszag, B. Nickel, R. H. Morf, and U. Frisch. Small-scale structure of the Taylor–Green vortex. *Journal of Fluid Mechanics*, 130:411–452, 1983.

[14] P.-T. Bremer, A. Gruber, J. Bennett, A. Gyulassy, H. Kolla, J. Chen, and R. Grout. Identifying turbulent structures through topological segmentation. *Communications in Applied Mathematics and Computational Science*, 11(1):37–53, 2015.

[15] A. Brock, T. Lim, J. M. Ritchie, and N. Weston. Generative and discriminative voxel modeling with convolutional neural networks. *arXiv preprint arXiv:1608.04236*, 2016.

[16] D. Buaria, A. Pumir, E. Bodenschatz, and P.-K. Yeung. Extreme velocity gradients in turbulent flows. *New Journal of Physics*, 21(4):043004, 2019.

[17] N. Canterakis. 3D Zernike moments and Zernike affine invariants for 3D image analysis and recognition. In *In 11th Scandinavian Conf. on Image Analysis*. Citeseer, 1999.

[18] H. Carr, O. Rubel, G. H. Weber, and J. Ahrens. Optimization and augmentation for data parallel contour trees. *IEEE Transactions on Visualization and Computer Graphics*, 2021.

[19] H. Carr and J. Snoeyink. Representing interpolant topology for contour tree computation. In *Topology-Based Methods in Visualization II*, pages 59–73. Springer, 2009.

[20] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. *Computational Geometry*, 24(2):75–94, 2003.

[21] H. Carr, J. Snoeyink, and M. van de Panne. Simplifying flexible isosurfaces using local geometric measures. In *Proceedings of the conference on Visualization'04*, pages 497–504. IEEE Computer Society, 2004.

[22] H. Carr, J. Snoeyink, and M. Van De Panne. Flexible isosurfaces: Simplifying and displaying scalar topology using the contour tree. *Computational Geometry*, 43(1):42–58, 2010.

[23] H. Carr, G. H. Weber, C. Sewell, O. Rübel, P. Fasel, and J. Ahrens. Scalable contour tree computation by data parallel peak pruning. *IEEE Transactions on Visualization and Computer Graphics*, 2019.

[24] H. A. Carr, G. H. Weber, C. M. Sewell, and J. P. Ahrens. Parallel peak pruning for scalable SMP contour tree computation. In *Large Data Analysis and Visualization (LDAV), 2016 IEEE 6th Symposium on*, pages 75–84. IEEE, 2016.

[25] Y.-J. Chiang, T. Lenz, X. Lu, and G. Rote. Simple and optimal output-sensitive construction of contour trees using monotone paths. *Computational Geometry*, 30(2):165–195, 2005.

[26] H. Choi, P. Moin, J. Kim, et al. Active turbulence control for drag reduction in wall-bounded flows. *Journal of Fluid Mechanics*, 262:75–75, 1994.

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

[28] D. B. De Graaff and J. K. Eaton. Reynolds-number scaling of the flat-plate turbulent boundary layer. *Journal of Fluid Mechanics*, 422:319–346, 2000.

[29] J. C. Del Álamo, J. Jimenez, P. Zandonade, and R. D. Moser. Self-similar vortex clusters in the turbulent logarithmic region. *Journal of Fluid Mechanics*, 561:329–358, 2006.

[30] S. Dong, Y. Huang, X. Yuan, and A. Lozano-Durán. The coherent structure of the kinetic energy transfer in shear turbulence. *Journal of Fluid Mechanics*, 892, 2020.

[31] S. Dong, A. Lozano-Durán, A. Sekimoto, and J. Jiménez. Coherent structures in statistically stationary homogeneous shear turbulence. *Journal of Fluid Mechanics*, 816:167–208, 2017.

[32] H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci. Morse-Smale complexes for piecewise linear 3-manifolds. In *Proceedings of the nineteenth annual symposium on Computational geometry*, pages 361–370. ACM, 2003.

[33] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics (tog)*, 9(1):66–104, 1990.

[34] M. S. Elbaz, E. E. Calkoen, J. J. Westenberg, B. P. Lelieveldt, A. A. Roest, and R. J. Van Der Geest. Vortex flow during early and late left ventricular filling in normal subjects: quantitative characterization using retrospectively-gated 4D flow cardiovascular magnetic resonance and three-dimensional vortex core analysis. *Journal of Cardiovascular Magnetic Resonance*, 16(1):1–12, 2014.

[35] A. Elnahhas, A. Lozano-Durán, and P. Moin. Toward a flow-structure-based wall-modeled large-eddy simulation paradigm. *arXiv preprint arXiv:2101.00528*, 2021.

[36] D. Gatti and M. Quadrio. Reynolds-dependence of turbulent skin-friction drag reduction induced by spanwise forcing. *Journal of Fluid Mechanics*, 802:553–582, 2016.

[37] S. Goto, Y. Saito, and G. Kawahara. Hierarchy of antiparallel vortex tubes in spatially periodic turbulence at high reynolds numbers. *Physical Review Fluids*, 2(6):064603, 2017.

[38] C. Gueunet, P. Fortin, and J. Jomier. Contour forests: Fast multi-threaded augmented contour trees. In *Large Data Analysis and Visualization (LDAV), 2016 IEEE 6th Symposium on*, pages 85–92. IEEE, 2016.

[39] C. Gueunet, P. Fortin, J. Jomier, and J. Tierny. Task-based augmented merge trees with fibonacci heaps. In *2017 IEEE 7th Symposium on Large Data Analysis and Visualization (LDAV)*, pages 6–15. IEEE, 2017.

[40] T. Günther, M. Gross, and H. Theisel. Generic objective vortices for flow visualization. *ACM Transactions on Graphics (TOG)*, 36(4):1–11, 2017.

[41] J. M. Hamilton, J. Kim, and F. Waleffe. Regeneration mechanisms of near-wall turbulence structures. *Journal of Fluid Mechanics*, 287(1):317–348, 1995.

[42] K. Horiuti and T. Fujisawa. The multi-mode stretched spiral vortex in homogeneous isotropic turbulence. *Journal of Fluid Mechanics*, 595:341–366, 2008.

[43] R. E. Horton. Erosional development of streams and their drainage basins; hydrophysical approach to quantitative morphology. *Geological society of America bulletin*, 56(3):275–370, 1945.

[44] J. C. Hunt, A. A. Wray, and P. Moin. Eddies, streams, and convergence zones in turbulent flows. 1988.

[45] N. Hutchins, K. Chauhan, I. Marusic, J. Monty, and J. Klewicki. Towards reconciling the large-scale structure of turbulent boundary layers in the atmosphere and laboratory. *Boundary-layer meteorology*, 145(2):273–306, 2012.

[46] N. Hutchins, T. B. Nickels, I. Marusic, and M. Chong. Hot-wire spatial resolution issues in wall-bounded turbulence. *Journal of Fluid Mechanics*, 635:103, 2009.

[47] J. Hwang and H. J. Sung. Wall-attached structures of velocity fluctuations in a turbulent boundary layer. *Journal of Fluid Mechanics*, 856:958–983, 2018.

[48] Y. Hwang. Statistical structure of self-sustaining attached eddies in turbulent channel flow. 2015.

[49] Y. Hwang and Y. Bengana. Self-sustaining process of minimal attached eddies in turbulent channel flow. *Journal of Fluid Mechanics*, 795:708–738, 2016.

[50] A. Iyer, F. Witherden, S. Chernyshenko, and P. Vincent. Identifying eigenmodes of averaged small-amplitude perturbations to turbulent channel flow. *Journal of Fluid Mechanics*, 875:758–780, 2019.

[51] J. JáJá. *An introduction to parallel algorithms*, volume 17. Addison-Wesley Reading, 1992.

[52] A. Jallepalli, J. A. Levine, and R. M. Kirby. The effect of data transformations on scalar field topological analysis of high-order FEM solutions. *IEEE transactions on visualization and computer graphics*, 26(1):162–172, 2019.

[53] J. Jeong, F. Hussain, W. Schoppa, and J. Kim. Coherent structures near the wall in a turbulent channel flow. *Journal of Fluid Mechanics*, 332(185-214):188, 1997.

[54] J. Jiménez and A. Pinelli. The autonomous cycle of near-wall turbulence. *Journal of Fluid Mechanics*, 389:335–359, 1999.

[55] G. Katul and B. Vidakovic. The partitioning of attached and detached eddy motion in the atmospheric surface layer using lorentz wavelet filtering. *Boundary-Layer Meteorology*, 77(2):153–172, 1996.

[56] A. Kheradvar and G. Pedrizzetti. Vortex formation in the heart. In *Vortex formation in the cardiovascular system*, pages 45–79. Springer, 2012.

[57] A. Khotanzad and Y. H. Hong. Invariant image recognition by Zernike moments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(5):489–497, 1990.

[58] P. J. Kilner, G.-Z. Yang, A. J. Wilkes, R. H. Mohiaddin, D. N. Firmin, and M. H. Yacoub. Asymmetric redirection of flow through the heart. *Nature*, 404(6779):759–761, 2000.

[59] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[60] M. K. Koch, P. H. Kelly, and P. Vincent. Identification and classification of off-vertex critical points for contour tree construction on unstructured meshes of hexahedra. *IEEE Transactions on Visualization and Computer Graphics*, 2021.

[61] A. G. Kravchenko, H. Choi, and P. Moin. On the relation of near-wall streamwise vortices to wall skin friction in turbulent boundary layers. *Physics of Fluids A: Fluid Dynamics*, 5(12):3307–3309, 1993.

[62] 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 *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1020–1031. IEEE, 2014.

[63] D. Laney, A. Mascarenhas, P. Miller, V. Pascucci, et al. Understanding the structure of the turbulent mixing layer in hydrodynamic instabilities. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1053–1060, 2006.

[64] M. Larsen, J. Ahrens, U. Ayachit, E. Brugger, H. Childs, B. Geveci, and C. Harrison. The alpine in situ infrastructure: Ascending from the ashes of strawman. In *Proceedings of the In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization*, pages 42–46. 2017.

[65] M. Lee and R. D. Moser. Direct numerical simulation of turbulent channel flow up to $Re_\tau = 5\,200$. *Journal of Fluid Mechanics*, 774:395–415, 2015.

[66] J. C. Lin. Review of research on low-profile vortex generators to control boundary-layer separation. *Progress in Aerospace Sciences*, 38(4-5):389–420, 2002.

[67] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM siggraph computer graphics*, volume 21, pages 163–169. ACM, 1987.

[68] A. Lozano-Durán and J. Jiménez. Time-resolved evolution of coherent structures in turbulent channels: characterization of eddies and cascades. *Journal of Fluid Mechanics*, 759:432–471, 2014.

[69] T. Lundgren. Strained spiral vortex model for turbulent fine structure. *The Physics of Fluids*, 25(12):2193–2203, 1982.

[70] S. Maadasamy, H. Doraiswamy, and V. Natarajan. A hybrid parallel algorithm for computing and tracking level set topology. In *High Performance Computing (HiPC), 2012 19th International Conference on*, pages 1–10. IEEE, 2012.

[71] E. Manders, R. Hoebe, J. Strackee, A. Vossepoel, and J. Aten. Largest contour segmentation: a tool for the localization of spots in confocal images. *Cytometry: The Journal of the International Society for Analytical Cytology*, 23(1):15–21, 1996.

[72] I. Marusic and J. P. Monty. Attached eddy model of wall turbulence. *Annual Review of Fluid Mechanics*, 51:49–74, 2019.

[73] A. Mascarenhas, R. Grout, C. S. Yoo, and J. Chen. Tracking flame base movement and interaction with ignition kernels using topological methods. In *Journal of Physics: Conference Series*, volume 180, page 012086. IOP Publishing, 2009.

[74] M. G. Meena and K. Taira. Identifying vortical network connectors for turbulent flow modification. *Journal of Fluid Mechanics*, 915, 2021.

[75] M. Metzger and J. Klewicki. A comparative study of near-wall turbulence in high and low reynolds number boundary layers. *Physics of Fluids*, 13(3):692–701, 2001.

[76] J. Milnor. *Morse Theory.(AM-51)*, volume 51. Princeton university press, 1963.

[77] F. Moisy and J. Jiménez. Geometry and clustering of intense structures in isotropic turbulence. *Journal of Fluid Mechanics*, 513:111–133, 2004.

[78] D. Morozov and G. Weber. Distributed merge trees. In *ACM SIGPLAN Notices*, volume 48, pages 93–102. ACM, 2013.

[79] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with CUDA: Is CUDA the parallel programming model that application developers have been waiting for? *Queue*, 6(2):40–53, 2008.

[80] G. M. Nielson. On marching cubes. *IEEE Transactions on visualization and computer graphics*, 9(3):283–297, 2003.

[81] G. M. Nielson and B. Hamann. The asymptotic decider: resolving the ambiguity in marching cubes. In *Proceedings of the 2nd conference on Visualization'91*, pages 83–91. IEEE Computer Society Press, 1991.

[82] P. Orlandi and J. Jiménez. On the generation of turbulent wall friction. *Physics of Fluids*, 6(2):634–641, 1994.

[83] R. Örlü and R. Vinuesa. Instantaneous wall-shear-stress measurements: advances and application to near-wall extreme events. *Measurement Science and Technology*, 31(11):112001, 2020.

[84] H. Park, G. Sun, and C.-J. Kim. Superhydrophobic turbulent drag reduction as a function of surface grating parameters. *Journal of Fluid Mechanics*, 747:722–734, 2014.

[85] V. Pascucci and K. Cole-McLaughlin. Parallel computation of the topology of level sets. *Algorithmica*, 38(1):249–268, 2004.

[86] S. D. Peckham. New results for self-similar trees with applications to river networks. *Water Resources Research*, 31(4):1023–1029, 1995.

[87] A. Perry and M. Chong. On the mechanism of wall turbulence. *Journal of Fluid Mechanics*, 119:173–217, 1982.

[88] G. Querzoli, S. Fortini, and A. Cenedese. Effect of the prosthetic mitral valve on vortex dynamics and turbulence of the left ventricular flow. *Physics of fluids*, 22(4):041901, 2010.

[89] W. Schoppa and F. Hussain. A large-scale control strategy for drag reduction in turbulent boundary layers. *Physics of Fluids*, 10(5):1049–1051, 1998.

[90] W. Schoppa and F. Hussain. Coherent structure generation in near-wall turbulence. *Journal of fluid Mechanics*, 453:57, 2002.

[91] W. J. Schroeder, B. Lorensen, and K. Martin. *The visualization toolkit: an object-oriented approach to 3D graphics (4th ed.)*. Kitware, 2006.

[92] J. A. Sillero, J. Jiménez, and R. D. Moser. One-point statistics for turbulent wall-bounded flows at reynolds numbers up to $\delta^+ \approx 2\,000$. *Physics of Fluids*, 25(10):105102, 2013.

[93] D. Silver. Object-oriented visualization. *IEEE Computer Graphics and Applications*, 15(3):54–62, 1995.

[94] D. Smirnov and D. Morozov. Triplet merge trees. In *Topological Methods in Data Analysis and Visualization*, pages 19–36. Springer, 2017.

[95] A. J. Smits and I. Marusic. Wall-bounded turbulence. *Phys. Today*, 66(9):25–30, 2013.

[96] A. N. Strahler. Quantitative analysis of watershed geomorphology. *Eos, Transactions American Geophysical Union*, 38(6):913–920, 1957.

[97] G. I. Taylor and A. E. Green. Mechanism of the production of small eddies from large ones. *Proceedings of the Royal Society of London. Series A-Mathematical and Physical Sciences*, 158(895):499–521, 1937.

[98] H. Tieleman, D. Surry, and J. Lin. Characteristics of mean and fluctuating pressure coefficients under corner (delta wing) vortices. *Journal of Wind Engineering and Industrial Aerodynamics*, 52:263–275, 1994.

[99] E. Tokunaga. The composition of drainage network in Toyohira River Basin and valuation of Horton's first law. *Geophys. Bull. Hokkaido Univ.*, 15:1–19, 1966.

[100] E. Tokunaga. Consideration on the composition of drainage networks and their evolution. *Geogr. Rep. Tokyo Metrop. Univ.*, 13:1–27, 1978.

[101] A. Townsend. Equilibrium layers and wall turbulence. *Journal of Fluid Mechanics*, 11(1):97–120, 1961.

[102] A. Townsend. *The structure of turbulent shear flow*. Cambridge university press, 1976.

[103] B. C. Vermeire, F. D. Witherden, and P. E. Vincent. On the utility of GPU accelerated high-order methods for unsteady flow simulations: A comparison with industry-standard tools. *Journal of Computational Physics*, 334:497–521, 2017.

[104] C. Waters. Characterising vortices using 3D shape analysis. Master's thesis, Imperial College London, Dept. of Computing, 2020.

[105] G. H. Weber, G. Scheuermann, H. Hagen, and B. Hamann. Exploring scalar fields using critical isovalues. In *Visualization, 2002. VIS 2002. IEEE*, pages 171–178. IEEE, 2002.

[106] F. Witherden and A. Jameson. Impact of number representation for high-order implicit large-eddy simulations. *AIAA Journal*, 58(1):184–197, 2020.

[107] F. D. Witherden, A. M. Farrington, and P. E. Vincent. PyFR: an open source framework for solving advection–diffusion type problems on streaming architectures using the flux reconstruction approach. *Computer Physics Communications*, 185(11):3028–3040, 2014.

[108] J. Yao, X. Chen, and F. Hussain. Drag control in wall-bounded turbulent flows via spanwise opposed wall-jet forcing. *Journal of Fluid Mechanics*, 852:678–709, 2018.

[109] P. Yeung, X. Zhai, and K. R. Sreenivasan. Extreme events in computational turbulence. *Proceedings of the National Academy of Sciences*, 112(41):12633–12638, 2015.

[110] Y. Yu, P. Shrestha, O. Alvarez, C. Nottage, and C. Liu. Investigation of correlation between vorticity, $Q$, $\lambda_{ci}$, $\lambda_2$, $\Delta$ and Liutex. *Computers & Fluids*, 225:104977, 2021.

[111] I. Zaliapin and Y. Kovchegov. Tokunaga and Horton self-similarity for level set trees of Markov chains. *Chaos, Solitons & Fractals*, 45(3):358–372, 2012.

[112] S. Zanardo, I. Zaliapin, and E. Foufoula-Georgiou. Are American rivers Tokunaga self-similar? New results on fluvial network topology and its climatic dependence. *Journal of Geophysical Research: Earth Surface*, 118(1):166–183, 2013.

[113] X. Zhang, M. Hadwiger, T. Theul, and P. Rautek. Interactive exploration of physically-observable objective vortices in unsteady 2d flow. *IEEE Transactions on Visualization & Computer Graphics*, (01):1–1, 2021.