DISSERTATION
submitted
to the
Combined Faculty for the Natural Sciences and Mathematics
of
Heidelberg University, Germany
for the degree of
Doctor of Natural Sciences

Put forward by

Dipl.-Math. Florian Flatow
Born in: Heidelberg, Germany
Oral examination: .....

Feature-Based Vector Field Representation and Comparison

Advisor: Prof. Dr. Michael Gertz

Abstract

In recent years, simulations have steadily replaced real world experiments in science and industry. Instead of performing numerous arduous experiments in order to develop new products or test a hypothesis, the system to be examinded is described by a set of equations which are subsequently solved within the simulation. The produced vector fields describe the system's behavior under the conditions of the experiment. While simulations steadily increase in terms of complexity and precision, processing and analysis are still approached by the same long-standing visual techniques. However, these are limited by the capability of the human visual system and its abilities to depict large, multi-dimensional data sets.

In this thesis, we replace the visual processing of data in the traditional workflow with an automated, statistical method. Cluster algorithms are able to process large, multi-dimensional data sets efficiently and therefore resolve the limitations we faced so far. For their application to vector fields we define a special *feature vector* that describes the data comprehensively. After choosing an appropriate clustering method, the vector field is split into its *features*.

Based on these *features* the novel *flow graph* is constructed. It serves as an abstract representation of the vector field and gives a detailed description of its parts as well as their relations. This new representation enables a quantitative analysis and describes the input data. Additionally, the *flow graphs* are comparable to each other through a uniform description, since techniques of graph theory may be applied. In the traditional workflow, visualization is the bottleneck, because it is built manually by the user for a specific data set. In consequence the output is diminished and the results are likely to be biased by the user. Both issues are solved by our approach, because both the feature extraction and the construction of the flow graph are executed in an un-supervised manner.

We will compare our newly developed workflow with visualization techniques based on different data sets and discuss the results. The concluding chapter on the similarity and comparison of graphs applies techniques of graph theory and demonstrates the advantages of the developed representation and its use for the analysis of vector fields using flow graphs.

Zusammenfassung

Über die letzten Jahre hinweg haben Simulationen klassische Experimente immer weiter aus der Anwendung in Industrie und Forschung verdrängt. Anstatt zahlreiche, aufwändige Versuche durchzuführen, um neue Produkte zu entwickeln beziehungsweise Fragestellungen zu überprüfen, wird das zu untersuchende System durch Gleichungen beschrieben, die dann innerhalb einer Simulation gelöst werden. Die erzeugten Vektorfelder beschreiben dieser Verhalten des Systems unter den spezifischen Simulationsbedingungen. Während die Simulationen immer aufwändiger und präziser werden, wird die Aufbereitung und Analyse weiterhin durch die selben visuellen Verfahren bewältigt wie zuvor. Diese sind allerdings durch die Leistungsfähigkeit des menschlichen Auges in ihren Möglichkeiten große, mehrdimensionale Datensätze darzustellen limitiert.

In dieser Arbeit ersetzen wir die visuelle Aufbereitung der Daten im klassischen Simulationsworkflow durch ein automatisiertes, statistisches Verfahren. Clustering-Algorithmen können große, mehrdimensionale Datensätze effizient verarbeiten und dadurch die bisherigen Einschränkungen aufheben. Für deren Anwendung auf Vektorfelder definieren wir einen eigenen *Feature Vektor*, der die Daten umfassend beschreibt. Nach der Wahl eines geeigneten Clustering-Verfahrens wird das Vektorfeld in seine *Features* zerlegt. Basierend auf diese *Features* wird der neuartige *Fluss-Graph* konstruiert. Er dient als abstrakte Darstellung des Vektorfeldes und beschreibt ausführlich seine Bestandteile und deren Beziehungen untereinander. Diese neue Darstellungsart ermöglicht erstmalig die quantitative Analyse der Eingangsdaten. Zusätzlich werden die *Fluss-Graphen* durch ihre einheitliche Beschreibung untereinander vergleichbar, da sich hierfür Techniken der Graphentheorie anwenden lassen. Im klassischen Workflow stellt die Visualisierung den Engpass dar, weil sie manuell durch den Anwender für einen spezifischen Datensatz angefertigt wird. Einerseits wird hierdurch der Durchsatz gemindert, andererseits erhält der Anwender einen großen Einfluss auf die Ergebnisse. Beide Effekte werden durch unseren Workflow verringert, weil sowohl die Extraktion der *Features* als auch die Konstruktion des *Fluss-Graphen* automatisch ausgeführt werden.

Anhand verschiedenen Datensätze vergleichen wir unseren entwickelten Workflow mit Visualisierungstechniken und diskutieren die Ergebnisse. Der nachfolgende Abschnitt über die Ähnlichkeit und Vergleichbarkeit von Graphen wendet Techniken der Graphentheorie an und demonstriert die Vorteile unserer entwickelten Darstellung beziehungsweise deren Nutzen für die Analyse von Vektorfeldern mittels ihrer Fluss-Graphen.

To Gottfried Flatow
(1956 - 2014)

Contents

Chapter 1

Introduction

In recent years, simulations have steadily replaced real world experiments in science and industry. Instead of performing numerous experiments to develop new products or test new hypotheses, the system is described by a set of equations. Those equations describe the properties of the system and are solved during the simulation process. The so obtained results give insight into the behavior of the system under the model's conditions. For instance, in 2010 the Virgin VR-01 Formula One racing car was developed and it is the first Formula One racing car ever designed entirely by computational fluid dynamics (CFD) [35]. This is a very noticeable event in motorsport, because it was the first attempt to forgo any wind channel testing or other costly experiments and to still be competitive on the cutting edge technology level of modern Formula One. Frankly, this first attempt did not end very well. Virgin finished the 2010 Formula One season in the very last position [3], but the team managed to run the season with a smaller budget than any other teams performing traditional wind channel tests. Since then the abandonment of wind channels has often been discussed in the Formula One community to reduce the expenses for car body development.

This trend is not only observable in Formula One, but also in many other domains that traditionally rely on experiments. Those experiments, which are costly in time and money, are replaced by cheaper and more versatile simulations. Additionally, the performance of computers is steadily growing by Moore's Law [100] meaning it doubles approximately every two years. This allows a more accurate description of the real world constraints, which affect the simulation's complexity in two ways. First, the simulation's resolution can be refined by an increased number of data points for the discretization, and second, the underlying model can be described by more equations and parameters, which increases the dimensionality of the output. The

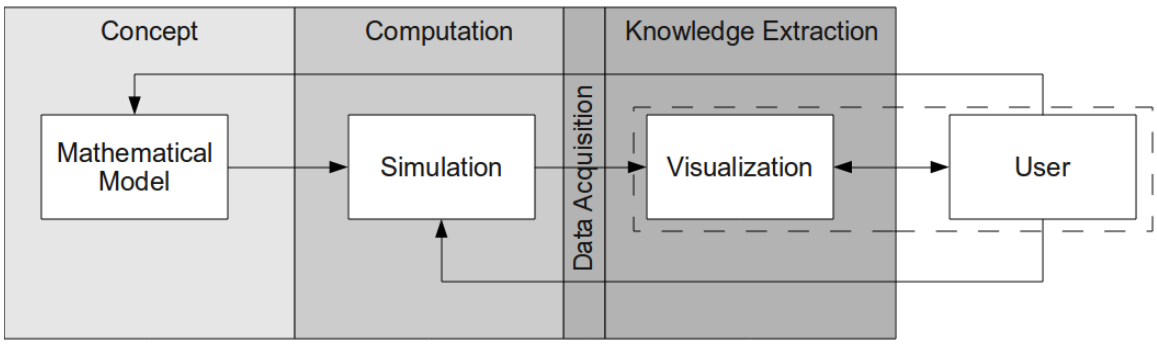


Figure 1.1: Traditional simulation workflow with concluding visualization. It consists of a model describing the system mathematically by a set of equations. Those are solved in the simulation step. In the knowledge extraction step, the user and the visualization form an iterative loop to analyze the computed data.

most obvious side effect of this development is the growing file size of the output, as more generated data needs to be stored. Writing the output files to larger storage systems is straightforward, since the systems grow proportionally as the computer's performance. The main challenge, however, is the analysis of the produced data and to understand the results of the simulation process. The analysis of the computed CFD data is very challenging even for field experts. For instance in context of the previous Formula One example, the well established Ferrari team, which is the biggest and best funded Formula One team, even occasionally has problems interpreting the data [4].

Traditionally, the data is visualized to highlight significant structures or so-called features [64]. The visualization forms the major knowledge extraction layer between the data acquisition and the user's perception as it is shown in Figure 1.1. It typically consists of different filters and feature extraction techniques forming the so-called pipelines. They are highly specialized for their specific input and purpose, i.e., they specifically focus on distinct features while disregarding others. Additionally, the pipeline must be reworked manually to adapt the parameters if the input data changes or is modified. This is even necessary for data from the same origin, e.g., consecutive time steps of the same simulation.

In the typical simulation workflow, the user is the most essential factor. He models the scenario, sets up the equations, executes the simulation and also builds the visualization pipeline - basically everything except solving the equations. By judging the visualization's output he also decides, if the results are just badly represented by the visualization or if he needs to model the real world constraints differently and re-run the complete process. Because the user is responsible for building the simulation

and judging its results, his visual analysis of the results may focus too narrowly on the effects, which he assumes and wants to demonstrate by the simulation. Therefore, an objective analysis is needed, which is less user-dependent [83] and more general than the classic visualization techniques.

1.1 Vector Fields and Visualization

The simulation's output and its properties depend on the simulated model. For high-dimensional scientific simulations, which are mainly formed by numeric values, vector fields are the common form of representation of the results. Vector fields are defined for a set of points, the so-called data points, forming the vector field's domain. The simulation's functions map them onto the vector field's vector space, i.e., a vector is computed for every given point in space representing a distinct behavior. This vectorial component is called data vector or just flow, because it represents the dynamics of the vector field. They can be seen as an uniform description of scientific data, which is mainly formed by numeric data, e.g., the weather forecast. The shown wind field in Figure 1.2 is a typical example for a simple visualization of a multidimensional vector field projected onto two-dimensional space.

Vector fields are found in many other disciplines, too. One of the oldest applications is classic mechanics, in which emerging forces are represented by vectors [87]. In a gravitational field of a large body, every point in space is attracted along a radial force towards the body's center of mass. In this example the force is represented by a vector with origin at the data point and pointing towards the center with the magnitude of the force. Another classical example is the magnetic field, which behaves analogously to the gravitational field. In life science and medicine vector fields are used to describe the blood flow through the vessels [154], but also other flows and physical processes inside the human body [101].

Vector fields are generated by simulations and experiments. Simulations compute the behavior at every given data point while experiments measure those values through sensors. The simulated or, respectively, measured properties of the vector categorize the vector field. Either the dynamics are described by a simple linear model or by a more complex one, and so the vector field is accordingly called linear or non-linear. In the following we focus on linear vector fields, because non-linear ones require different techniques. Both types can be generated by a CFD simulation and are very similar to each other, although non-linear ones can contain some special properties, e.g., shocks.

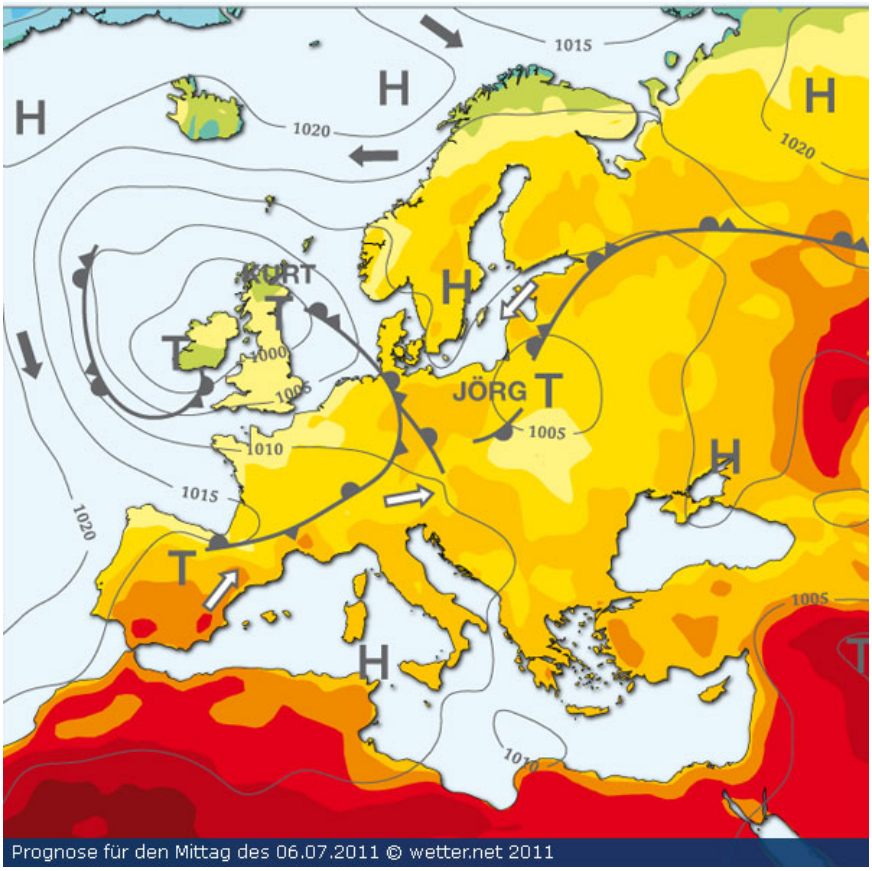


Figure 1.2: European weather forecast visualized by several different techniques (heat-map, glyphs, iso-lines, custom symbols) combined in one image [2].

Research in CFD focuses mainly on the development of new algorithms and solvers, but there are still very few published methods for analyzing the computed data [83]. With the increasing performance of every new computer generation, this need becomes more and more crucial because the newly generated vector fields are increasing either in terms of size or of resolution. And as shown in Figure 1.1, the knowledge extraction is the bottleneck of the complete workflow, because at this point the data volume is the largest and is processed completely manually. The actual knowledge extraction is performed by the user who is applying visualization techniques as tools for highlighting significant structures and trends. Therefore, most of the recent proceedings are contributed by the visualization community, which develops new methods and improves established ones. Yet even the newest approaches cannot overcome the very limited possibilities of the human eye [133], e.g., the individual perception of colors and the poor spatial resolution. Visualization is consequently

restricted to the two dimensional space, respectively to a pseudo three-dimensional one, if using advanced rendering and display equipment, e.g., oculus rift [1].

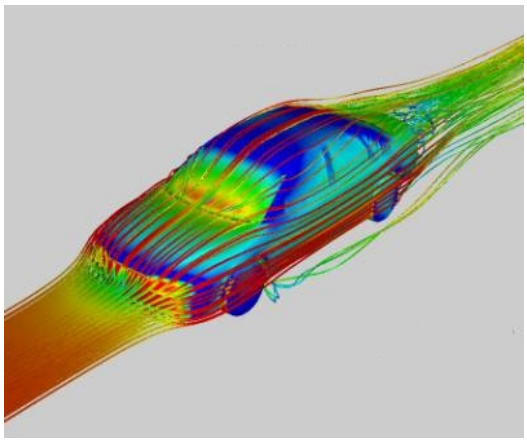
The weather over Central Europe is visualized in Figure 1.2. Basic visualization concepts are introduced in this simple example. The image encodes many different types of data visually and is composed of different layers containing certain information. The background is a geographical map of Europe helping the user to orient in the spatial space. In the next layer the temperature is color coded intuitively. Higher temperatures are represented by colors with a higher red component. The result is a so-called heat map. The next upper layer contains gray lines symbolizing zones of identical air pressure, so-called iso or contour lines. Along those lines the air pressure for the corresponding sensors have the same value. Consequently, the lines are dividing the vector field into two. Regions with lower pressure are indicated by a T and regions with higher pressure are labeled with an H. Every layer highlights a different feature of the vector field by a specialized technique. For instance the heat map conveys the information of the temperature distribution, which is actually not vectorial but scalar. It forms a scalar field, which can be regarded as a subset of vector fields. However, the example also contains a visualized vector field, too. The arrows, so-called glyphs, identify the wind blowing at the sampled points into the direction of the arrows head and the size of the arrows identifies the magnitude of the wind velocity. To complete the image's description, the heavier lines represent the different types of air masses, e.g., spikes identify warm fronts and semi-spheres representing cold fronts.

Although this example is very simple and familiar to the viewer, it demonstrates the abilities and restrictions of visualizations. A single image is composed of different layers, each conveys different features of the data set to the viewer. The more layers included the more complex the visualization becomes. Although the chosen example is familiar to all of us, some of the visualization entities used are not as well known to non-specialists, e.g., the symbols indicating the types of air masses. The symbols and colors must be chosen wisely, for example the symbols for the different air masses or the different colors for the wind field glyphs to increase the contrast. Therefore, the choice of the applied technique is as crucial to the quality of the visualization as the parameters used. In the regularly published state-of-the-art reports, the recent improvements and new techniques are summarized to keep the visualization community updated on those changes to guarantee a high quality of the visualizations [[109], [65], [89], [116], [21]].

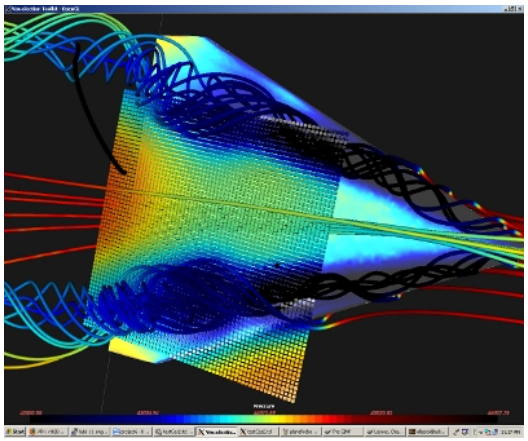
[1] Product homepage: https://www.oculus.com/en-us/

(a) Experimental path lines for a car in a wind channel [5]

(b) Manually placed path line to examine flow properties in distinct regions [5]



(c) Simulated path lines for a car model [8]



(d) Simulated path lines for a delta wing [1]

Figure 1.3: Application of path lines in wind channel experiments to visualize the flow properties around obstacles, e.g., cars.

Visualizing higher-dimensional data requires even more experience and knowledge in the choice of the visualization method and its correct settings to minimize the drawbacks and fully profit from the abilities of the visualization. The images in Figure 1.3 show experiments and simulations of three-dimensional objects in wind fields. In the upper row the classic wind channel experiments are shown in Figure 1.3(a) and Figure 1.3(b). Those images demonstrate the efforts, which are necessary to perform the experiments. First of all, a model must be built according to the blueprints, which is later actually put into the wind channel. Because the experiments are performed on an originally scaled model, i.e., the model has the same size as the original, the wind channel must be large enough to fit in a model of roughly three by two by two meters. Since the car is stationary in the experiments, the air must be accelerated to the simulated speed of the car. To visualize the air flow, smoke is brought in manually (Figure 1.3(b)) or in a grid (Figure 1.3(a)). The emitted smoke is blown by

the air around the car and indicates areas of high air resistance. If the experiments identify major weaknesses in the car's design, it is adapted, a new model is built and the experiments are re-run. This iteratively, experimental method used to be an important part in the development of new cars. Nowadays, those experiments are partly performed virtually, as shown in Figure 1.3(c). A 3D CAD model of the car is built according to the blueprints. This modeled car is placed into a dense wind field and only a few paths of air particles are highlighted, which produces the colored path lines. The path lines are close to each other, actually too close, causing the occlusion of the lines closer to the car's body. The seeding of the path lines is crucial to the quality of the visualization, because badly placed path lines are either occluding others or are placed in regions which are not interesting, e.g., too far away from the car's body. To overcome the problematic occlusion an expensive 3D virtual reality environment can be created, e.g., using head-mounted displays like oculus rift. Those systems provide the user with an illusion of a virtual reality in three dimensions. No matter which system is used to display the visualization, the seeding of path lines is very important to obtain an accurate and precise visualization [138]. This is also the motivation for the engineer in the upper right image to enter the wind channel himself and place the smoke source manually. Still, the visualizations are limited to the abilities of human perception. The lower right Figure 1.3(d) shows the path lines of a delta wing airplane. At first sight it looks very sophisticated, which is the result of combining several techniques here. First, the already mentioned path lines, which are color coded themselves. The second technique is cutting planes which select a subset of the complete data. Standard visualization techniques are applied on the subset selected by the plane. For instance in the delta wing example, the selected scalar field is color coded and indicates the pressure at the distinct points in space. If the visualization indicates a weakness in the design, it can directly be adapted in the CAD model and the simulation is instantly re-run without any further changes. Consequently the visualization focuses on regions of interest and significant structures and features of the flow.

1.2 Vector Field Features and Representation

So far, the features of the vector field have been visualized directly by drawing them into a new layer above the raw data, e.g., zones of high or low pressure. Other visualization techniques are more feature-specific and extract the features first to build a visualization based upon them [[68], [79], [109], [116]]. The feature-based visualiza-

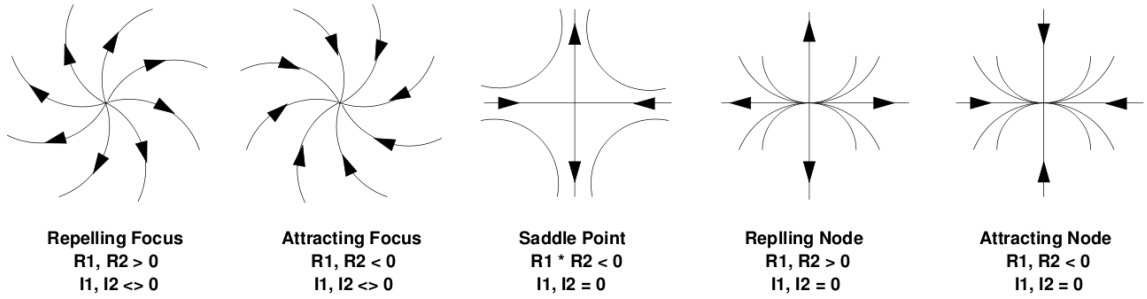


Figure 1.4: Critical points in vector fields. R1 and R2 denote the real parts of the Eigenvalues of the Jacobian. I1 and I2 denote the imaginary parts. Taken from Helman et al. [68].

tion techniques are more abstract and focus on conveying the flow characteristics of the vector field by its features.

The feature's definition of Helman et al. [68] is commonly cited and referred to for the description of features. To give the reader a first impression, the categorization is shown in Figure 1.4. The definition is based on the vector field's critical points, in which the flow is zero. Because the flow properties change at those critical points, they are defined as points of interest. There are many techniques for the feature extraction and feature-based visualization [[109], [116]]. Most of them focus just on enriching the visualization, but some actually include a second knowledge extraction step. Based on the features, a directed graph is constructed which represents the flow of the vector field [[132], [143], [144]]. The so-called topological skeleton is a highly sophisticated and complex representation of the vector fields topology. Because it is based on visualization entities, the representing graph is formed by critical points as the set of nodes and path lines connecting them as the corresponding set of edges. Therefore, it can be seen as a multi-layer representation of the vector field, rather than an abstraction of it.

Outside the visualization community it has been suggested that the complex, high-dimensional scientific data be abstracted for its analysis, e.g., Bailey et al. [15]. With the exception of Yang et al. [149], who denoted their work as an approach towards such an abstraction, the vector field analysis is still preformed visually. However, this abstraction would overcome many problems of the classic visualization techniques, because the representation is simplified without losing any information on the data.

1.3 Objectives

Within the traditional simulation workflow (Figure 1.1) the interaction between user and his visualization preforms the major knowledge extraction. The visualization highlights the significant structures from the acquired data and conveys them, such that the user can perceive them more easily. The actual knowledge extraction is performed by the user, who judges the visualization's output and, if necessary, improves or modifies it.

The combination of visualization and user interaction is the bottleneck of the workflow in several ways. First of all, the throughput of the complete workflow is limited by the user to his abilities in and knowledge of visualization techniques. Because the data is manually visualized by the user, his working speed dominates the overall execution time for the knowledge extraction. The iterative loop between visualization and user even amplifies this issue. Secondly, the user is not only the bottleneck for the workflow's throughput but also the limiting factor for its quality. Because of the significant user impact on the visualization, the visualized result reflects the user's subjective and personal perception of the data. Although guidelines for visualization exist [64], they are rarely applied or even intentionally ignored to emphasize the intended message [91]. Third, the significant user impact on the results prevents their comparison. Every user manually produces a unique visualization, which is biased by his personal preferences on visualization techniques and settings [124]. Lastly, in general visualization techniques create a qualitative analysis [83]. For instance, if filters, clipping planes and transformations are applied to improve the visualization, they transform the raw data and its numeric properties and are both either irretrievable or entirely lost. As a result, the efforts focus on the visual representation of the data rather than preserving its properties.

Our main objective is to break up the user-visualization loop by replacing the visualization within the traditional workflow with a more powerful element. To overcome the typical visualization problems, e.g., only a qualitative analysis is possible, we chose a feature extraction that is based on statistics rather than visualization. This allows a quantitative analysis, because all features are described in a uniform manner. A graph is constructed representing the input vector field and its dynamics based upon the extracted features. Since the features are precisely described by their properties, the graphs can be labeled with those properties. In particular, the contained spatial information of the feature's description is used to construct a label geometric graph, which we will define as flow graph. The flow graph is the major knowledge extraction

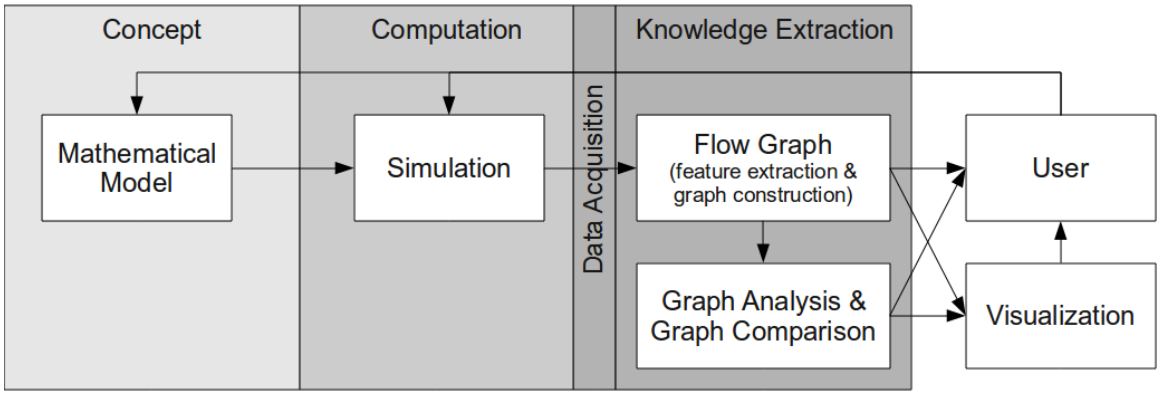


Figure 1.5: Novel simulation workflow. The iterative loop between the user and the visualization is replaced by an automated feature extraction and graph construction. The constructed flow graph is specified by its adjacency matrix, which can be read by the user or used for visualization. Additional analysis and comparison techniques might be included in the workflow to obtain more information. These are conveyed to the user either directly or via a visualization. The supplementary information gives insight to the vector field's evolution over several time steps or in alternative to the differences and similarities of different vector fields.

module in the proposed workflow, see Figure 1.5. The obtained knowledge is either conveyed directly to the user or is exploited to enrich a concluding visualization. The major difference between both workflows is that the visualization is not part of the knowledge extraction anymore, because in our workflow it just conveys the extracted information. This way the user impact on the analysis is minimized, increasing the workflow's throughput as well as its quality by guaranteeing an objective analysis.

The flow graph represents the vector field in a very abstract manner, but preserves the properties of the raw data at the same time. Both make the flow graph suitable for the application of techniques from graph theory, e.g., graph similarity. The flow graphs link the domain of visualization with graph theory. We consequently integrate a second knowledge extraction module into the workflow, which analyzes and compares the flow graph's properties. Again, the results are either conveyed directly to the user or exploited to enrich the visualization.

The combination of both modules forms a comprehensive and qualitative analysis workflow for vector fields. This presented workflow is constructed to process the complete output of the simulation without any supervision and in an efficient manner, which is not possible when applying traditional visualization techniques.

1.4 Challenges and Contributions

For the user the analysis is the key to understand the data. It is the actual knowledge extraction level between the data acquisition and the user perception. In contrast to common techniques, our approach is not based on visualization but uses data mining techniques. Although vector fields are well established, most of the research in their analysis is done in the field of visualization. Therefore, we need to transfer established definitions from one domain to the other, or even introduce new ones which are compatible to established ones of both domains.

- Because the presented analysis workflow is attached to a complete simulation workflow, the previous components need to be built, too. Thus individual values and parameters can be modified in the simulation and observe their propagation through the workflow and their effect on the result can be observed. For this reason three CFD test cases or scenarios were constructed which are used throughout this work to demonstrate and validate the findings. The generic backward step is the simplest case. It is fully generic and ground-truth information exists for it. The other two cases, the backward step and the lid-driven cavity are real simulations which are well known and precisely described in theory. This makes it possible to validate the simulation workflow before the actual knowledge extraction is performed. Furthermore, the extracted information can also be compared to the anticipated appearance and characteristics of the vector field. For each test case, several simulations were run to produce a sequence of vector fields with different properties.

- Vector fields as large, high-dimensional data sets are formed by thousands of data points. To manage the complexity in dimension and size, the analysis must include a simplification or abstraction step. This is typically done by extracting so-called features. Features are the interesting points, regions or structures inside the vector field which describe its most significant properties. The statistical feature extraction, which is new to vector fields, must be compatible with established techniques to compare the results and to be of practical use. On the other side, the definition must fit the constraints of automated extraction techniques, which require a precise description of the feature's properties. Therefore, a precise but general feature definition must be found which is compatible to the established ones.

- Although many different visual feature extraction techniques exist for vector fields, none is comparable to our approach. Therefore, the presented approach needs to be compared to the existing ones and it must be demonstrated that it performs better in terms of both quality and efficiency. For this comparison, user guided techniques must be compared with automated approaches, which is very difficult, because of their totally different designs.

- Many automated feature extraction approaches exist. They need to be evaluated and the one, that best fits our context must be selected. Vector fields are not loosely sets of data points. The defined features expose specific properties. Both must be considered in the choice of the feature extraction approach. Additionally, the chosen algorithm must be adapted to this particular context, e.g., the optimal number of extracted features representing the data set must be evaluated.

- For the construction of the flow graphs, the extracted features must be set in context to each other. This context information is not initially encoded in the raw data and must be derived or generated first. Because no comparable algorithm exists, a proof of concept must validate the correctness of the developed technique.

- The setting for the graph analysis is similar to the feature extraction one. Over decades many efficient algorithms were published which compare graphs by their structures and properties. For our specific flow graph, one technique must be chosen and its correctness for this specific setting must be validated.

We address these aspects one-by-one and present a qualitative analysis workflow for vector fields. We link the traditional domains of visualization with graph theory by the introducing of flow graphs, which are the key-stone in our analysis. Furthermore, we are the first to use this link to transfer techniques from one domain to the other one. Because we established this novel link, we need to set common definitions which are valid in both settings. Additionally, the transfer must be implemented carefully, because of the varying properties of graphs and vector fields. The statistical feature extraction, which is new to vector fields, is a major improvement over the established techniques. Also the novel flow graph representations significantly improves the visualization of vector fields by overcoming most of the visualization's drawbacks. Not only the visualization profits from the graph representation, we also improve

the analysis by including graph analysis techniques. Furthermore, our workflow is un-supervised which addresses the steadily increasing complexity of simulations.

1.5 Thesis Outline

This thesis is structured as follow:

- **Chapter 2** gives an overview of the relevant related work and the essential definitions. It is split into three parts. First, discrete vector fields and their features are introduced, which are the main subjects of this work. Second, different extraction methods for these features are introduced. We distinguish between interactive visual analysis and automated data analysis. Traditionally, vector fields are analyzed visually, i.e., filters and specialized techniques highlight the significant structures to make them stand out to the human eye. Established visualization techniques are summarized to give readers who are new to visualization an overview. Those visualization techniques will be used throughout the complete thesis to show and evaluate the findings. Research on vector fields began with the foundation of modern mathematics and the domain of flow visualization is dedicated to them, so we focus on the most relevant and established, traditional techniques. The second group of feature extraction techniques is formed by the automated approaches. Because the used data is formed by numeric values only, we also focus on the relevant work on it, namely cluster analysis. The third part of this chapter introduces graphs as a high-level representation for complex structures and relations. We assume the reader is familiar with the basic notation of graphs and thus keep their description brief. The focus is on the similarity of graphs and their comparison to complete the overview, because additional and more specific related work will be provided in the respective chapters.

- **Chapter 3** describes the feature extraction in detail. Before the actual extraction, the feature vector is defined. It precisely describes the properties of the vector field's elements and it is used to separate those elements into groups which are the vector field's features. This definition is the necessary requirement for the application of the cluster algorithm, which performs the actual extraction. The concluding evaluation focuses on the data pre-processing and the appropriate clustering algorithm. Several different algorithms of different categories are tested on the so-called lid-driven cavity data set. In this chapter

the data set is introduced along with current visualization techniques to which our proposed approach is compared. Finally, we discuss the advantages and disadvantages of our approach as oppose to current visualization techniques.

- **Chapter 4** links vector field analysis, which is a typical visualization problem, to graph theory. We introduce geometric flow graphs which enable a comprehensive and quantitative description of vector fields according to its features. Because no comparable techniques have been published, we provide a proof of concept before the algorithm is tested on the lid-driven cavity and the backward-step. A runtime analysis is also performed on these data sets to demonstrate that the throughput of the analysis workflow can be increased by the algorithm.

- **Chapter 5** applies graph comparison techniques to the extracted flow graphs. This demonstrates the new possibilities in understanding the simulation and its corresponding vector fields with our flow graphs. Because of the increased throughput of the analysis pipeline, the user is now able to examine the complete set of vector fields produced by a simulation. This is a very challenging and time-consuming task, if the visualizations are manually produced for every vector field individually. Furthermore, the evolution of vector fields gives insight to the simulation itself by describing the appearance and changes of the vector field for every time step. The graph representation also allows the comparison of vector fields to provide the user with additional information on the performed simulation and in particular on the differences between two different simulations.

- **Chapter 6** summarizes the complete thesis and our presented workflow from pre-processing and feature extraction to possible applications for our technique. Additionally, open issues for further studies are discussed.

Chapter 2

Background and Related Work

The goal of this work is to derive a graph representation for vector fields and to link visualization with graph theory. Traditionally, vector field analysis is part of flow visualization, simply just flow vis, which is again part of visualization. On the other side, graphs are a field of their own and graph theory provides powerful tools for graph analysis and comparison. Combining both domains allows the application of graph theory techniques to vector fields, which improves their analysis by providing precise and quantitative graph descriptions. Furthermore, the constructed flow graphs enable a comparison of the vector fields.

Visualization techniques focus on significant structures or features of the vector field to highlight them and make them easier to perceive for the user. The feature extraction reduces the complexity, because the complete data set, which consists of several thousand points, is reduced to just a couple of features. Over the last decades highly specialized visualization techniques have been developed, each of them is specific to one property or feature of the vector field. If several properties or features are to be conveyed, the visualizations must combine different techniques or even several individual visualizations must be built to represent the data set comprehensively. Because the human visual system is the key visual knowledge extraction, these techniques suffer from the limitations of the human eye, e.g., poor spatial resolution and biased perception of colors. In order to keep up with the increasing model complexity of the simulated data sets, the modern visualization techniques are becoming more and more complex, which overcharges the abilities the user's visual system.

Additionally, intensive user interaction is required in the construction of visualization pipelines. To reduce the user interaction, which consequently reduces the human bias and increases the throughput of an analysis workflow, the presented approach is based on an automated data analysis applying statistics rather than visualization

techniques. As the result of our workflow, the vector field is represented by a geometric flow graph. The graph not only provides a simplified and abstract representation of the input vector field, but also enables the comparison of vector fields via their geometric flow graphs. So far, comparing two vector fields is a very challenging task, because two individually manufactured visualizations need to be compared. Both of these visualizations are heavily influenced by the user during their construction, which further increases the difficulties in comparing them. The comparison is based on the properties of the applied visual entities and techniques rather than the initial vector field data.

This chapter is divided into four sections. First, vector fields and their features and properties are defined. Since vector fields are the input to the workflow, they are the starting point. Their structure and origin is described in order to understand their appearance and properties, which includes the features and significant structures. This first part is concluded by the definition of features (Definition 2.6), which is the transition to the feature extraction in part two. We distinguish between two different approaches, the user-guided visualization techniques and automated statistical approaches. The third part focuses on graphs and geometric graphs derived from the extracted features and are the key point of our feature-based flow graph representation. In the fourth part, we summarize classic graph matching techniques, which are necessary to compare graphs with each other and allow a quantitative analysis of vector fields using their geometric flow graphs. The chapter is finally concluded by a summary and a discussion of the previous parts.

2.1 Vector Fields

Vector fields represent complex, high-dimensional data in research and industry. Every element in vector fields has a distinct location in spatial space and an attached high-dimensional vector. For instance in the daily weather forecast, every location is given by its latitude and longitude. Based on their location in the simulation, the data vector for every pair of latitude and longitude is computed. As a result every point of the map is labeled by a data vector. The complete simulation pipeline consists of three major components, which were previously shown in Figure 1.1. A set of equations describes the system mathematically, while the actual simulation solves the equations with the given input parameters to obtain the results, which are then visualized.

Definition 2.1. *Simulation*
The model $\mathcal{M} := \{\mathcal{E}, \mathcal{P}\}$ *describes the behavior of the system using the set of equations* $\mathcal{E} := \{\mathcal{E}_1, ..., \mathcal{E}_n\}$ *and the set of parameters* $\mathcal{P} := \{\mathcal{P}_1, ..., \mathcal{P}_m\}$. *Executing the simulation* $\mathcal{S} := \{\mathcal{M}, T\}$ *results in a sequence of discrete vector fields* $\mathbb{V}_0, ..., \mathbb{V}_t$ *for the specified reference frame and the finite time interval* $T = [0; t]$ *consisting of* t *time steps.*

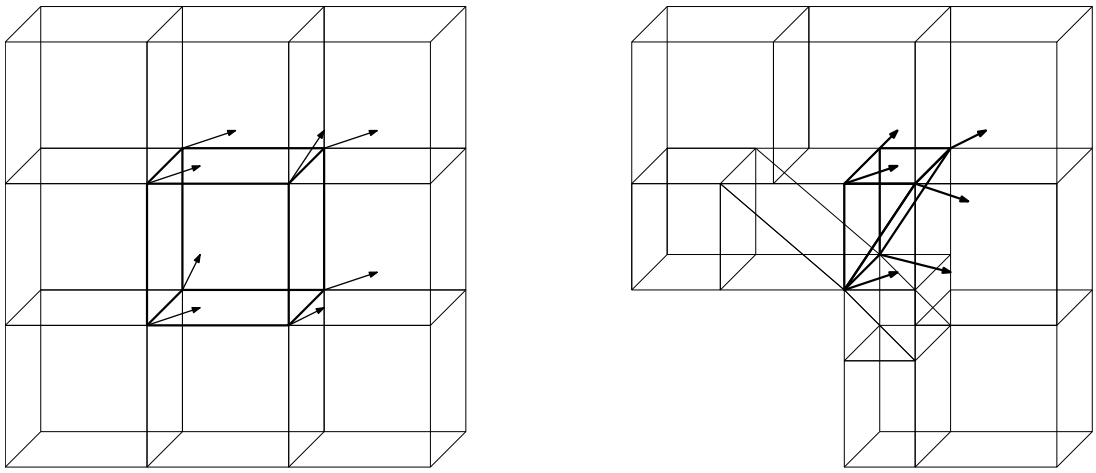
The definition clearly distinguishes between a mathematical model $\mathcal{M}$ and the simulation $\mathcal{S}$. The simulation's settings define the reference frame which specifies the scaling and translation of the simulation. As an example, the car body inside an air-stream is a system in which the mathematical model $\mathcal{M}$ describes the shape of the car and the properties of the air by the set of parameters $\mathcal{P}_1, ..., \mathcal{P}_m$ and the flow of the air around the car by the set of equations $\mathcal{E}_1, ..., \mathcal{E}_n$. For each of the t time steps, a discrete vector field $\mathbb{V}_t$ is computed by the simulation as the result of the equations and parameters solved by a numerical solver.

Definition 2.2. *Discrete Vector Field*
Given a subset of points $\mathbb{V} = \{\vec{x}_1, ..., \vec{x}_m\} \subset \mathbb{R}^n$, *the discrete vector field is represented by the vector-valued function* $\gamma : \mathbb{V} \to \mathbb{R}^n$, *which computes a corresponding data vector* $\gamma(\vec{x}_v) \in \mathbb{R}^n$ *for every data point* $\vec{x}_v \in \mathbb{V}$.

Vector fields represent various physical processes of different domains and therefore the data vectors $\gamma(\vec{x}_v)$ refer to shear forces, gradients, etc. In the following the focus is on flow fields that are the result of computational fluid dynamics. The vectors $\gamma(\vec{x}_v)$ are consequently named velocity vectors. The set $\mathbb{V}$ consists of a finite number of points and is a discretization of the continuous set $\mathcal{V} \subset \mathbb{R}^n$. Cells are constructed to approximate the continuous counterpart of the discrete vector field more accurately.

Definition 2.3. *Cell*
Every data point $\vec{x}_v \in \mathbb{V}$ *is member of at least one cell* $\mathcal{C}_v \in \mathcal{C}$, *in particular the data points are the corner points of the cells. Every cell* $\mathcal{C}_v$ *represents a closed set and the inside can be interpolated by the corner points. The set of cells* $\mathcal{C}$ *covers the complete vector field* $\mathbb{V}$.

The cells' shape and appearance depend on the number of its corner points which form the cell. Therefore, cells may be any kind of polygon, also concave cells are possible. The cell $\mathcal{C}_v$ is interpreted as a dense set of points, while $\mathbb{V}$ is a discrete set of points, i.e., there are points $\underline{\vec{x}}_w \in \mathcal{V}$ and $\underline{\vec{x}}_w \notin \mathbb{V}$. Through the introduction of cells the data vectors $\gamma(\underline{\vec{x}}_w)$ can be approximated by the data vectors $\gamma(\vec{x}_c)$ of the



(a) Hexahedral cell in a structured grid (b) Triangular cell in an unstructured grid

Figure 2.1: Neighborhood of a cell (simplified). The velocity vectors of the cell's corner points are visualized by glyphs.

surrounding cell's corner points $\vec{x}_c$. By Definition 2.3 the discrete vector field not only approximates a continuous one, but the data points are set into context $\vec{x}_v \in \mathbb{V}$ to each other. Therefore, the vector field is not a set of loose points. This has to be considered while extracting the features. In Figure 2.1 two sets of cells covering a vector field are shown. If the set of cells $\mathcal{C}$ consists only of cells formed by regularly distributed data points it is called a *structured grid*. It is called an *unstructured grid* if the cells in $\mathcal{C}$ are different from each other. Structured grids are less common in applications than unstructured grids, because if more complex geometric objects are included in the simulation, the object can be embedded in an unstructured grid mores easily than in a structured one. The latter have the advantage of an increased resolution in regions of interest, which results in a decrease of the computation error in these areas.

Although higher dimensions are not excluded, $\mathbb{V}$ is typically a subset of $\mathbb{R}^2$ or $\mathbb{R}^3$ and is formed by thousands to several millions of cells. The construction of grids for a given vector field is a research domain of its own [102], [56]. Most simulations use either a triangulation of the vector field, because it can be constructed for any geometry, or grids formed by hexahedrons, which intuitively map the vector field to the Euclidean space.

The Definitions 2.2 and 2.3 are based on data points $\vec{x}_v$ as the most atomic level of data representation. As a first step to extract regions of interest rather than points of interest, which will be our overall goal, the representation of the vector field is changed from a point-centered towards a cell-centered one, in which cells are regarded as the

most atomic level. Because of the Definition 2.3, the region of the vector field that is covered by the cell can be linearly interpolated. Therefore, the properties of the corner points can be transferred to the cell's center by interpolation.

Definition 2.4. *Cell-Centered Vector Field Representation*
A vector field is represented by the cells of its grid. Every cell $\mathcal{C}_v \in \mathcal{C}$ *has its center at its center of mass* $\vec{x}_c = \frac{1}{p}\sum_{i=1}^{p} \vec{x}_i$ *defined by the cell's* p *corner points. The data vector for the the cell center* $\gamma(\vec{x}_c)$ *is interpolated by the data vectors* $\gamma(\vec{x}_v),\ v = 1, ..., p.$

Comparing the initial Definition 2.2 with Definition 2.4, the representation of the vector field changed from a loose set of discrete points to a closed, continuous representation. This will be of use for the feature extraction and in particular for the vector field's graph representation.

2.2 Feature in Flow Fields

Since the analysis of vector fields is based on visualization, which is restricted in its effectiveness and power by the limitations of the human eye, the user must be guided to the most significant structures inside the vector field. These structures, independent of their appearance or properties, are called features. In this section, the properties and different extraction techniques of features are the main subject.

Zeros of a function are the key aspects in examining the properties and the function's behavior. In these points the function's appearance changes, e.g., zeros in the first derivative indicate an extrema of the function. In mathematical analysis the zeros are regarded as the significant points to describe the function, because the function value vanishes here.

Definition 2.5. *Critical Point*
For the critical point $\underline{\vec{x}}_c \in \mathcal{V}$ *the magnitude of vector-valued function* $\gamma(\cdot)$ *is zero,*

$$\|\gamma(\underline{\vec{x}}_c)\| = \sqrt{\underline{x}_{c,1}^2 + ... + \underline{x}_{c,n}^2} = 0. \tag{2.1}$$

Analogously to vector-valued functions, critical points form the key aspect for the dynamics and the changing behavior of vector fields. Unfortunately, the critical point $\underline{\vec{x}}_c$ is not necessarily an element of the discrete set $\mathbb{V}$, but it is an element of the set $\mathcal{V}$, which is approximated by $\mathbb{V}$.

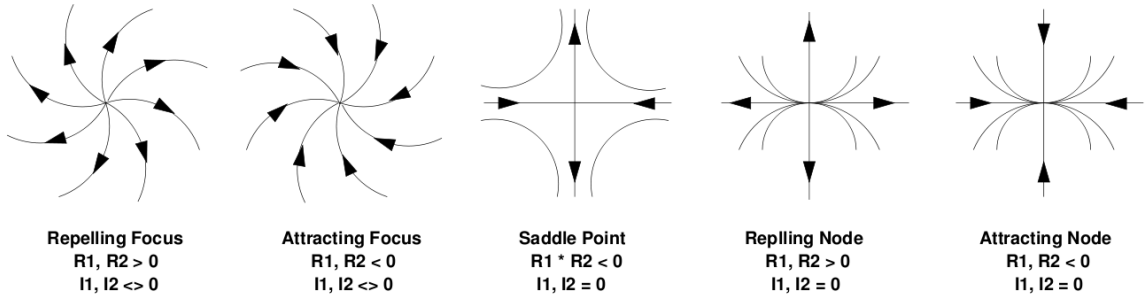


Figure 2.2: Critical points in vector fields. R1 and R2 denote the real parts of the Eigenvalues of the Jacobian. I1 and I2 denote the imaginary parts. Taken from Helman et al. [68].

The first approaches and techniques that were particularly developed to understand vector fields go back to the beginning of modern mathematics. In 1858 von Helmholtz [69] published the theory on vector field decomposition, which is still the basis for recent work on vector field's feature. The Helmholtz-Theorem states that the overall vector field can be decomposed into different components, namely its *divergence* and *rotation*. Significant points are critical points in which either the divergence or the rotation component of the vector field is zero, i.e., the critical point is a source or a sink or a rotation center, respectively. This theorem is still applied in recent work to identify significant points of vector fields [[107], [60]].

Helman et al. [68] identify the critical points by the partial derivatives of the flow at every given point of the vector field. By calculating the Eigenvalues of the Jacobian they can furthermore categorize the singularities and their influence on the flow field, as seen Figure 2.2. They describe repelling or attracting focus for a vanishing rotation component and repelling or attracting node for zeros in divergence. Because of the derivation via the Eigenvalues of the Jacobian, these features are called the first-order singularities. Scheuerman et al. [119] extend the definition to also include singularities of higher orders, i.e., singularities in the second or higher derivative. Still, these singularities are found at the critical points. In non-linear vector fields *shocks* are additionally defined, in which the function $\gamma(\cdot)$ is no longer linear anymore [38].

The feature categorization of Helman et al. [68] is widely accepted and applied by the flow visualization community. To obtain this categorization, different techniques exist which are based on different approaches. These approaches not only describe the singularities, but they also extract the critical points. The classic differential approach was already published by Helman et al. [68]. The description is based on

the computation of the partial derivatives, the Jacobian matrix and its Eigenvalues. The Jacobian matrix for the function $f(\cdot)$ contains the spatial derivatives in multidimensional space at point $\vec{p}$ and is defined as:

$$J_f = \begin{pmatrix} \frac{\partial f_1(p)}{\partial p_1} & \frac{\partial f_1(p)}{\partial p_2} & \cdots & \frac{\partial f_1(p)}{\partial p_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m(p)}{\partial p_1} & \frac{\partial f_m(p)}{\partial p_2} & \cdots & \frac{\partial f_m(p)}{\partial p_n} \end{pmatrix} \tag{2.2}$$

with $\vec{p} = (p_1, p_2, ..., p_n) \in \mathbb{R}^n$. The partial derivatives are typically computed by the central difference quotient in every dimension before the Eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_n$ are calculated by standard numerical methods. For $n = 2$ the singularities are categorized as in Figure 2.2.

The given set of singularities can also be identified by applying complex analysis. An integral along a closed curve γ is computed. If the inside of the curve contains a singularity, the integral is unequal zero.

$$ind_p\ f = ind_\gamma\ f = \frac{1}{2\pi} \oint_\gamma \frac{f_1\ df_2 - f_2\ df_1}{f_1^2 + f_2^2} \tag{2.3}$$

The Poincaré-Hopf index ind_p [99] is applied as singularity counting index. Scheuerman et al. [119] presented this approach to extract first-order singularities but also higher-order singularities. Another variational approach was published by Polthier et al. [106] and Guo et al. [60]. They apply the Hodge-Helmholtz Decomposition for discrete vector fields. Figure 2.3 shows an exemplary decomposition.

The approach of Elbing et al. [46] extracts predefined structures S_n, e.g., the previously mentioned singularities, and convoluts them with the vector field V to find possible occurrences and positions of the predefined patterns inside the field [22],

$$s_n(r) = \int \int \int_\Omega < S_n(\xi), V(r - \xi) > d\xi \tag{2.4}$$

Indeed, the structures S_n can also be defined differently, for instance Wang et al. [141] propose matching pre-defined trajectories to find correspondences inside the vector field. This indicates the disadvantage of including only critical points in the analysis, which is focused too narrowly on points of interest. In the following section, especially in the paragraph on feature-based visualization, regions of interest are

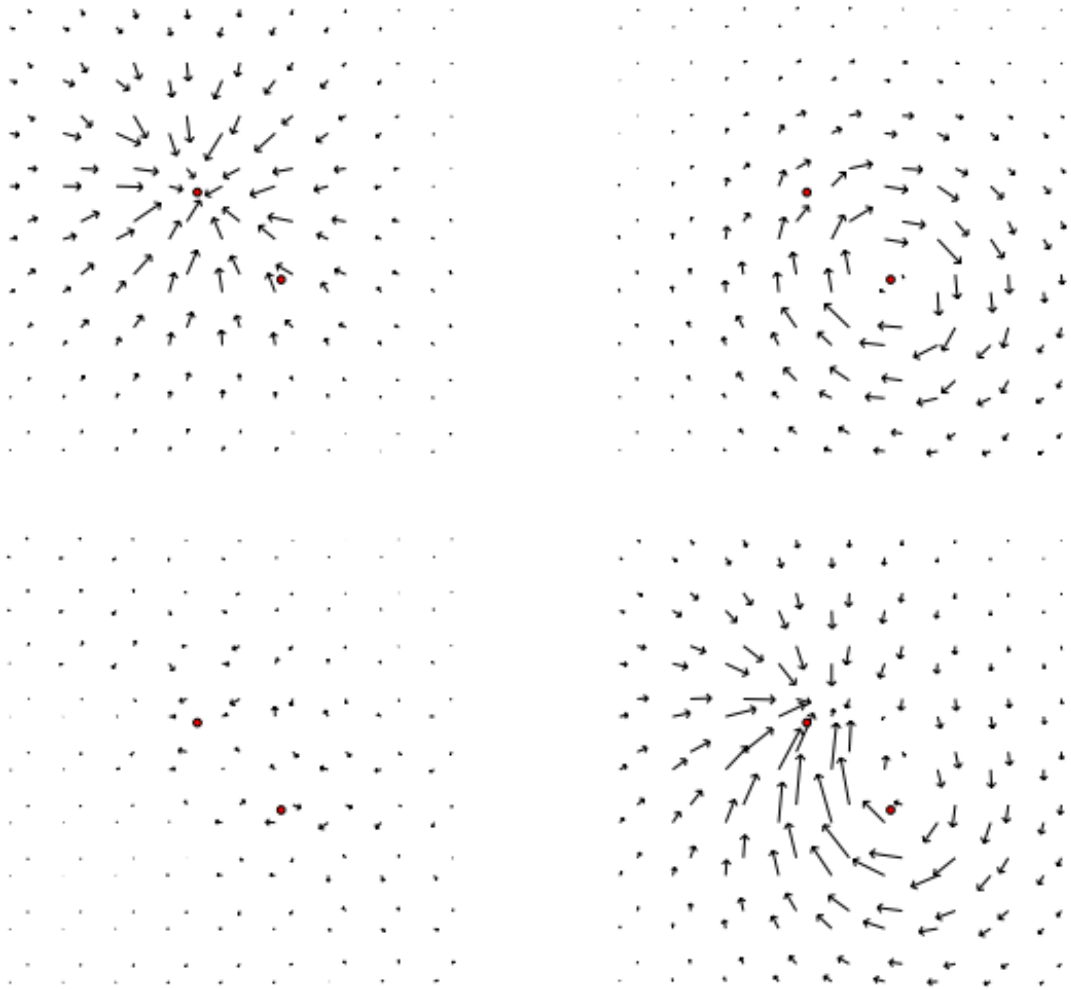


Figure 2.3: Decomposition of test vector field (bottom right) in rotation-free (upper left), divergence-free (upper right) and harmonic component (bottom left). The two dots in each image indicate the centers of the original potential. Notice, that the dots in the combined vector field do not seem to lie in the centers indicated by the vector field, although they do. But the components clearly recover the original centers. Taken from Polthier et al. [106].

examined rather than points of interest. Therefore, in our definition of features both concepts are unified.

Definition 2.6. *Feature*
A connected set of cells $\{\mathcal{C}_a, \mathcal{C}_b, ..., \mathcal{C}_k\} \subset \mathcal{C}$ *forms one feature* $\mathcal{F}$*, if, and only if, the cells* $\{\mathcal{C}_a, \mathcal{C}_b, ..., \mathcal{C}_k\} = \mathcal{F} \subset \mathcal{C}$ *expose similar properties in terms of their data vectors* $\gamma(C_a), \gamma(C_b), ..., \gamma(\mathcal{C}_n)$.

The definition only assumes the connectivity of the cells, i.e., $\forall\ \mathcal{C}_i, \mathcal{C}_j \in \mathcal{F}, \exists\ \vec{x}_v \in \mathcal{C}_i$ and $\vec{x}_v \in \mathcal{C}_j$ and the cells share at least one common point. This assumption does not restrict the shape of the features, and especially includes concave features. The feature's size can differ between one individual cell to all cells of $\mathcal{C}$. This includes points of interest like critical points, which are inside one cell, as well the possibility of regarding the complete vector fields as one huge individual feature. In this particular context of computational fluid dynamics both extremes are possible. In regions including critical points the flow changes significantly, but for the majority of cells the flow is homogeneous and the velocity vectors are identical for adjacent cells. These regions of homogeneous flow are intentionally set up like this to prevent accidental side effects from the outside on the inner points of the vector field.

2.3 Feature Extraction

Extracting features from scientific data sets is the first step towards understanding the data. Significant points or structures of the data set which are relevant to the user are identified and this step can be seen as the first step of knowledge extraction after the data acquisition. By extracting the features the input data is segmented into two sets: features and non-features. This segmentation allows the user or a concluding algorithm to focus on the interesting feature subset of the data. The resulting dimensionality reduction helps the user to concentrate on the relevant data and is a pruning for concluding algorithms.

Most extraction techniques are specialized for one specific scientific context, e.g., keyword extraction [76] for corpora and feature extraction for face recognition [151]. For each of these applications, the extraction technique is developed to find the significant properties of the features which separate them from the non-features.Therefore, a categorization based on the extraction strategy or the extracted feature property is not possible. Because of this, we categorize the different techniques by the amount of user interaction that is necessary for their execution.

Interactive Visual Analysis	Automated Data Analysis
+ user-guided analysis possible	- needs precise definition of goals
+ detects new features w/o looking for them	- limited tolerance of data artifacts
+ understands results in context	- results without explanation
+ uses power of human visual system	- computationally expensive
- human involvement not always possible	+ hardly any interaction required
- limited dimensionality	+ scales better w.r.t. dimensions
- often only qualitative results	+ precise results
- often unfamiliar	+ long history of application

Table 2.1: Advantages (+) and disadvantage (-) of feature extraction techniques. Adapted from Kehrer et al. [81].

A fully user-guided analysis has numerous of advantages over fully automated unsupervised techniques, but also some disadvantages. In Table 2.1 both are summarized. In general, interactive analysis methods strongly rely on their visualization component in the workflow, which conveys the information from the data to the user. The visualization is the main knowledge extraction layer here, because it is easier for the user to interpret the data visually than to understand the raw numeric values. The human visual senses are very powerful and are able to extract features and trends of different appearances and properties. This ability, paired with human creativity, make user-guided approaches superior in their flexibility over any automated techniques. Automated data analysis requires strictly predefined features and structures and is consequently less flexible. Because the unsupervised techniques are run fully automatically, the throughput is higher than for any technique that includes human interaction. The human visual system is powerful, but can be tricked easily [133], because it performs a qualitative analysis rather than a quantitative one. Additionally, automated data analysis scales better with respect to the dimensions of the data set, in particular regarding model complexity. In contrast to that, the human visual system is limited to two dimensions and a limited set of colors.

In the following, representatives of both categories are selected by their applicability to our specific context. This is necessary, because a more general overview would be excessive, in particular for the chosen interactive visual analysis techniques. The feature definition 2.6 was kept very general in order to be applicable to many different automated approaches.

2.3.1 Interactive Visual Analysis

The key component for the knowledge extraction is a visualization method, i.e., the overall goal of the techniques is to make structures inside the vector field visually more distinct. The quality of the results is judged by the user building the visualization pipeline and, if necessary, modifying the pipeline and tuning it to improve the output. This iterative improvement of the visualization is the so-called visualization loop [64].

Over the decades this loop has been expanded by techniques to highlight many different features inside the vector field. Hansen et al. [64] summarized established techniques for different visualization techniques, but also Hauser et al. [65], Laramee et al. [89], Salzbrunn et al. [116] among many others, too. All of these surveys give an overview of the visualization techniques in general and flow visualization in particular. Despite their large numbers, many of the mentioned techniques are found in most of the surveys. We chose the categorization by Salzbrunn et al. [116], shown in Figure 2.4, which is probably the most intuitive categorization for readers not familiar with visualization.

Here, the visualization is the knowledge extraction layer between the data acquisition and the user perception. Again the main purpose of the visualization is to point out and to work up the data, s.t., the user can understand it easier. The mentioned categories become more and more abstract from left to right, i.e., the techniques summarized in the direct visualization group are more intuitive than the classes to its right. On the other hand, the class of the less abstract visualizations requires a lot more experience from both the researcher building the visualization and the viewer understanding the results.

Direct visualization

This class contains the simplest and also oldest visualization techniques. Properties of the vector field are simply drawn into it, i.e., the vectorial property is symbolized by an arrow or glyph to identify its direction and its magnitude. The resulting visualization is familiar to all of us and can be seen in the daily weather forecast to represent the wind field. In Figure 2.5 the direct visualization methods are combined for an exemplary vector field indicating the air flow around a car's body.

Definition 2.7. *Glyph[29]*
A glyph is the visual representation of a piece of data where the attributes of a graphical entity are dictated by one or more attributes of a data record.

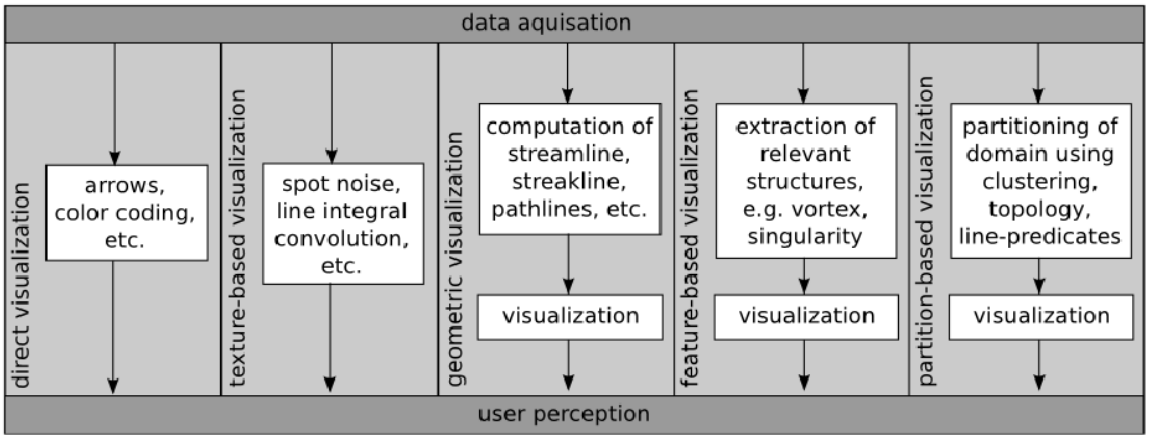


Figure 2.4: Flow visualization techniques. The visualization forms a knowledge extraction layer between data acquisition and user perception. From left to right, the techniques are becoming more and more complex. While the class of direct visualization techniques utilizes simple visual entities, e.g., arrows or glyphs, the feature-based visualizations are built on a complex definition of features or relevant structures. Taken from Salzbrunn et al. [116].

In regions of the vector field in which many data points are located, glyphs tend to occlude each other. This is solved by sampling the data points and adding glyphs to this subset. Since glyphs are very common, there are huge variations of this visualization technique. Yet none of these can overcome the disadvantages of glyphs, which heavily suffer from occlusion and therefore are restricted to two dimensions mainly. Borge et al. [21] summarize the state of art for glyph visualization.

The actual feature extraction is performed by the user, who specifies the parameters for the glyphs, e.g., the sampling rate and their placing inside the date. He examines the vector field by the visualization and improves the parameters. Through this iterative and manual improvement of the visualization the features inside the vector field become more and more visually distinct. The quality of the result is nevertheless dependent on the experience of the user who sets the parameters appropriately. Even very experienced users can oversee trends and features inside the vector field while they are exploring it and setting these parameters. Recent works propose the glyphs as an input for a concluding clustering [67]. The clustered glyphs provide a better overview of the overall data and reduce the likelihood of occlusion, making it less likely to oversee features. Instead of using the underlying data of the vector field, a similarity measure for the glyphs is introduced. This measure geometrically computes the differences of the magnitudes and the orientations of the glyphs.

Therefore, the clustering is solely based on the glyph's properties and works as a simplification step.

Another technique belonging to this class is color coding, in which the properties of the data are represented by colors. It requires mapping from spatial space, in which the vectorial values are defined, into a color space [128]. Still, this technique lacks resolution, because the viewer's eyes may not be capable of resolving all colors correctly. Therefore, color coding is commonly used for scalar values only. The same is true for contour lines, which also are applied mainly to scalar fields.

Definition 2.8. *Contour line[39]*
A contour line of a function of two variables is a curve along which the function has a constant value.

For a contour line, a fixed threshold t is defined. For every pair of adjacent cells $c_i, c_j \in \mathcal{C}$ and their scalar data $\gamma(c_i), \gamma(c_j) \in \mathbb{R}$, the cells c_i, c_j are separated by the contour line, if $\gamma(c_i) < t < \gamma(c_j)$, or $\gamma(c_i) > t > \gamma(c_j)$, respectively. As a result the vector field is segmented into two sets, one in which the elements have scalar value larger than the threshold and one in which they are smaller. These sets do not need to be coherent and therefore the vector field is possibly segmented into several parts. A scalar value must be extracted from the vector field as a pre-processing step. This is commonly the magnitude of the velocity vector, i.e., the velocity vector is projected from the vector field's domain into the lower-dimensional $\mathbb{R} \subset \mathbb{R}^3$. Through this projection all information about the divergence and the rotation is lost, even the direction of the flow can no longer be retrieved anymore.

Contour surfaces can be defined for higher dimensions, but those are commonly nested and therefore they occlude each other. Despite their disadvantages, contour lines are a very common technique to get a first impression of the data set and are very intuitive, e.g., they are applied in every daily weather forecast to represent the isobars or the temperature distribution.

Texture-based visualization

These techniques portray the effect of flow on the vector field. The input vector field's flow is applied to a known texture and the result is analyzed. The vectorial component of the vector field morphs the texture and consequently identifies the properties of the vector field. Wijk [136] proposed the usage of stochastic texture, but other structures are possible, too [89]. Texture-based visualization techniques are of particular interest for demonstrating the effect of the vector field on surfaces.

For higher-dimensional data these techniques are hardly applicable and instead of textures geometric objects are used. Diffusion tensor visualization [146] can be seen as a combination of texture-based and geometric visualization, in which predefined geometric objects, e.g., ellipsoids are exposed to the vector field's flow.

Geometric visualization

Geometric visualization is also a class of very intuitive and classic methods to gain understanding of the vector field. In contrast to the direct visualization techniques more sophisticated geometric entities are derived and used here.

Path lines are regarded as a high-dimensional extension of glyphs. Instead of showing the velocity vectors for a sample of points, an even smaller sample is selected and the velocity vectors are connected. As a result, a trajectory through the vector field is obtained starting at the one of the sampled points.

Definition 2.9. *Path line[49]*
For a seed $\vec{x}_s$ the path line $pl(\vec{x}_s)$ is given by

$$pl(\vec{x}_s) := \begin{cases} \dfrac{d\vec{x}_t}{dt} = \gamma(\vec{x}_t) \\ \vec{x}_t = \vec{x}_s \end{cases} \tag{2.5}$$

and iterated over time t, for which the resulting $\vec{x}_t \in \mathbb{V}$.

The quality of the visualization heavily depends on the seeds, however seeding the path lines is still an open research question [138]. The seeding requires an experienced user, who places the points properly in order not to overlook interesting regions in the vector field. However, computing the path lines for large samples and long time steps, i.e., very large t, is very costly and results in a visualization with occluding entities. Nevertheless, well-chosen settings path lines give a very detailed impression of the vector field, which is also intuitive at the same time.

Feature-based visualization

In contrast to the previously described classes, feature-based visualization is motivated by the explicit search for pre-defined features. Such features can be formed by critical points or by more complex geometric structures. Lugt [95] describes vortexes as features of vector fields. The vortexes are parts of the vector field which rotate around an axis. In general, a more abstract description of vector fields is possible

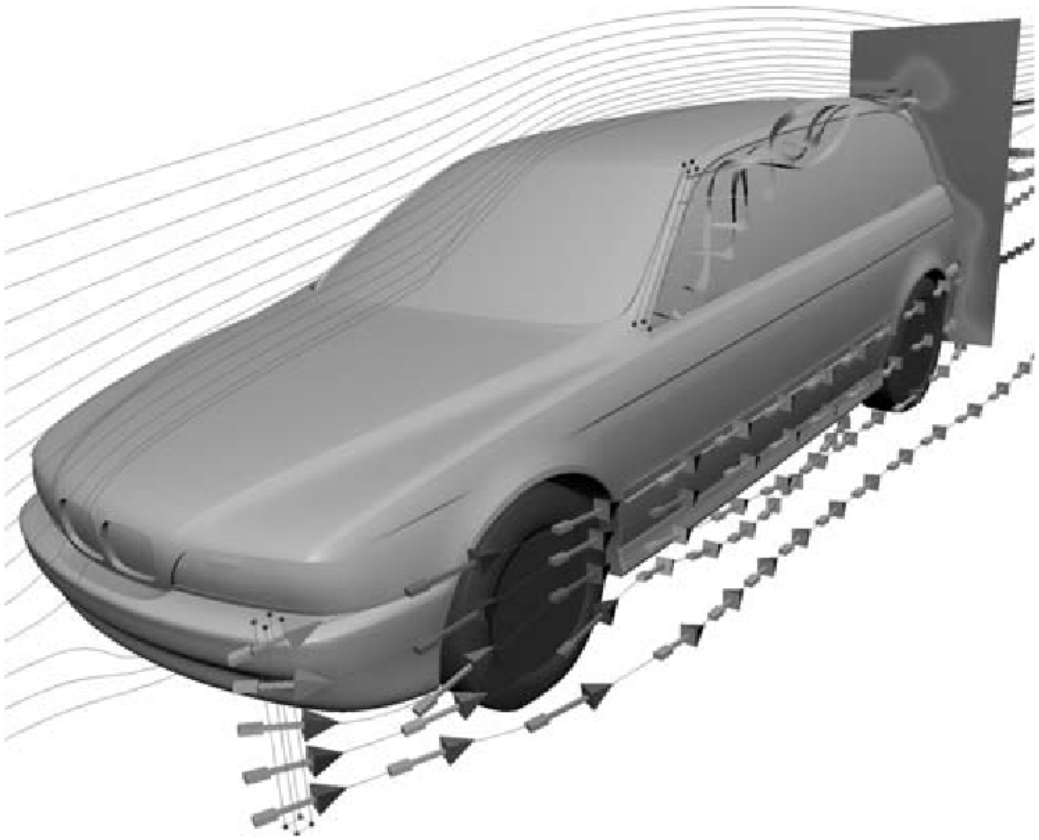


Figure 2.5: Combination of different direct visualization techniques. Glyphs indicate the flow at the car's side, while the path lines are used for the flow above it. Contour lines for the velocity's magnitude are drawn on a clipping plane at the back of the car. The techniques must be used carefully in order to avoid occlusion of the individual visualization entities. Taken from Hansen el al. [64].

with a feature-based visualization. Yet the description is yet limited to the pre-defined structures or features. In the previous section features in flow fields were introduced along with possible extraction techniques for those points of interest. For regions of interest, e.g., vortexes, the extraction methods are more complex and specialized on the properties of the extracted structure. For example, vortexes emerge around edges of an obstacle in the vector field, e.g., wingtips of an airplane. Rotations are caused by the obstacle and the fluid spins around the rotation axis. This curl can be compared to a tunnel in which the fluid is flowing with little interaction with the outside of the vortex. Jiang [78] extracts these vortexes, which are of particular interest for aerospace engineering. He builds a more general framework in which they are used to describe the vector field more precisely.

Partition-based visualization

This method segments the vector field into several distinct parts. Contour lines can be seen as the simplest partitioning technique. Using one scalar criterion, the vector field is separated into two parts. Also the already mentioned concluding clustering step for glyphs published by Heckel et al. [67] builds a segmentation of the vector field into parts of similar flow properties.

Another way to segment vector field is proposed by Chen et al. [30]. They extract periodic orbits of the vector field. Periodic orbits work analogously to contour lines, but instead of a scalar value, the orbits are computed based on the vectorial component. An orbit is a closed path line, i.e., if starting at any point of the orbit and integrating over time this starting point will be reached again after finite numbers of time steps. Therefore, the orbit separates between the particles inside the closed curve, which spin around the rotation center, and the outside particles, which move freely in the vector field.

More general than periodic orbits is the approach proposed by Theisel et al. [[132], [131]] and Weinkauf et al. [143]. First, the critical points defined by Helman et al. [68] are extracted. Second, path lines between these points are computed. The separation lines segment the vector field into regions in which particles of certain parts can move to accessible and inaccessible ones. This segmentation is, like the previous one, very descriptive and conveys the physical properties of the vector field.

The techniques mentioned so far extract the main features of the vector field and convey them to the user, who gains knowledge by putting the extracted features into context of the data set. Thus, the most creative pattern learning machine is utilized - the human brain. Conveying the data to the user is the bottleneck in the learning process, hence it is limited to the human visual system. This system is restricted to a couple of dimensions, few colors and can be heavily biased. Therefore, Kirby et al. [83] noticed the need for more precise methods that are able to precisely compute similarities and differences of the extracted structures by using statistical methods, which are common in most domains except visualization. The same is also stated by Fayyad et al. [51], who suggested using machine learning algorithms for analyzing scientific data sets.

2.3.2 Automated Data Analysis

The Definition 2.6 of features describes their appearance less specifically than it is done for traditional visual feature extraction. The more general definition is a pre-

requisite to transfer this classic visualization problem to other domains. Grouping cells into sets of similar properties and behavior is traditionally solved by clustering. Cluster analysis is a core task in data mining [63], which provides many statistical and automated approaches to analyze complex, high-dimensional data sets. Its problem statement is either defined bottom-up or top-down [48].

Definition 2.10. *Clustering - bottom-up[44]*
A given set of data points is grouped by some natural criterion of similarity into subsets.

Definition 2.11. *Clustering - top-down[11]*
A given heterogeneous population is segmented into a number of more homogeneous subgroups by clustering.

Both definitions are equivalent, only the view is changed. In the bottom-up definition the individual data points are grouped together, while in the top-down approach large sets of data points are split to obtain subgroups in which the members are more similar to each other than in the former set.

Clustering is more abstract than visual feature extraction, so is the definition of features therein. Instead of searching for points with significant properties as it is done in visualization, the data points are clustered into subsets of similar properties, s.t., every data point belongs to exactly one subset and a complete segmentation is obtained, similar to the application of contour lines. The change from significant to similar properties is the key point and the biggest advantage of automated techniques over visualization. Because of their construction, visualization techniques focus on the visual representation and are limited to two dimensions. Consequently the feature extraction concentrates on a subset of the data vector which contains the significant property. Clustering techniques, in contrast, are constructed to process high-dimensional data sets. They also have some disadvantage, too, e.g., overall runtime and restrictions on the extracted segments, which will be the focus of the next paragraphs. The feature definition must also be more precise, because a computer algorithm is less flexible than the human visual system.

Centroid-based clustering

The oldest class of clustering algorithms is also the simplest one. The common representatives of this class are k-means [92] and k-medoids [80]. We will briefly introduce both for the evaluation.

The parameter k specifies the number of assumed clusters in the data set. This a major disadvantage of these approaches, because the number of clusters must be be known initially. Therefore, to give a good estimation, the user needs knowledge of the data and its properties. Given the initial k cluster centers the other data points are assembled around these groups. The clustering $\mathcal{C}l$ is evaluated by the sum of squares of the within-cluster distances between every cell $\mathcal{C}_j$ of the cluster $\mathcal{C}l_i$ and its assumed center μ_i. The objective function is defined by

$$\arg\min_{\mathcal{C}l} \sum_{i=1}^{k} \sum_{\mathcal{C}_j \in \mathcal{C}l_i} \|\mathcal{C}_j - \mu_i\|_p^2 \tag{2.6}$$

The choice of the centers μ_i differs for k-means and k-medoids. For k-means the center can be selected arbitrarily, but for k-medoids the centers must be an element of the data set, i.e., $\mu_i \in \mathbb{V}$. The choice p of $\|\cdot\|_p$ is arbitrary as well, but strongly influences the clustering quality. Although other values are vaild for p, we only examined the most common values, the Manhattan metric $p = 1$, the Euclidean metric $p = 2$ and the Maximum metric $p = \infty$.

Both algorithms, k-medoids and k-medoids, suffer from the so-called curse of dimensionality [18]. This effect occurs when high-dimensional data is measured by one of the suggested metrics and influences it significantly, i.e., $\|\vec{c}_j - \vec{c}_i\|_p \to 0$ for $m \to \infty$, $\vec{c}_j, \vec{c}_i \in \mathbb{R}^m$. Additionally, by construction of the algorithms, they are only capable of finding concave-shaped clusters and are very sensitive to outliers.

Density-based clustering

Density-based clustering addresses the two disadvantages of the previous class. The representatives DBSCAN [47], Optics [13] and HiCo [9] are less sensitive to noise and are able to find clusters of arbitrary shape. Also the number of clusters is not initially required.

When DBSCAN was presented, it defined this class. It introduced the concept of reachability, which enables finding clusters of arbitrary shape and separating outliers. Every cluster must be formed by at least *minpts* of data points within an ϵ neighborhood. All data points in ϵ distance from a cluster members are member of this cluster, too, i.e., their data points are reachable. Therefore, *minpts* and ϵ must be initially defined. Optics and HiCo are variations of DBSCAN, which were developed to cluster large data sets more efficiently than DBSCAN, but all are constructed around the concept of reachability.

Distribution model

An underlying model for the distribution of data points is assumed, which is the case for CFD simulations. Therefore, the data points are not arbitrarily distributed in spatial space of vector field $\mathbb{V}$, but belong to distinct distributions. For natural processes one can assume a Gaussian distribution, i.e., the input data is interpreted as it is formed by a mixture of Gaussian distributions. Starting with this assumption a random mixture of Gaussians is initialized for the input and the approximation is refined by applying an expectation-maximization heuristic [41]. The actual maximization step is equivalent to the optimization step of k-means (2.6).

Hierarchical clustering

It can again be categorized into two sub-classes: the agglomerative and the divisive algorithms [142]. They only differ in how the overall data set is treated. The agglomerative algorithms start at the atomic level and merge the elements until there is only one cluster left containing all data points. The divisive ones start with one large cluster, which is split until every data point is a cluster of its own. The result is a dendrogram, a tree-like representation consisting of all merges or splits.

Similar to the density-based clustering approaches the hierarchical ones are based on connectivity of cells as well. To decide if two clusters $\mathcal{C}l_a, \mathcal{C}l_b$ are merged to one for the next agglomerative step a fusion function is applied. For instance:

$$\min \{ \|\vec{x}_1, \vec{x}_2\|_p : \vec{x}_1 \in \mathcal{C}l_a,\ \vec{x}_2 \in \mathcal{C}l_b \} \tag{2.7}$$

$$\max \{ \|\vec{x}_1, \vec{x}_2\|_p : \vec{x}_1 \in \mathcal{C}l_a,\ \vec{x}_2 \in \mathcal{C}l_b \} \tag{2.8}$$

are two possible fusion functions. Note that the choice of the metric also depends on p like for the centroid-based clustering. Eqn. 2.7 is the so-called single-link criterion and Eqn. 2.8 the complete-link criterion on which the Slink [123] and the Clink [40] algorithms are based.

The major advantage of these algorithms is that the number of clusters does not need to be known a priori as for k-means or k-medoids. Instead, it can be chosen a posteriori. The number of clusters is equal to the cutting level of the dendrogram. To work efficiently on large data sets, CuRe [59] performs a clustering of samples from the input vector field and represents the sample hierarchically. The key point is the sampling of the vector field. Naively, a random sample of data points $\vec{x}_v \in \mathbb{V}$ can

be picked. Since the major focus of interest lies on the flow singularities and their small neighborhood, the sample must be large enough to pick at least one member of it. Other than that, the vector field can be split by the Helmholtz decomposition into its different components. In the next step the singularities in every component must be found and combined with a set of random points to obtain the overall sample representing the complete vector field. The convenience of picking the cutting level a posteriori is traded in for a generally longer runtime. Birch [153] was developed to overcome this issue and performs a hierarchical clustering on large data set efficiently.

Subspace clustering

The techniques are specialized on high-dimensional data sets. They are constructed to be more stable against the curse of dimensionality for high-dimensional data. Kriegel et al. [85] and Parson et al. [105] summarized recent approaches. Subspace clustering maps the data onto a lower-dimensional space which is spanned by the most significant axis to represent the input data. Agrawal et al. [10] apply this concept for CLIQUE.

Grid-based clustering

This group of clustering algorithms is developed to process very large data sets. Typically the domain in which the data set is embedded is segmented into grid cells or blocks. These blocks are not necessarily equally sized, their volume depends on the number of data points they are enclosing. The actual clustering process uses this segmentation to reduce the runtime, because not all data points need to be compared with every other one.

Schikuta [120] developed GridClus. It is the simplest grid clustering technique. It applies the mentioned pruning method to speed up the nearest neighbor search by comparing the data points to the blocks instead of to all data points inside the block. Consequently, it can be interpreted as the k-means algorithm for very large data sets.

GridClus can be modifying in two different ways, either by improving the partitioning techniques or by changing the concluded clustering algorithm. OptiGrid [71] builds the blocks according to the distribution of points. Because of this optimized partitioning the runtime is reduced. GNDBSCAN [74] adapts DBSCAN, such that it can be applied for the data set's grid. The main idea is the same - to reduce the number of comparisons by initially grouping the data points into blocks. Because of the combination of DBSCAN with a partitioning of the data set's domain, the advantages of both techniques are joined.

2.4 Feature-Based Flow Field Representation

The definition of features has been changed and we need to derive a new representation, because the classic ones are not compatible with this modified definition. So far, the visualization focused on significant points or structures, as a result the complete data set was reduced to just some points or regions. The vector field is segmented into features covering it completely. Building a visualization which contains all extracted features would be overloaded and of no use to the viewer. Furthermore, the established visualization techniques are qualitative, i.e., they highlight one effect, but do not describe it precisely. The proposed feature extraction using clustering provides a quantitative description of every feature and its properties. Therefore, the proposed vector field representation is based on the feature definition and must be capable of conveying the extracted quantitative information efficiently. To overcome the conceptual weaknesses and disadvantages of visualization techniques, it is constructed more generally and less dependent on visual entities.

Graphs represent complex structures or connections in many different contexts, e.g., in life science to describe metabolic pathways [150] or for object recognition in computer vision [148]. In Figure 2.6 basic examples for different types of graphs are provided, but all graphs are formed by at least one set of nodes and one set of edges, which connect nodes with each other. The simplest graphs are the undirected ones, in which unlabeled nodes are connected by unweighted edges. The dependencies between the nodes are represented by the directions which are added to the edges. Furthermore, properties of nodes and edges can be included in the representation, too. This makes the graphs measurable and consequently comparable to each other. Metrics for graphs take the labels but also the overall graph's structure into account. Based on these metrics, similarities in two graphs can be found and a matching between both graphs is computed. Finding similarities between graphs is not trivial though, even if they are labeled by their properties.

The graph representation is also suitable to convey geometric data sets. For instance, computer-aided design aims to construct and present complex structures. In order to gain an overall, simple representation of a complex object, it is segmented into components which are used to build a graph [[53], [137]]. For example all parts of a car can be portrayed by nodes. Assembling two parts spans an edge between their nodes and so the complete car can be represented by one graph. Grouping nodes into subsets simplifies the representation, e.g., the tyre, rim, lug nuts, disk brakes and knuckle can be grouped together into the superset wheel. The grouping also

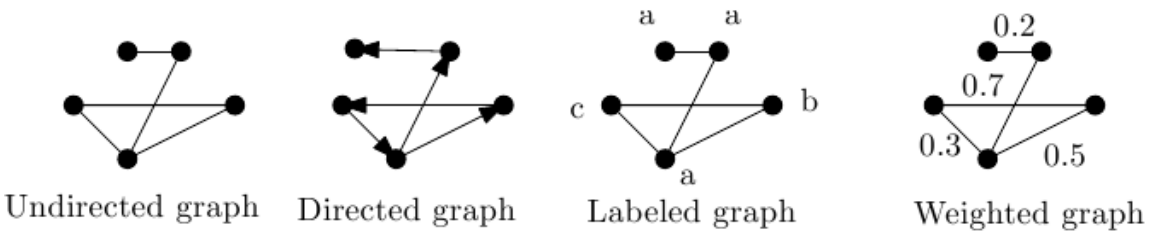


Figure 2.6: Different types of graphs ordered by their complexity. The very basic undirected graph consists of nodes and connection edges, while the groups to the right contain labels or weights, respectively. Taken from Diestel [43].

allows a scalable display of the object, to which the visualization community refers as level-of-detail [27].

Graphs are also used for knowledge extraction. In image processing, complex structures, e.g., faces, are modeled by graphs. The graphs are extracted from the input images [[108], [93]]. This concept is also found in visualization [15]. Helman et el. [68] already proposed the extraction of a topological graph built on their features. Later Weinkauf et al. [[144], [143]] along with Theisel et al. [131] picked up the concept and developed the topological skeleton. The topological skeleton was developed by the visualization community and therefore is built on their specific features, e.g., vortexes, saddle, curls, which are connected by path lines, as seen in Figure 2.7. It summarizes an already finished visualization, which is, due to the special features, not intuitive for a viewer unfamiliar with the concept. This, along with the solely visually described features, makes it hard to compare topological skeletons with each other.

The topological skeleton is based on complex visual features which are not intuitive for the user to understand. In contrast to this, the feature's Definition 2.6 is more general. It defines features as segments consisting of individual cells. The definition focuses on the individual features and not the interaction between them. Therefore, the dynamic flow of the vector field is hardly conveyed at all. Setting the features into context to each other does not require a new definition, because the necessary information is already implicitly included in the data vectors $\gamma(\cdot)$. Based on the extracted features along with their relation to each other, a geometric graph can be built to represent the vector field.

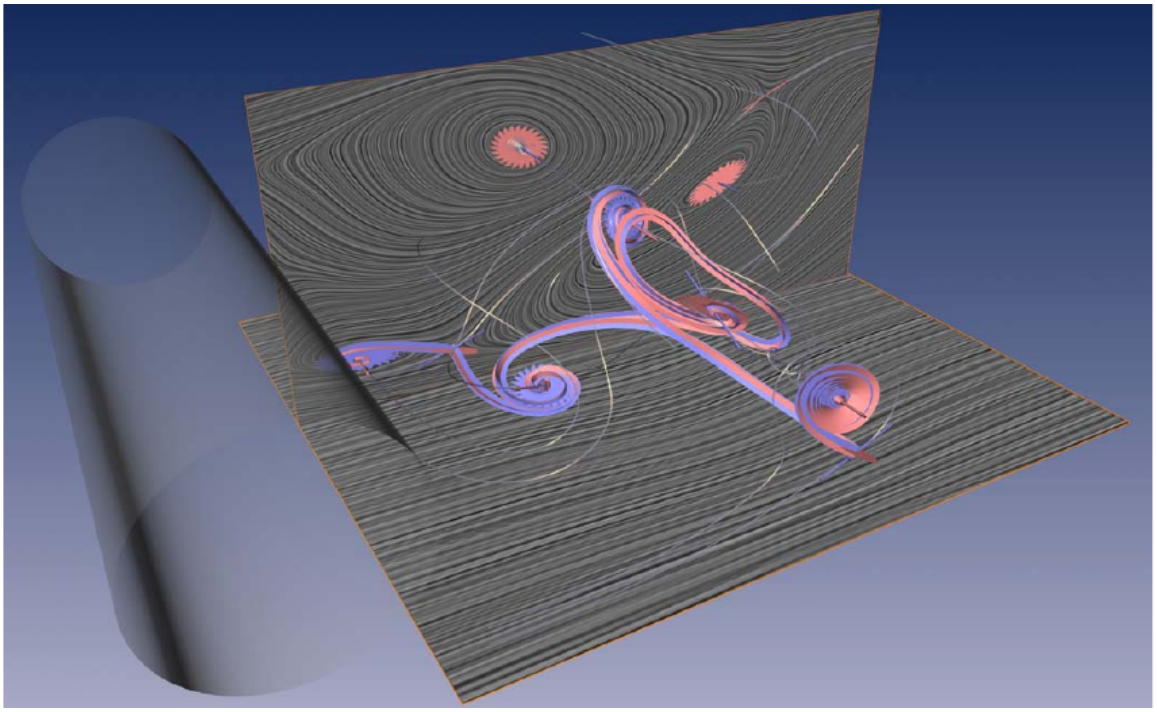


Figure 2.7: Flow behind a circular cylinder. 13 critical points and 9 saddle connectors have been detected and visualized. Additional LIC planes have been placed to show the correspondence between skeleton and flow. Taken from Theisel et al. [131].

Definition 2.12. *Geometric Graph*
The labeled directed geometric graph $G = (V, E, L) \in \mathcal{G}$ *consists of a finite set of nodes* V*, a finite set of edges* E *connecting the nodes pairwise and a set of labels* L*.* L *contains labels for every node* $n \in V$ *and every edge* $e \in E$ *including the node's spatial coordinates.*

This definition is basically the standard graph definition, but the set of labels L is added to describe the properties of the nodes and the behavior of the edges. In contrast to the topological graph or the topological skeleton, which are both applied by the visualization community, the introduced geometric graph is not based on any specialized visual entities. We will refer to geometric graphs representing vector fields as geometric flow graphs or simply flow graphs. Because of its generality, the geometric graph is not only easy to understand for the viewer, but it is also possible to apply standard techniques for graph similarity and graph comparison to obtain a quantitative analysis for the vector field and its evolution.

2.5 Feature-Based Flow Field Comparison

The constructed geometric flow graph provides the information necessary to compare the represented vector field with graphs of other vector fields. By extracting the features first and describing the vector field using those features, a more detailed description is obtained. Furthermore, graph comparison approaches can be applied on this scenario and enable us to apply powerful concepts from graph theory for vector field analysis.

Typically a simulation consists of several time steps describing the properties and the behavior of the flow for a defined time interval, in which a vector field is computed for every time step. Regarding the complete time interval the vector field evolves continuously. Therefore, between two time steps close to each other, the obtained vector fields is slowly changing and the overall appearance is mainly contained. Comparing vector fields of different simulations requires different techniques. Such two vector fields can have a totally different appearance and thus a transition between both is more challenging to find. Therefore, we distinguish between the two scenarios, the *evolution* and the *comparison* of vector fields.

Traditionally two vector fields are compared based on their visualizations. Therefore, one pipeline must be built that is applicable to the different vector fields. This includes not only the applied visualization technique, but also the major parameters. Then the resulting visualizations must be compared either manually by judging them visually or by extracting parameters for the entities used in the visualization. The first strategy leads to a qualitative comparison only, because it is not based on any numeric values or properties of the vector field. Additionally, it only relies on the visual system of the user, which can easily be deceived. In contrast, the second strategy extracts numeric values and provides a quantitative analysis. The extracted values describe the visual entities and not the vector field itself. Post et al. summarize the disadvantages of the recent approaches: *"The techniques are generally very specific for a certain type of problem (such as vortex detection), the relation with the original raw data is indirect, and the reduction is achieved at the cost of loss of other information, which is considered not relevant for the purpose" [109].*

In general, feature-based techniques are the simplest methods of describing the evolution of vector fields, because only the evolution of the features must be described. However, this is still very difficult to achieve. Even for the simplest features, the critical points, a proper description includes several properties. To describe the transition between two critical points, the two components, the divergence and the

rotation, must be taken into account. Lavin et al. [90] and Batra et al. [17] performed this multi-dimensional comparison applying the earth-mover-distance, which describes the change in appearance as an overall sum of different components. However, they did not take the displacement of the critical points into account. Theisel et al. [130] extracts the toplogical skeleton, which is based on the critical points, and preserves the topology and traces it over time steps via path lines.

For geometric visualizations Verma et al. [139] propose comparing the evolution of path lines between the different time steps. This is done by comparing the geometric properties of the path lines as geometric entities, including their orientation and speed. The mentioned visualization techniques are lacking a suitable representation for the vector field data. Therefore, the graphs of the vector fields can only be judged visually and additionally a meaningful measure cannot be defined to match the graphs.

Graph matching is very important in many different domains, e.g., computer vision, chemistry and life science. In all those domains, graphs are used to represent complex data in a simplified and more abstract manner. For instance, in computer vision, pictures of the human face are compared indirectly by extracting features, e.g., eyes, nose, mouth, and the constructed graph can be compared to a database of known faces [148]. For geometric graphs the structure but also the labels of the graph are compared to determine the similarity of different graphs. Recent work distinguishes between the terms of *graph comparison* and *graph matching* [14]

Definition 2.13. *Graph Comparison*
Given two graphs $G_1, G_2 \in \mathcal{G}$ the function $gc : \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}$ defines the distance between the graphs G_1 and G_2, $gc := dist(G_1, G_2)$.

Definition 2.14. *Graph Matching*
Given two graphs $G_1, G_2 \in \mathcal{G}$ the subgraphs $SG_1 \subset G_1$ and $SG_2 \subset G_2$ are mapped by the function $gc(SG_1, SG_2) = 0$.

The graph comparison can be seen as a generalization of the graph matching. For two matching graphs the properties must be identical, i.e., the distance of both graphs G_1, G_2 must be $gc(SG_1, SG_2) = 0$. In contrast to this, the graph comparison gives a quantitative answer as to how similar both graphs are. In the following we discuss the two groups of matching algorithms, which answer the previously defined problems. We follow the work of Armiti [14], which nicely summarizes established techniques for both problems, they are additionally portrayed in Figure 2.8.

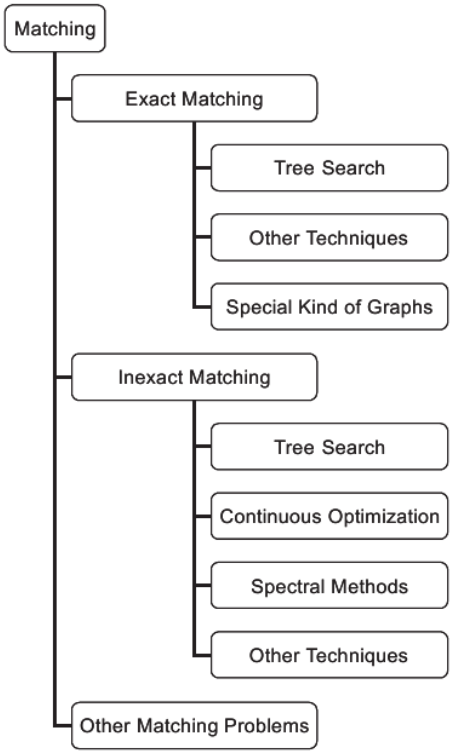


Figure 2.8: Overview of graph matching techniques categorized by their matching strategy. Taken from Conte et al. [36].

2.5.1 Exact matching

Exact matching answers the previous question of exact matching graphs or subgraphs, respectively. If two graphs G_1, G_2 can be matched, a graph isomorphism can be found which maps one graph onto the other one. Since this transformation is an isomorphism, the mapping is particular bijective. If the two graphs cannot be completely matched, a subgraph isomorphism can be defined sometimes.

The assumption of finding an isomorphism between two geometric flow graphs is too restrictive. If two geometric flow graphs are isomorph, they are exactly the same, i.e., the represented vector field of both graphs is the same. This scenario occurs only if the simulation is in a steady state. Such steady states typically describe the end state of the simulation, because the system will remain in the steady state. Therefore, graph isomorphism can only be applied to identify a steady state, but not to analyze the simulation output before a steady state is reached.

Definition 2.15. *Graph Isomorphism [14]*
Two graphs G_1 and G_2 are isomorphic, if there is a bijective function between the graphs $i(G_1) = G_2$.

The function $i(\cdot)$ finds correspondences between the graphs $G_1 = (V_1, E_1, L_1)$, $G_2 = (V_2, E_2, L_2)$, such that every node $v_{1,i} \in V_1$ is mapped onto exactly one $v_{2,j} \in V_2$. Both nodes $v_{1,i}, v_{2,j}$ have identical labels and edges. Indeed, the assumption can be weakened by finding matching subgraphs instead of matching graphs.

Definition 2.16. *Subgraph Isomorphism [14]*
Two graphs G_1 and G_2 are subgraph isomorphic, if the subgraph $SG_1 \subset G_1$ is isomorphic to $SG_2 \subset G_2$.

Therefore, the complete graphs are not matched, but the subgraph of one with the other complete graph. This can happen if one of the graphs has fewer nodes than the other one. The graph with fewer nodes, is regarded as subgraph of graph G_2 without loss of generality G_1. Read et al. [111] proved that the subgraph isomorphism is actually NP-complete, because G_1 needs to be compared to all subgraphs $SG_{2,i}$ which have as many nodes as G_1. As already mentioned, the graph isomorphism and the subgraph isomorphism are very similar. Therefore, the techniques to compute one of them can be adapted to compute the other one as well. One possible algorithm performs a tree search with backtracking [36], in which the matching is iteratively built by adding pairs of nodes to the initial empty matching. One node of one graph is chosen to be added and the algorithm checks possible matchings in the other graph. If two nodes of the two graphs match, they are added to the matching. If not, a backtracking is applied and the last pair is removed from the matching in order not to run into this state again.

Generally tree search with backtracking scales poorly with respect to graph size n. To reduce the runtime to $O(n^3)$, Ullmann [134] adds a look-ahead strategy to minimize the search space. Cordella et al. [37] proposed VF2 , which further improves the runtime to $O(n^2)$. Instead of randomly adding nodes like the algorithm published by Ullmann, the nodes are added to the matching which is connected to a node already in the matching. Both techniques can be also applied to subgraph matching by relaxing the size criteria, because for the subgraph isomorphism, the graphs are of different sizes, which is different to graph isomorphism.

Another generalization of graph isomorphism is the maximum common subgraph, which is even less restrictive than the subgraph isomorphism. The definition of subgraph isomorphism assumes that the smaller graph is a subgraph of the other one, while the maximum common subgraphs search for the largest graph which is subgraph of both other graphs.

Definition 2.17. *Maximum Common Subgraph[25]*
For two given graphs G_1, G_2 the maximum common subgraph msc is the maximum subgraph of G_1 which is isomorphic to a subgraph of G_2.

The complexity of this problem is NP-complete [55]. It requires finding an subgraph isomorphism between the *mcs* and G_1, but also between *mcs* and G_2. Furthermore, the solution is not unique, because different subgraphs of the same size may suit the definition requirements. McGregor [98] proposes a direct solution to the maximum common subgraph problem. However, the problem is commonly transferred to the problem of finding a maximum clique of the product graph [[16], [36]] for which efficient algorithms exist [23].

2.5.2 Inexact matching

While exact matching techniques search for exact identical structures inside the graphs, which are supposed to be matched, inexact matching techniques are robust to changes inside the graph and identify similar structures. Therefore different numbers of nodes and edges are not as critical as they are for exact graph matching [118]. In fact, instead of correspondences between the graphs an optimization problem must be solved, which describes the necessary changes to transform one graph into the other one [57].

Definition 2.18. *Inexact Graph Matching [14]*
For the two graphs G_1, G_2 the mapping between both is given by the matrix $M^{|G_1|\times|G_2|}$. The matrix's entries $m_{i,j} = 1$ if the i-the node of $n_i \in G_1$ is mapped onto the j-th node $n_j \in G_2$ and $m_{i,j} = 0$ otherwise. The optimal solution is given by:

$$M^* = arg\min_M \sum_i \sum_j dist_l(n_i, n_j)\ m_{i,j} + \sum_i \sum_j \sum_k \sum_l dist_s(e_{ij}, e_{kl})\ m_{ik}\ m_{jl}$$
$$\text{with the constraints} \tag{2.9}$$
$$\forall k \in \{1, ..., |G_1|\}, \sum_{i=1}^{|G_2|} m_{ik} \leq 1$$
$$\forall i \in \{1, ..., |G_2|\}, \sum_{k=1}^{|G_1|} m_{ik} \leq 1$$

where $dist_l(\cdot)$ measures the distance in the nodes labels and $dist_s(\cdot)$ the structural difference.

The objective function consists of two terms to take the structural differences but also the different labels of the graphs into consideration. For an exact matching the M^* vanishes and for the inexact matching the objective function is minimized. Note

that the constraints are defined such that the graphs can have different number of nodes. Therefore, not all nodes of the graphs must be matched, but if the node of one graph is matched then exactly with one node of the other graph. Solving this optimization problem is NP-hard [26].

Spectral Graph Matching

This method solely relies on the structure of the graphs, i.e., the nodes and the edges but not their labels, and was developed for unlabeled graphs. The spectral graph theory [34] provides different strategies which can be utilized to match two graphs.

Definition 2.19. *Spectrum [14]*
The spectrum of the graph is the Eigenvalues for its adjacency or (normalized) Laplacian matrix.

In this definition the adjacency matrix for the graph's connectivity is used, but other connectivity matrices such as the Laplacian Matrix and the Normalized Laplacian Matrix are possible, too. The Eigenvalues are the root of the characteristic polynomial, and so they are not unique. Therefore, two graphs can have the same spectrum, but are not isomorphic. In this case they are called cospectral. Umeyama [135] applied the spectrum in his graph matching approach. The Eigenvector of the adjacency matrix spans the Eigenspace, in which the nodes are embedded. Then he used the Hungarian algorithm to approximate the graph matching.

Continuous Optimization Approaches

Continuous optimization approaches approximate the solution by relaxing the discrete graph matching problem to another continuous non-linear one. For the latter problem many algorithms exist which find the solution for the optimization problem iteratively. After the optimal solution is found, the result must be mapped back to the discrete interval. The non-continuous optimization algorithm finds solutions in the range $[0; 1]$, but for the discrete optimization problem it must be mapped onto $\{0; 1\}$. The actual continuous non-linear optimization problem can be solved by standard techniques. Wilson et al. [147] applied the gradient ascent to minimize the objective function. Luo et al. [96] propose a maximum likelihood estimation.

Graph Edit Distance

This distance was proposed by Sanfeliu et al. [118]. Their approach not only takes the structural information into consideration like the previous approaches, but includes

the labeling information as well. To measure all changes necessary to map one graph onto another, a set of edit operations is introduced. This set also includes operations for adding and removing nodes or edges, respectively. Every operation has a cost or penalty to measure the differences between the graphs quantitatively. Those penalties are initially defined by the user [113].

Definition 2.20. *Graph Operations [14]*
For the graph $G(V, E, L) \in \mathcal{G}$ *the set of graph operations* $\mathcal{T}$ *transforming it to* $G'(V', E', L') \in \mathcal{G}$ *consists of*

$$\begin{aligned}
&\textit{Node operations:} && \\
&\textit{Node insertion:} && (\epsilon \to v_i'),\ v_i' \in V' \\
&\textit{Node deletion:} && (v_i' \to \epsilon),\ v_i' \in V' \\
&\textit{Node substitution:} && (v_i' \to v_i'),\ v_i' \in V',\ v_i' \in V' \\
&\textit{Edge operations:} && \\
&\textit{Edge insertion:} && (\epsilon \to e_i'),\ e_i' \in E' \\
&\textit{Edge deletion:} && (e_i' \to \epsilon),\ e_i' \in E' \\
&\textit{Edge substitution:} && (e_i' \to e_i'),\ e_i' \in E',\ e_i' \in E'
\end{aligned} \tag{2.10}$$

where ϵ *defines a node or edge, which is newly created. The costs of each operation is given by the function* $pen(\cdot)$*.*

The set of graph operations consists of six operations for the graphs nodes and edges. Applying an operation on a node triggers operations on the attached edges, e.g., the deletion of a nodes triggers the deletion of the attached edges. The change of the labels is included in the substitution operations.

Definition 2.21. *Edit Path [14]*
The sequence of graph operations to transform graph $G_1(V_1, E_1, L_1) \in \mathcal{G}$ *into* $G_2(V_2, E_2, L_2)\mathcal{G}$*, is the edit path*

$$P(G_1, G_2) := \{t_1, ..., t_n\},\ t_i \in \mathcal{T}, n \in \mathbb{N} \tag{2.11}$$

Because there is no further limitation involved in the edit path's definition, the path to transform G_1 into G_2 is not unique. By adding an additional constraint at the costs of the edit path this ambiguity is solved and one distinct path $P_i \in \mathcal{P}$ is chosen.

Definition 2.22. *Graph Edit Distance [14]*
For the two graphs $G_1(V_1, E_1, L_1), G_2(V_2, E_2, L_2)\mathcal{G}$ *the graph edit distance is the edit path* $P(G_1, G_2) \in \mathcal{P}$ *with the minimal sum of all graph operations costs*

$$dist_g := \min_{P_i \in \mathcal{P}} pen(P_i(G_1, G_2)) = \min_{P_i \in \mathcal{P}} \sum_{t \in P_i} pen(t) \tag{2.12}$$

Sanfeliu et al. [118] proposed choosing the costs $pen(\cdot)$ such that the distance $dist_g(\cdot)$ is a metric function. This is achieved if $dist_g(\cdot)$ is positive definite, symmetric and the triangular inequality is fulfilled by it. Therefore, the cost for deletion and insertion of nodes and edges with identical labels must be the same and the substitution operations must be based on metric functions, too.

2.6 Summary and Discussion

In this chapter we summarize related concepts of different domains. For a better overview the individual steps of the our workflow were split and the related work was summarized for each part.

1. **Feature.** Their appearances and the corresponding properties vary a lot, especially for different visualization techniques. Our feature definition (Definition 2.6) is kept as general as possible in order to suit most of the scenarios, but it is still compatible to established definitions. The definition is built on cells as the smallest unit and defines regions of interest as features.

2. **Feature Extraction.** Traditionally the feature extraction, which is seen as the first layer of knowledge extraction, is performed visually. Therefore, the user is the key to building and understanding the visualization pipeline. This causes advantages and disadvantages of a user guided and interactive techniques. We judged the user's impact and influence on the results are the major stumbling blocks of the established techniques. Thus, we apply automated techniques like they are found in data mining to extract the features.

3. **Feature-Based Representation.** Automated techniques lack the easiness and intuitiveness of user-guided techniques. We therefore introduced the representation of flow fields as geometric flow graphs. It provides an intuitive representation for the automatically extracted features and the overall vector field. More importantly, it links the two domains vector field visualization and

graph theory. This allows us to apply powerful graph similarity techniques for vector field analysis and comparison.

4. **Feature-Based Analysis and Comparison.** Graph representing techniques are common in many domains and analytics on graphs, too. The vector fields can be compared by their representing graphs. We distinguish here between the evolution and the comparison of vector fields. For the evolution the vector fields are only slowly changing in their appearance and properties over time, which makes an inexact graph matching approach based on graph edit distance more preferable. However, for the comparison of two different graphs, the structural properties are more important than the labeling information. Therefore, spectral matching techniques fit this scenario better than those for the graph evolution.

The following chapters will give a more detailed explanation of the mentioned components of the workflow and we start with the extraction of features.

Chapter 3

Feature Extraction

Feature extraction is the first processing step after data acquisition and therefore the first layer of knowledge extraction. It indicates the vector field's features and consequently simplifies the vector field in a way, that its major information is easier to understand and interpret. This simplification is achieved by some level of abstraction, e.g., the data points are grouped based on their similarity or visual structures. Traditionally, features of vector fields are extracted by searching for critical points. Helman et al. [68] and Scheuerman et al. [119] compute the partial derivatives, build the Jacobian Matrix and identify its Eigenvalues to categorize the features by the order of its derivatives. This can be seen as a bottom-up approach, since it extracts points of interest on the lowest level of the hierarchy and aggregates neighboring points around those to compose regions of interest.

The search for critical points is also the key inside the visualization pipeline to understand the data set. These points are most significant to describe the vector field's properties and so they are the origin point for analysis. The efforts are concentrated on them and their extraction to reduce the complexity of the complete data set from several thousand points to just a few features. This reduction enables the user to comprehend the data, because his visual system would be overcharged by the number of points of the complete vector field.

A reduction in complexity bears the risk of oversimplifying the data, though. If the user focuses on critical points only, other significant structures inside the vector field may be not realized because they are clipped away. Furthermore, structures, that are not related to critical points, are filtered and not considered any longer. In contrast to the user-guided visualization techniques, automated extraction techniques do not require such a radical reduction in complexity. Since their execution requires little or no user interaction, they are capable of processing large, high-dimensional data

autonomously. Therefore, the risk of missing or cutting away relevant information is reduced to a minimum.

The advantages of automated extraction techniques are utilized in many different applications, each of which requires different properties and specifications of the extraction method. For instance, finding key words in corpora is specialized on words and text [76], while finding objects in images requires techniques, which recognize complex geometric objects [93]. In our context, the vector fields consist of numeric data only, on which classic clustering techniques are applied. Because clustering algorithms group the data points into clusters of similar properties, the feature extraction process is changed to a top-down view. The complete input vector field is segmented into clusters of similar data points. Consequently, significant points are extracted correctly but so are regions of homogeneous flow, which do not necessarily contain a flow singularity and are ignored in classic visualization approaches. Finding elements which are exposing similar properties and grouping them into subsets is the objective of clustering techniques. By the new definition we not only transfer the feature extraction in vector fields from the visualization domain to the domain of data mining, but also from a qualitative analysis to a more quantitative one. The proposed feature extraction is based on numerical properties of the feature and their measurable similarity amongst each other.

However, automated techniques require a more general definition of features, which is not based on one flow property only, e.g., rotation or divergence. The feature vector summarizes all properties and the behavior of a cell, which is the smallest possible feature. Furthermore, the feature vector is constructed, such that it is applicable on features of several cells, too. The resulting feature vector consequently characterizes regions of interest instead of points of interest.

The feature extraction is based on our definition of features (Definition 2.6). Because this definition differs from established ones significantly, the following chapter does not only focus on the extraction but provides details on the feature's properties combined in the feature vector as well. This combination is a significant change towards a comprehensive analysis. Because visualization techniques are limited to two dimensions, they focus on distinct properties of the vector field but not on all available information. Therefore, the possibility of missing correlations and trends in the data is very high. This change in the feature properties affects the complete workflow, but the novel feature description must still be compatible to established ones. Thus, a feature vector is defined, which includes properties of traditional techniques but also newly derived ones. The feature vector must be structured to be a valid input for a

clustering algorithm. This also includes considerations about the dimensionality and the corresponding length of the feature vector, because some clustering algorithms are sensitive to high-dimensional data.

The first part of this chapter focuses on the feature's properties and how they are derived from the input data. The results will be our definition of the feature vector, which precisely describes features as regions of interest. In the second part, the complete workflow is tested against established visualization techniques. This part also includes the pre-processing of the input as well as the choice of the clustering algorithm. The evaluation is concluded by a comparison of results in terms of quality and runtime. The experimental evaluation is performed on two test cases, the generic backward step and the lid-driven cavity. The generic backward step is fully synthetic and ground-truth information is available to validate the findings and prove the presented concept. The lid-driven cavity is also a well-known test case and well described in theory.

3.1 Feature Vector

In the background section, different features descriptions were introduced. Each of those descriptions is specific to one specialized extraction technique. In contrast to those, the Definition 2.6 used here is more general. This generality has two major advantages, it is applicable to more scenarios and it can be used for automated methods.

Traditional feature-based visualization techniques are constructed to highlight and visually extract pre-defined features. Those features are usually critical points, which are characterized by their divergence and rotation. In order to be comparable to those techniques, the definition of the feature vector must include those properties, too. Instead of searching for specific points and building a visualization based on those points, our approach groups cells into groups of similar properties. The feature vector is defined for the most atomic level of data representation, an individual cell.

Definition 3.1. *Cell's Feature Vector*
For every cell $\mathcal{C}_i \in \mathcal{C}$ *of the vector field* $\mathbb{V}$ *the cell's feature vector* $\mathcal{FV}_{C_i} := \{\vec{c}, \vec{u}, div, rot, t, V, data\} \in \mathbb{R}^m,\ m \geq 10$ *is given by the cell center,* $\vec{c} \in \mathbb{R}^3$*, the flow vector at the cell center* $\vec{u} \in \mathbb{R}^3$*, the divergence* $div \in \mathbb{R}$ *and rotation* $rot \in \mathbb{R}$ *over the cell, the cell type* $t \in \mathbb{N}$*, the approximated volume of the cell* $V \in \mathbb{R}$ *and an additional data vector,* $data \in \mathbb{R}^{(d-3)}$.

Based on the cell's feature vector, the similarity of the cells among each other is estimated, which is necessary to group the cells into clusters or groups of similar properties and behaviour. Additionally, the definition precisely describes every cell and the features formed by them. This leads towards a quantitative vector field analysis. Visualization techniques focus on a qualitative analysis of the vector fields only and quantitative methods are still missing [83].

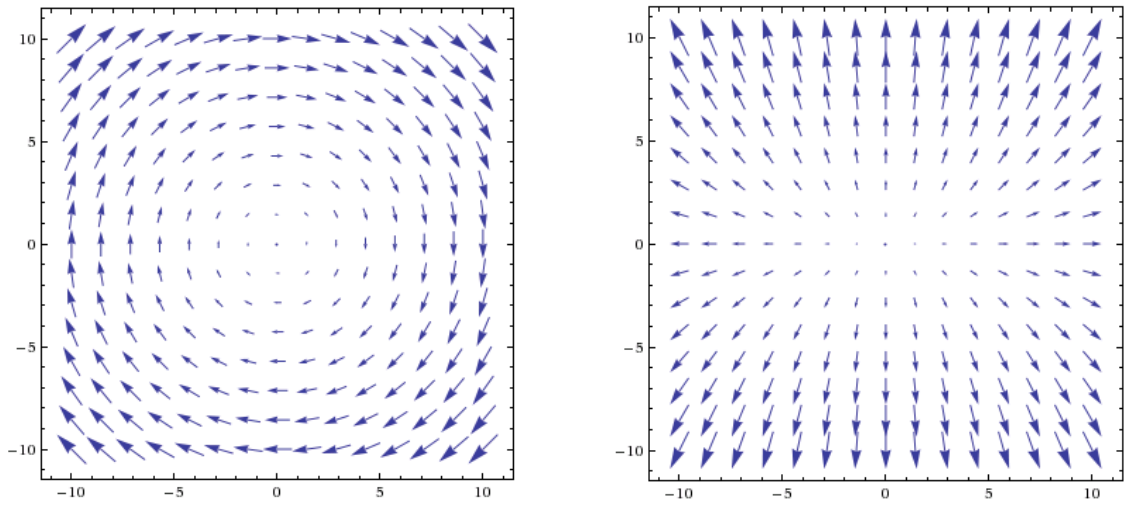
The individual components of the cell's feature vector are used in various different visual extraction techniques, but the combination of all properties to a single high-dimensional vector is new here. Visualization techniques are limited to two dimensions and therefore the properties must either be pre-selected carefully or the properties must be combined pairwise to one visualization each. Because the clustering algorithms which are the key of this automated feature extraction, are capable of processing high-dimensional feature vectors, the combination of all properties enables a comprehensive analysis. This is a major advantage over traditional visualization techniques and crucial to process the modern data sets with their increasing model complexity.

In the following, the single components of the cell's feature vector are derived from its corner points. The components are aggregated just by the properties of the corner points, except the rotation and the divergence. Both components are computed by an integration over the cell's volume. This is more costly, when it is only computed for the individual cell. In a later chapter, when the properties of the cell's are passed on to describe the extracted features, this computation is advantageous over the computation via the partial derivatives. The integrals for rotation and divergence sum up simply and do not require to be re-computed.

Center and velocity vector

Both components are mentioned briefly in Definition 2.4 and are discussed in more detail here. All corner points of the cell are regarded as equally important, so there is no special weighting for the cell's points. Therefore, by computing the arithmetic means of the corner points' coordinates, the cell's center of mass is estimated:

$$\vec{c_v} = \frac{1}{p} \sum_{i=1}^{p} \vec{x}_i, \ \{\vec{x}_1, \dots \vec{x}_p\} = \mathcal{C}_v \tag{3.1}$$



(a) Vector field with rotation

(b) Vector field with divergence

Figure 3.1: Vector fields.

The same argumentation applies to the velocity vector $\vec{u}_v$ at the cell's center $\vec{c}_v$:

$$\vec{u}_v := \gamma(\vec{c_v}) = \frac{1}{p}\sum_{i=1}^{p}\gamma(\vec{x}_i),\ \{\vec{x}_1, \dots \vec{x}_p\} = \mathcal{C}_v \tag{3.2}$$

Those two properties are necessarily included in the Definition 3.1, because they are required for a proper vector field definition (Definition 2.4). Additionally, they can be inherited by features of cells by computing the average overall cells forming it.

$$\vec{c}_{\mathcal{F}_v} = \frac{1}{q} \quad \sum_{i=1}^{q}\vec{c}_{\mathcal{C}_q},\ \{\mathcal{C}_1, \dots \mathcal{C}_q\} = \mathcal{F}_v \tag{3.3}$$

$$\vec{u}_{\mathcal{F}_v} := \gamma(\vec{c}_{\mathcal{F}_v}) = \frac{1}{q} \quad \sum_{i=1}^{q}\gamma(\vec{c}_{\mathcal{C}_q}),\ \{\mathcal{C}_1, \dots \mathcal{C}_q\} = \mathcal{F}_v \tag{3.4}$$

Note the similar formulations of Equation 3.1 and 3.3 and of Equation 3.2 and 3.4, respectively.

Rotation

The rotation and the following divergence describe the behavior of the flow over a particular cell. If a source or a sink impacts the cell, the divergence is non-zero. If the cell is influenced by a curl, the rotation for it is non-zero. In particular, the computed rotation vector indicates the direction and strength of the curl. Mathematically expressed, rotation is an operator applied on the vector field. The obtained result is a

new vector field giving the angle velocity for every point of the original vector field, to which we refer as the rotation of the vector field.

$$\begin{aligned} \text{rot } \gamma(\vec{x}_v) := \nabla \times \gamma(\vec{x}_v) &= \begin{pmatrix} \frac{\partial}{\partial x_1} \\ \frac{\partial}{\partial x_2} \\ \frac{\partial}{\partial x_3} \end{pmatrix} \times \begin{pmatrix} \gamma(x_1) \\ \gamma(x_2) \\ \gamma(x_3) \end{pmatrix} = \begin{pmatrix} \frac{\partial \gamma(x_3)}{\partial x_2} - \frac{\partial \gamma(x_2)}{\partial x_3} \\ \frac{\partial \gamma(x_1)}{\partial x_3} - \frac{\partial \gamma(x_3)}{\partial x_1} \\ \frac{\partial \gamma(x_2)}{\partial x_1} - \frac{\partial \gamma(x_1)}{\partial x_2} \end{pmatrix} \\ \vec{x}_v &= (x_1, x_2, x_3)^t \in \mathbb{V} \end{aligned} \tag{3.5}$$

The rotation of the vector field is computed by its partial derivatives and the application of Nabla operator ∇. If the points inside the vector field are influenced by a rotation, its strength and direction is given by the rotation value rot $\gamma(\vec{x}_v) \neq 0$. Consequently, the vector field given contains a center of rotation. For instance the vector field of Figure 3.1(a), it is defined by $\mathbb{V}_1 : \gamma_1(\vec{x}) \mapsto (-x_2,\ x_1,\ 0)^t$, i.e., it is spinning clockwise around the origin. Applying the rotation operator on this field results in a homogeneous vector field in which every vector is pointing along the negative z-axis, i.e., rot $\gamma_1(\vec{x}) = (0, 0, 3)^t$.

$$\begin{aligned} \frac{\partial \gamma(\vec{x}_v)}{\partial x_i} &:= \lim_{h \to 0} \frac{\gamma(x_i + h) - \gamma(x_i - h)}{2h} \\ \vec{x}_v = (x_1, x_2, x_3)^t \in \mathbb{V} \quad &, \quad i = 1, \ldots, 3,\ h \in \mathbb{R} \end{aligned} \tag{3.6}$$

For an explicitly given vector field, the Equation 3.6 can easily be solved and its rotation rot $\gamma(\vec{x}_v)$ is obtained. In contrast, for an implicitly defined discrete vector field, the computation of the partial derivatives is more complicated. The points which are in h-distance to $\vec{x}_v \in \mathbb{V}$ are not necessarily element of the discrete vector field as well, but they are needed to compute the central differential quotient in Equation 3.6. Because of the definition of cells (Definition 2.3), a continuous representation of the discrete vector field is constructed, which is required to apply the Rotation Theorem.

Theorem 3.1. *Rotation Theorem [12]*
For the vector field $\mathbb{V}$, which is compact subset of $\mathbb{R}^n$ the rotation can be represented by:

$$\int_C \text{rot } \gamma(\vec{x}_v)\ \mathrm{d}C = \int_{\partial C} \gamma(\vec{x}_v)\ \mathrm{d}S \tag{3.7}$$

with $\vec{x}_v \in \mathbb{V}$, $C \subset \mathbb{V}$ compact and ∂C the piecewise linear margin of boundary S of $\mathbb{V}$.

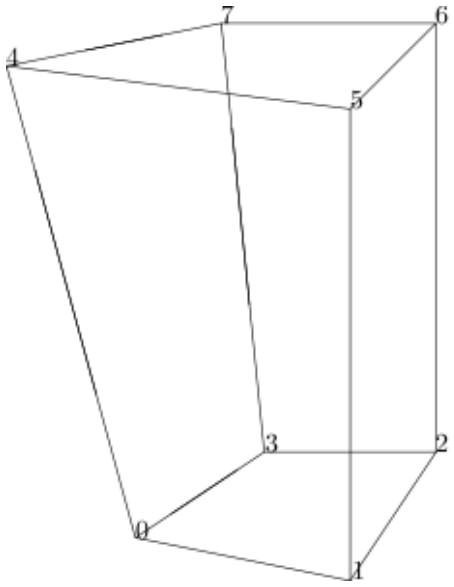


Figure 3.2: Sketch of hexahedral cell with indexed corner points.

Proof. omitted here □

It states that instead of computing the rotation within a cell, it is equivalent to integrate over the surfaces and gain the rotation's effect on the complete cell. Therefore, we need to derive a parametrization for the surfaces to solve the right-hand side of the integration 3.7. In the following this is shown for a hexahedron ($p = 7$) exemplary (see Figure 3.2 for the indexing of the cell's corner points). First, the set of surfaces $\mathcal{S}$ for the given cell $\mathcal{C}_v$ is defined as following:

$$\mathcal{S}_{\mathcal{C}_v} := \{(0,1,5,4),(1,2,6,5),(2,3,7,6),(3,0,4,7),(0,3,2,1),(4,5,6,7)\} \tag{3.8}$$

Note that the corner points are indexed counterclockwise with respect to the normal pointing outwards of the cell. The set consists of linear elements which are required for Theorem 3.1. For each of the elements the right hand side integral of Equation 3.7 is computed by

$$\int_{\partial C} \gamma(\vec{x}_v)\,\mathrm{d}S = \sum_{s\in\mathcal{S}} \int_{s} \gamma(\vec{x}_v)\,\mathrm{d}S \tag{3.9}$$

The faces are decomposed into line segments to compute the rotation over a surface. Therefore, the computation of the integral in Theorem 3.1 is simplified to the computation of line integrals.

Theorem 3.2. *Line Integral [70]*
For the vector field $\mathbb{V}$*, which is compact subset of* $\mathbb{R}^n$*, the line integral along a piecewise smooth curve* $\vec{p} \in \mathbb{V}$ *is defined as:*

$$\int_p \gamma(\vec{p})\, d\vec{p} = \int_a^b \gamma(\vec{p}(t)) \cdot \vec{p}'(t)\, dt \tag{3.10}$$

where $\vec{p}(t)$ *is a bijective parametrization of* $\vec{p}$ *with* $\vec{p}(a) \leq \vec{p}(t) \leq \vec{p}(b)$*.*

Proof. omitted here □

The line integral must be computed for every line segment, but without loss of generality this is shown in example for just $(0, 1, 5, 4) \in \mathcal{S}_{\mathcal{C}_v}$. For $(0, 1, 5, 4) \in \mathcal{S}_{\mathcal{C}_v}$ the set of boundary lines $\mathcal{L}$ is defined as:

$$\mathcal{L}_{(0,1,5,4)} := \{(0, 1), (1, 5), (5, 4), (4, 0)\} \tag{3.11}$$

The elements of $\mathcal{L}_{(0,1,5,4)}$ are forming a closed curve, which is piecewise linear and spinning the surface counterclockwise. With those requirements, the line integral for the complete path is given by the elements of $\mathcal{L}_{(0,1,5,4)}$ and for every line segment connecting $\vec{x}_i, \vec{x}_j \in \mathcal{E}$ the interpolation is given by:

$$p_{(i,j)}(t) := \vec{x}_i + t \cdot (\vec{x}_j - \vec{x}_i),\ t \in [0; 1] \text{ and } (i, j) \in \mathcal{L}_{(0,1,5,4)} \tag{3.12}$$

$$\frac{\mathbf{d}p_{(i,j)}(t)}{\mathbf{d}t} = \vec{x}_j - \vec{x}_i \tag{3.13}$$

After defining the closed curve for the integration, the data vectors $\gamma(\vec{x}_w), \vec{x}_w \in \{\vec{x}_i + t \cdot (\vec{x}_j - \vec{x}_i)\} \notin \mathbb{V}$ must be computed as well, because they are not given explicitly. Unfortunately, they cannot be computed by linear interpolation like the previous curve, because the approximation of the function $\gamma(\cdot)$ must be at least continuously differential twice to fulfill the constraints of the previous theorems. Therefore, they are interpolated by cubic splines, as is defined by a third order polynomial q [6]:

$$q_i = (1 - t)y_{i-1} + ty_i + t(1 - t)(a_i(1 - t) + b_i t) \tag{3.14}$$

$$\text{where : } k_0 = q_1'(x_0),\ k_i = q_i'(x_i) = q_{i+1}'(x_i),\ i = 1, ..., n - 1,\ k_n = q_1'(x_n)$$

$$t = \tfrac{x - x_{i-1}}{x_i - x_{i-1}},\ a_i = k_{i-1}(x_i - x_{i-1}) - (y_i - y_{i-1}),\ b = -k_i(x_i - x_{i-1}) + (y_i - y_{i-1})$$

Because three points $x_i,\ i = 0...3$ with their functional values $\gamma(x_i) = y_i,\ i = 0...3$ are necessary to compute such splines, the neighboring cells must be included in the spline construction. These neighboring cells share four common points with the cells, for which the rotation is supposed to be computed. For instance, to compute the spline for the segment $(0,1) \in \mathcal{L}_{(0,1,5,4)}$, the corner point $1'$ along with its functional value $\gamma(1')$ from the neighboring cell is used. The neighboring cell $\mathcal{C}_n$ and the cell $\mathcal{C}_v$ share a common side, to which the segment $(0,1)$ is orthogonal, i.e., $\mathcal{S}_{\mathcal{C}_v} \ni (1,2,6,5) = (0,3,7,4) \in \mathcal{S}_{\mathcal{C}_n}$. The spline's parameters for the line segments $q_1 = (0,1)$ and $q_2 = (1,1')$ are calculated:

$$\begin{aligned} a_1 &= k_0(x_1 - x_0) - (y_1 - y_0) \\ b_1 &= -k_1(x_1 - x_0) + (y_1 - y_0) \\ a_2 &= k_1(x_2 - x_1) - (y_2 - y_1) \\ b_2 &= -k_2(x_2 - x_1) + (y_2 - y_1) \end{aligned} \tag{3.15}$$

Now all requirements to solve Equation 3.10 are fulfilled. Inserting Equations 3.12 and 3.13 into Equation 3.10 along with the interpolated functional values of q_1 obtains the desired result: the rotation over every cell of the vector field without computing the partial derivatives.

For a compound of coherent cells, it would be very costly to repeat this computation. However, because of the construction of the surfaces $\mathcal{S}$, this is not necessary. The rotation value of the individual cells forming the feature can easily be summed up. For two cells sharing a common face, $\mathcal{I}_{C_1} \cap \mathcal{I}_{C_2} = \mathcal{B} \cong \mathbb{R}^2$, the face $\mathcal{B}$ is element of both $\mathcal{S}_{\mathcal{C}_1}$ and $\mathcal{S}_{\mathcal{C}_2}$.

$$\mathcal{S}_{\mathcal{C}_1} := \{(0,1,5,4),(1,2,6,5),(2,3,7,6),(3,0,4,7),(0,3,2,1),(4,5,6,7)\} \tag{3.16}$$

$$\mathcal{S}_{\mathcal{C}_2} := \{(0,1,5,4),(1,2,6,5),(2,3,7,6),(3,0,4,7),(0,3,2,1),(4,5,6,7)\} \tag{3.17}$$

without loss of generality

$$(1,2,6,5) \in \mathcal{S}_{\mathcal{C}_1} \equiv (3,0,4,7) \in \mathcal{S}_{\mathcal{C}_2} \tag{3.18}$$

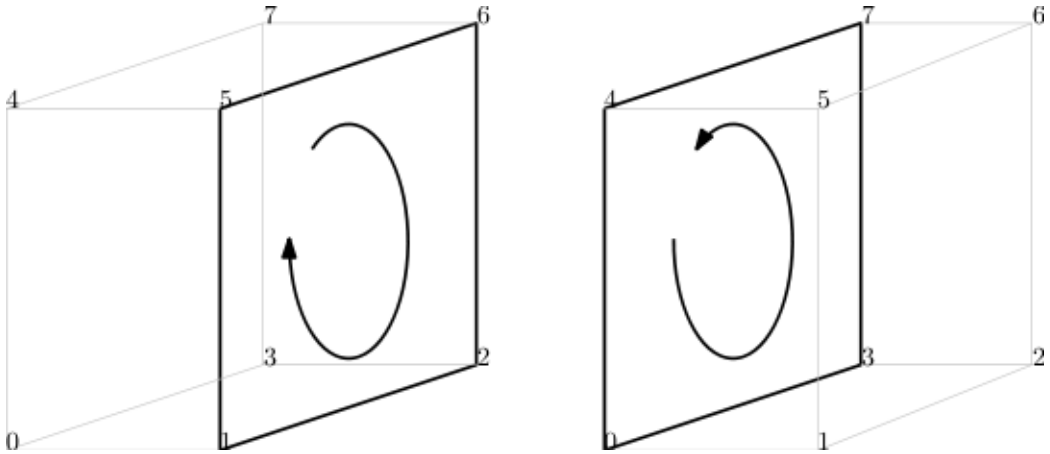


Figure 3.3: Two adjacent cells of the same grid. In both, the closed curve used for integration is highlighted. Because the surface normal is always pointing away from the cell's center, the circulation of the index points is reversed.

Thus, the line segments of $\mathcal{L}_1$ can be identified with elements of $\mathcal{L}_2$:

$$\{(1,2),(2,6),(6,5),(5,1)\} \in \mathcal{L}_1 \equiv \{(3,0),(0,4),(4,7),(7,3)\} \in \mathcal{L}_2 \tag{3.19}$$

See Figure 3.3 for a schematic illustration. The identical points of set $\mathcal{L}_2$ are replaced by their corresponding elements of $\mathcal{L}_1$:

$$\{(3,0),(0,4),(4,7),(7,3)\} \rightsquigarrow \{(2,1),(1,5),(5,6),(6,2)\} \tag{3.20}$$

Comparing this term with $\mathcal{L}_1$, the lines appear in opposite order than in $\mathcal{L}_2$. Because of the linearity of integrals, i.e., $\int_a^b = -\int_b^a$ the integral over the composed cell is equivalent to the sum of the integral over the individual cells.

Divergence

Analogous to rotation, divergence is also an operator applied on vector fields. The resulting scalar field is indicating if the vector field contains sources or sinks, as seen in Figure 3.1(b).

$$\begin{aligned} \text{div } \gamma(\vec{x}_v) := \nabla\, \gamma(\vec{x}_v) &= \frac{\partial}{\partial x_1}\gamma(x_1) + \frac{\partial}{\partial x_2}\gamma(x_2) + \frac{\partial}{\partial x_3}\gamma(x_3) \\ \vec{x}_v &= (x_1, x_2, x_3)^t \in \mathbb{V} \end{aligned} \tag{3.21}$$

Instead of solving the partial derivatives, the divergence over a cell is computed by applying a theorem of vector analysis. The so called divergence theorem or integral theorem (applied on three dimensions):

Theorem 3.3. *Divergence Theorem [52]*
For the vector field $\mathbb{V}$*, which is compact subset of* $\mathbb{R}^n$*, the divergence can be represented by:*

$$\int_C \operatorname{div} \gamma(\vec{x}_v)\, \mathrm{d}C = \int_{\partial S} \gamma(\vec{x}_v)\ \cdot \vec{n}\, \mathrm{d}S \tag{3.22}$$

with $\vec{x}_v \in \mathbb{V}, C \subset \mathbb{V}$ *compact,* ∂C *the piecewise linear margin of boundary* S *of* $\mathbb{V}$ *and* $\vec{n}$ *the normal vector of boundary* S*.*

Proof. omitted here □

The previous argumentation can adapted to solve for Equation 3.22. For the divergence, the normal $\vec{n}$ of the boundary S must be calculated additionally compared to Equation 3.7. Indeed, this can easily be achieved by taking the cross-product of two lines connecting three data points, which are already specified for the parametrization of the piecewise linear curve. The latter adds $\vec{x}_i \times \vec{x}_j,\ (i,j) \in \mathcal{S}_\mathcal{C}$ as a factor to Equation 3.10.

Cell type and Size

The type of the cell is defined by the number of its corner points. During the adaptive grid refinement, cells are split into several parts, usually tetrahedrons, to adapt their size. Therefore, the cell type can identify regions in which the cells were adapted to fit the simulation's demands. This property does not require any additional computation, because it is already given by the input data format. If a feature is formed by cells of different types, the smallest number of corner points is used for the whole feature.

The grid's cells may appear in different shapes and especially different sizes. For some numerical solver, grid refinement strategies are applied to increase the resolution in region of larger interest. For regions close to a flow singularity, smaller cell sizes are used to represent the fast changing velocity vector better. And in regions of homogeneous flow, larger cells are applied to reduce the necessary storage. So for unstructured grids, the cell sizes give details on their importance for the representation of the vector field.

Several methods are possible to estimate the volume of a cell, e.g., a rotation ellipsoid spanned by the main rotation axis of the individual cell or circumscribed spheres for every cell. Since grids can be formed by several ten thousands of cells, the computation of the volume does not have to be costly. For every cell $\mathcal{C}_v \in \mathcal{C}$ a simple bounding box completely enclosing the cell is constructed.

$$V_{C_v} := \prod_{i=1}^{3} (max\{x_{a_i}, \ldots, x_{p_i}\} - min\{x_{a_i}, \ldots, x_{p_i}\}) \tag{3.23}$$
$$\{\vec{x}_a, \ldots, \vec{x}_p\} \in \mathcal{C}_v,\ \vec{x}_a = (x_{a_1}, x_{a_2}, x_{a_3})^t$$

This computation can be performed very cheaply for every cell and the volume is approximating the cells' quite well. Other methods would do better but be too expensive computationally, as well.

Additional data

For simulations and especially for measurements, more data is collected than just the velocity vector, e.g., for a proper simulation of the lid-driven scenario, the pressure. This data is also labeling every data point $\vec{x}_v \in \mathbb{V}$, and the average over all corner points, which are forming the cell, is taken to obtain the correspondent cell's feature vector components.

3.2 Experimental Evaluation

For the experimental evaluation, two data sets are introduced here, the generic backward step and the lid-driven cavity. The generic backward step mimics the simulation of the backward step, which is, as well as the lid-driven cavity case, a well-described and often used simulation for benchmarks. The generic backward step is a fully synthetic data set, for which the ground-truth information is known. The second data set is the lid-driven cavity data set, which is solved by OpenFoam [1]. On the data sets different characteristics are examined. The generic backward set is too simple and provides only a proof of concept. The real evaluation and comparison of the results is performed on the lid-driven cavity data set.

The proposed automated feature extraction by clustering, which uses the previously defined feature vector is applied on both scenarios. Before the actual feature

[1] The open source CFD toolbox: http://www.openfoam.com/

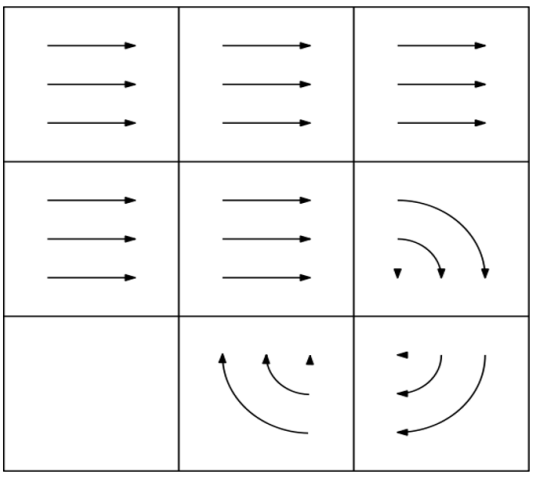


Figure 3.4: Setup of the generic backward step simulation. The test chamber is split into nine regions in which the corresponding flow is visualized by glyphs. The simulation is three-dimensional and the shown in projection to $x - y-$plane.

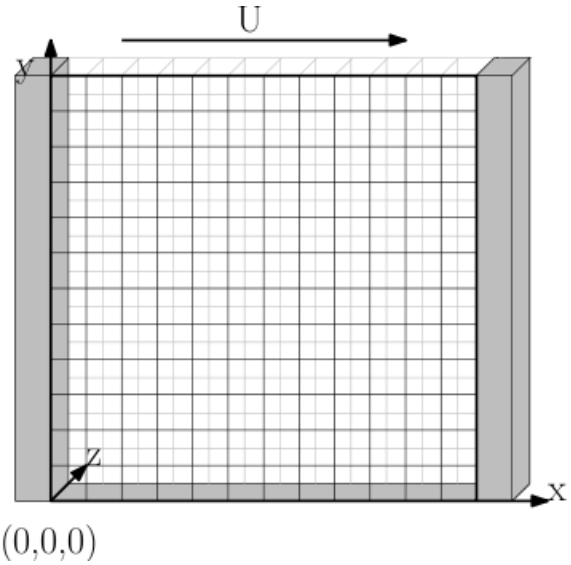
extraction, some pre-processing steps are described. These are necessary for the application of the clustering and improve its quality. In the second part of the evaluation, the results are compared to those of the visual feature extraction. The comparison considers two aspects, the quality of the extraction and the time spent to obtain it.

Generic backward step

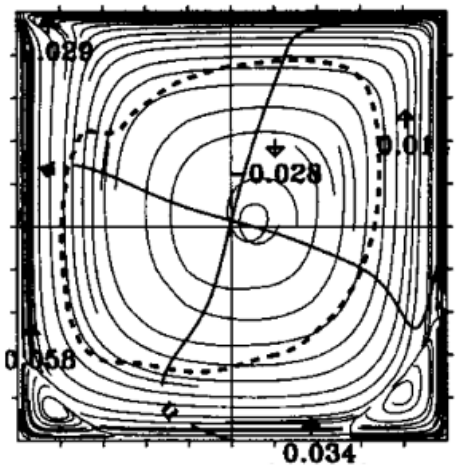
The grid is split into nine regions, in which the velocity vectors have been manipulated manually as indicated in Figure 3.4. The main flow direction is from left to right. In the right corner, a curl is built, which redirects the flow backwards again. The magnitude of the velocity vectors of the data points inside the curl is the same as for the data points outside the curl and in the part of homogeneous flow. In the lower left corner an obstacle is modeled and the flow is zero consequently. This obstacle gives the scenario its name, because the flow is facing a backward step, which causes the curl. This simulation mimics the backward step, coming later. In this setting, the generic version is used for the evaluation, because ground-truth information exists for it.

Lid-driven cavity

The lid-driven cavity is a very basic simulation, which is well understood and described theoretically [24]. In Figure 3.5(b), the expected results are shown. Along

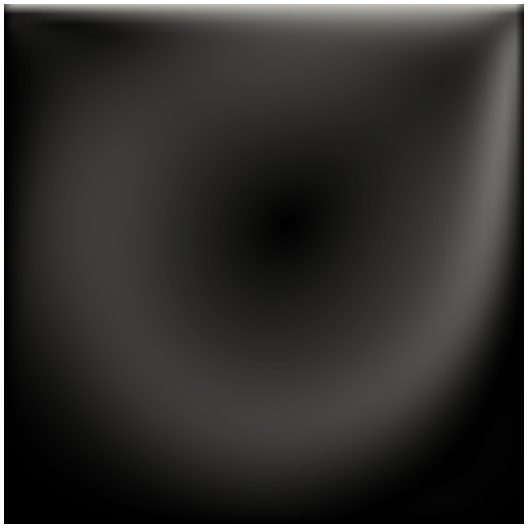


(a) Test chamber of the simulation with the corresponding reference frame

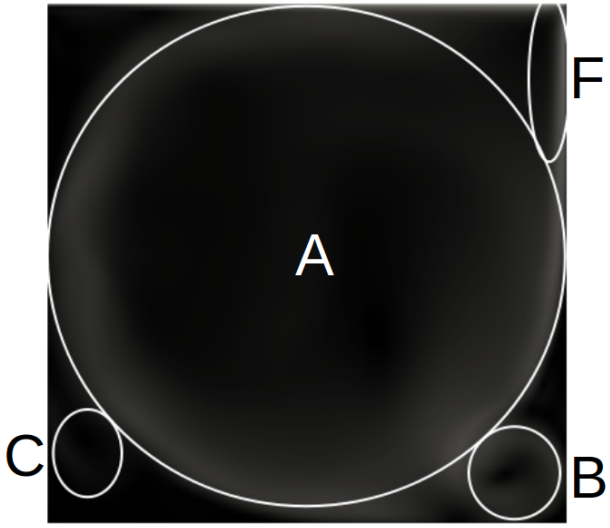


(b) Expected results for the lid-driven cavity simulation. Taken from Chiang et al. [33]

Figure 3.5: Setup for the lid-driven cavity simulation. The walls of the test chamber are fixed with the exception of to top one, which is sliding from the left to the right with a constant velocity.



(a) $Re_1 = 1000$



(b) $Re_2 = 167$

Figure 3.6: Steady state for the lid-driven cavity for two different Reynolds numbers. On the $x - y-$plane the heat map visualizes the velocity vectors. The brighter the color, the larger the magnitude of the corresponding velocity vector. For an appropriate choice of the Reynolds number secondary curls are formed in regions B and C.

with the backward step, this is a standard test scenario for newly developed numerical solvers.

The setup for this simulation is more complex than the previous backward step. Different to the backward step the test chamber is closed for the lid-driven cavity simulation, i.e., no flow from outside affects the simulation. The walls of the surrounding chamber are all fixed, except the top one, which moves from left to right. Because of the friction between this moving lid and the fluid inside the chamber, a flow on the inside is caused. The properties of this flow are dependent on the speed of the moving lid and the fluid's viscosity.

For the simulation, the Reynolds number $Re = 167$ is used because of the additional difficulty to detect the secondary curls [33]. The Reynolds number is defined as the ratio of inertial forces to viscous forces and consequently defines the properties of the fluid and its behavior under these conditions [50]. In Figure 3.5(b), the expected result for the given parameters is shown and Figure 3.6(b) shows our simulation on which we base our experiments. The evaluation focuses on four distinct regions, which are marked in Figure 3.6(b). In region F, the particles have a high velocity due to the acceleration on the top lid. Additionally, they are reflected on the right wall, which causes a rapid change in the orientation of their velocity vectors. Therefore, clusters of just a few cells are expected. The energy brought into the system forms the primary curl A, where the orientation of the velocity vectors changes slowly and clusters formed by more cells than in region F are expected. Those two regions are common to all lid-driven cavity simulations. Regions B and C are the so called secondary curls. Region B can be regarded as a smaller primary curl than A. Especially the border between both zones is challenging to identify, because the vectors on both sides expose a very similar behavior. The same situation is found between A and C. In contrast to the curl in B, the secondary curl C is much weaker.

3.2.1 Feature Extraction by Clustering

Clustering algorithms can work on huge data sets of millions of high-dimensional data points efficiently. However CFD data sets are adding new constraints. As mentioned already, the utilized data sets are represented by a grid of cells, which puts them in context with each other. So, instead of regarding the data points as a loose set of points, their neighborhood is important for the segmentation. Additionally, the data points are distributed continuously not only in the domain of the vector field,

as their data vectors are continuous, as well. As a result, the data is not noisy, i.e., the outliers identified are not noise, but points of interest.

Pre-Processing

The cell's feature vector is high-dimensional and formed by different components (Definition 3.1). Two major pre-processing steps are performed. In the first step, the influence of each component is estimated and the data set is projected down to the subspace of significant components. The second step normalizes the components because they typically differ by magnitudes. By the normalization to $[0; 1]$, respectively $[-1, 1]$ for non-positive components, the influence of every single component on the results is easier to distinguish. The latter fact allows to apply a weighting to the data set to increase the influence of individual components on the overall result.

Feature selection for clustering is commonly used to reduce the dimensionality of the data input, which leads to both a reduction in runtime and an improvement in clustering quality. This is caused by the simplification of the comparison between the data points, which is done by applying a metric. For increasing dimensions, the metrics steadily loose their significants. This effect is the so-called curse of dimensionality. Consequently, fewer dimensions lead to better results and the smaller the dimension of the data points is, the faster is the comparison.

Feature selection

Several different techniques are proposed to reduce the dimension of the input data [61]. This is motivated by the focus on significant components of the feature vector and aims to obtain a better clustering in a shorter time. Wrapper model and hybrid approaches are not applicable this scenario, because they rely on the comparison of the resulting segmentation, which is obtained by iterative clustering with different entries of the feature vector. The resulting runtime is too high and not reasonable here.

Filter models form the third class of feature selection methods. In contrast to the previous approaches, they do not require a clustering step to evaluate the influence of the components. The naive approach is to remove unnecessary components by simple statistics. For instance, the cell type is often identical for all cells and can be excluded without influencing the clustering result.

More complex filter models are based on Principal Component Analysis (PCA), forward modeling and backward modeling. PCA regards the components of the fea-

ture vector as univariate, i.e., they are independent of each other. By PCA, the data is mapped into a new space, which is spanned by the most significant entries of the feature vector. To reduce the number of dimensions, a subspace is chosen, e.g., the space spanned by the five most significant components. Forward modeling and backward modeling assume that the components are multivariate, i.e., depending on each other. In our context, we experiment with a linear model to fit the data, because this feature selection technique promises the best results here.

Normalization

Normalizing the components of the cells' feature vector (Definition 3.1) is straightforward since it is formed by numeric values only. Indeed, the components may differ in magnitudes between each other, e.g., the velocity vector is quite large in comparison to the divergence or rotation over a cell. To make the components comparable to each other, they are normalized. Non-negative components are mapped to $[0; 1]$ and arbitrary components to $[-1, 1]$ in order to keep the signature. The mapping is given by:

$$\begin{aligned} min_{old} \leq x_{old} \leq max_{old} \quad \text{onto} \quad max_{new} \leq x_{new} \leq max_{new} \\ x_{new} := \frac{x_{old} - min_{old}}{max_{old} - min_{old}}(max_{new} - min_{new}) - min_{new} \end{aligned} \tag{3.24}$$

and in this specific context

$$\begin{aligned} 0 &\leq min_{old} \leq x_{old} \leq max_{old} : \\ x_{new} &:= \frac{x_{new} - min_{old}}{max_{old} - min_{old}} \qquad (3.25) \\ zmin_{old} &\leq x_{old} \leq max_{old} : \\ x_{new} &:= \frac{x_{old} - min_{old}}{max_{old} - min_{old}}(max_{new} - min_{new}) - min_{new} \qquad (3.26) \end{aligned}$$

Choice of Norm

For many clustering algorithms, a norm must be specified in order to compare the data points among each other and to compute their similarity. The choice of the norm affects the quality of the clustering results heavily. CFD data sets consist of numeric

algorithm	cluster models	applicability
k-means	centriod / partitioning	?
k-medoids	centriod / partitioning	?
DBSCAN	density	x
Optics	density	x
HiCo	density / hierarchical	x
EM algorithm	distribution	(✓)
CLIQUE	subspace	?
GridClus	centriod / partitioning	?
GNDBSCAN	density	x
OptiGrid	centriod / partitioning	?
CLINK	hierarchical	✓
CuRe	hierarchical	(✓)
BIRCH	hierarchical	✓

Table 3.1: Potential clustering algorithms for application to our scenario. The quality of the obtained results is indicated by symbols. The segmentation is computed correctly according to our feature definition (Definition 2.6) : ✓ for all settings; (✓) only for optimal parameters; ? dependent on input data; x never.

data only. Thus, it is sufficient to focus on norms for numbers only. A major class of norms are the so-called p-norms where $p \in [1; \infty[$:

$$\|\vec{x}\|_p := \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p} \tag{3.27}$$

The domain of the vector field is the Euclidean space, in which the cell centers are embedded (Definition 3.1). Experiments showed, that the shape of the cluster may vary depending on the choice of p. For $p = 1$ the clusters are stretched along the coordinate axis of the spatial space. To decrease this influence, the cells' centers and the velocity could be excluded from the feature vector but this would exclude the major properties of the data points. Therefore, the Euclidean norm $p = 2$ is applied in the following as the most intuitive one for Euclidean space.

Clustering algorithm

Different classes of clustering algorithms are applied to this specific scenario. Each of the classes has its advantages and disadvantages over the others and therefore at least one representative is chosen and tested for every class. They will also be explained in the following in more detail. In Table 3.1 the tested clustering techniques are summarized briefly, before we discuss them in detail.

Partitioning clustering algorithms

This class is probably the best studied amongst all clustering algorithms. Along with the input data set, the algorithms need an initial set of seeds for every cluster. The data points are grouped around the seeds for partitioning the data set into clusters.

k-means [92] partitions the data set into k clusters, such that the variance between the cluster's members and the cluster's seed is minimized

$$Var(Clustering(k, \vec{s})) = \sum_{i=1}^{k} \sum_{\vec{x}_v \in \mathbb{V}} \|\vec{x}_v - \vec{s}_i\|_2^2 \tag{3.28}$$

with $\vec{s}_i \in \vec{s}$ are the k seeds. This problem can be seen as the Least Square Problem which can be solved efficiently. It can be applied to large data sets, but because of the Euclidean norm it suffers heavily from the curse of dimensionality. Consequently, for clustering high-dimensional data sets, k-means is not the optimal choice. The input parameter k must be specified as well, but k is unknown initially. Typically the user runs the algorithm several times with different setting for k and estimates the best choice for it manually. By picking the input seeds at random a non deterministic behaviour of the algorithm is obtained. Additionally k-means is only able to extract ellipsoidal clusters, which is a major limitation for clusters of unknown appearance.

We tested two variants of the k-means algorithm, namely the Llyod [92] and the MacQueen approach [97]. The resulting partitioning of the data points was obtained in very short time and the quality was good. With the disadvantage of only finding ellipsoidal clusters and the problems to determine k, we searched for more suitable algorithms.

k-medoids [80] is very similar to k-means. They differ only in picking the seeds. k-medoids chooses k points of the data points as seeds. The results are very similar, though. This algorithm suffers from the same disadvantages as k-means, unfortunately.

Density-based clustering

This class introduces a different and new concept, the so called density-reachability. Two points are density reachable if they are connected by a link of density connected points, i.e., two points within a certain range. Because of the density-reachability the density-based clustering algorithms are stable to noise and outliers. Additionally, the number of clusters does not need to be specified a priori and the clusters itself can be of arbitrary shape.

DBSCAN [47] is the best studied representative of this group. It introduced the concept of density reachability. The distance threshold is specified by *eps* as input parameter for the algorithm. The second input parameter is *minpts*, which specifies the minimal number of points inside the sphere of radius *eps* for a data point to be a core object.

DBSCAN is robust against noise, which is of little importance for our scenario. We assume very little noise, because the data is retrieved from numerical simulations. We even observed that this robustness against noise is disadvantageous in our context. Because of the different cell sizes, the distance between the cell centers differs, too. Therefore, large cells, which actually form a region of homogeneous flow, are identified as noise and ignored by the clustering algorithm. This effect largely influences the optimal choice of *eps*. Larger values of *eps* are necessary to group cell centers of homogeneous flow properties, but small values are necessary around singularities. On the other hand, many cells are situated close to a singularity. Therefore, *minpts* must be chosen carefully and has to be smaller than in regions of homogeneous flow.

Finding the best balance of both parameters is crucial for the quality of the clustering, but remains very challenging for our scenario. We need to work with a rather large value for *eps*, which also influences the runtime of DBSCAN. Overall the clustering is performed in a reasonable time, but we need to improve the parameters iteratively.

The experiments with DBSCAN showed that the initial parameter *minpts* must be set very small (<5) in order to find the minute features as well as the partitional clustering algorithms. Being able to split the neighborhood of flow singularities correctly and not to segment regions of homogeneous flow into several parts inflicts two contradicting constraints on the optimal choice of ϵ.

Optics [13] is based on DBSCAN and is supposed to have a shorter run time. In the experiments, we face the same problems finding a suitable setting for the input parameters. The choice of *eps* especially, which is influencing the performance of Optics significantly. We need to set *eps* very large, which slows down the performance of the algorithm heavily. HiCo [9] is based on DBSCAN and specialized to find clusters in arbitrary oriented subspace of the input data set. Therefore, it would be suitable to deal with a high model complexity. However, we face the same problems as with the other density-based clustering algorithms.

Distribution-based clustering

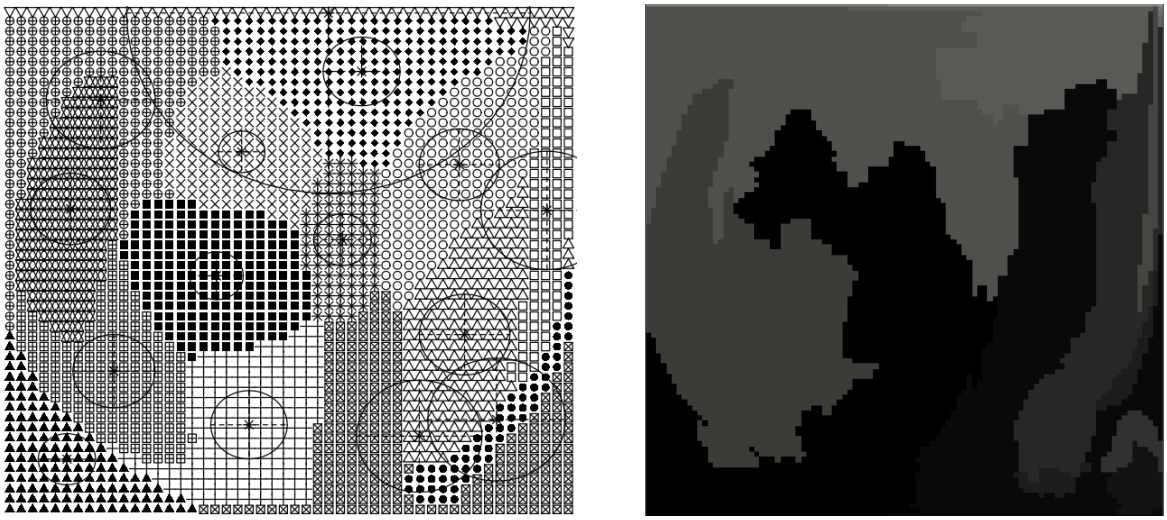
These cluster techniques assume that each found cluster belongs to a different statistical distribution. Therefore, it is assumed, that the input vector field can be represented by a set of distributions. The distributions defined initially are matched to the data by adapting their properties and changing their parameters. Distribution-based clustering methods usually require at least two input parameters. First, the types of the underlying distribution, which can be any statistical distribution, e.g., mixture of Gaussian distributions. The choice of the distributions is dependent on the assumed process that formed the data. For most natural processes, Gaussian distributions can be assumed. The second parameter specifies the number of expected distributions forming the data set, which must be chosen carefully to prevent overfitting.

As representative for this class, we chose the expectation-maximization algorithm [41] working with a mixture of Gaussians. We tested different numbers of distributions and the resulting segmentations were promising. The algorithm was able to detect clusters of different sizes, which results in a good segmentation of the neighborhood close to singularities. However, compared to the segmentations obtained by other clustering algorithms, the resolution was not equally well, i.e., the clusters found by the expectation-maximization algorithm tend to ignore small changes of the data within a cluster. This could be explained by the robustness of the algorithm to noisy data. This effect is visible in the lower right corner of the individual images in Figure 3.7. The resolution would converge by increasing the number of underlying distributions, but we would face the problem of overfitting the data.

Subspace clustering

This class is specialized on high-dimensional data sets and able to find clusters in subspaces of it. Kriegel et al. [85] and Parson et al. [105] summarized recent approaches. Subspace clustering algorithms work very efficiently on very high-dimensional data sets, e.g., to evaluate microarray chips for gene expression analysis [85].

In contrast to microarray data, vector field data sets are structured by the underlying spatial grid and have fewer dimensions. The vector field domain also simplifies the search for clusters, hence a suitable representation for the data is given by the coordinate axis. Additionally to the special structure of the vector field domain, the flow properties have special properties, too. They change steadily in all components of the feature vector over spatial space. And therefore, members of different clusters



(a) Expectation-maximization algorithm (b) Hierarchical clustering

Figure 3.7: By application two different algorithms, the steady state of the lid-driven cavity simulation is segmented into 15 clusters.

are distinguishable in all their components of the feature vector, i.e., all components are important to detect the cluster and not only few, forming a subspace.

Building a subspace over the feature vector also counteracts the derivation of additional properties to enrich the description. In Figure 3.14, a comparison of the clustering results between different subsets of the feature vector is provided. We applied hierarchical clustering, because in this scenario, subspace clustering techniques, such as CLIQUE [10], take all available components of the feature vector into account. This is necessary because of the special properties of the data set. Based on this results, we decided to pre-process the input data and select relevant features instead of using a subspace clustering.

Grid-based clustering

This class is particularly developed to process large data, too. Different to the subspace clustering techniques, not the dimensionality of the data is reduced but its physical size. The data set is segmented into blocks which represent the data points on the inside. This reduces the number of comparisons necessary to build the clusters. For instance the nearest neighbor search is simplified by comparing the data point with the block instead of comparing it to all data points inside the block. GridClus [120] applies this strategy. It performed faster in the experiments than k-means,

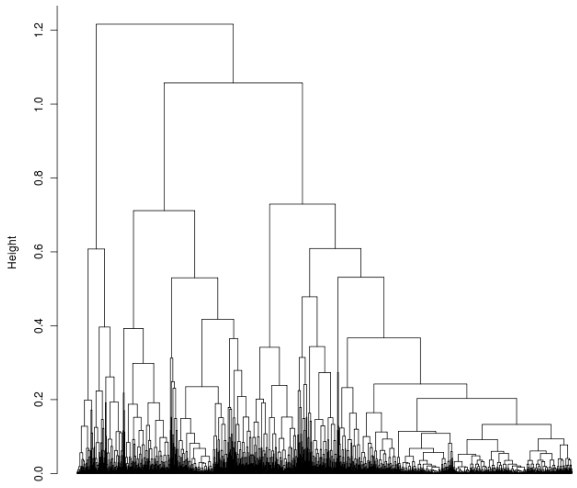


Figure 3.8: Dendrogram for the lid-driven cavity simulation at its steady state. The tree-like structure indicates the splits/merges of two clusters between consecutive levels of the hierarchy.

which is also based on a neighbor search. Indeed, both approaches share the same disadvantages of finding convex cluster.

GNDBSCAN [74] can be seen as an adaption of DBSCAN to a data set which is segmented by a grid. Again, the runtime of the grid-based algorithm is faster, but since both algorithm are constructed on the principle of reachability, we observed the same drawbacks in our experiments. GNDBSCAN detected critical points as noise and tuning the parameters was as challenging as for DBSCAN.

While running the experiments with OptiGrid [71], we examined the main problems of grid-based clustering techniques in this scenario. OptiGrid builds the blocks in correlation to the data points distribution. This is very advantageous for data sets in which the data points are concentrated in several spots, e.g., astronomic data sets. In our particular scenario, the data points are distributed more equally, which disfavors grid-based clustering techniques.

Hierarchical clustering

Hierarchical clustering techniques build a dendrogram over the input data set which represents the clustering for every possible number of clusters. Therefore, the a-priori problem of choosing the optimal number of clusters, e.g. k-means, is now a-posteriori one, i.e., the optimal cutting level of the dendrogram can be estimated after the clustering process (see Figure 3.8 for an example dendrogram).

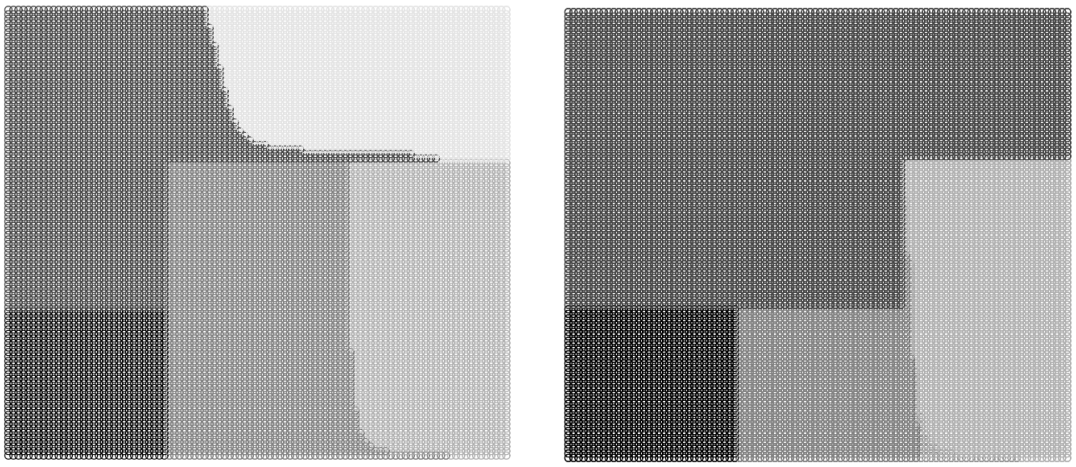
For the highest level of the hierarchy, the complete vector field is represented as complete segment, for the next lower level, this segment is split into two, such that

every cell of the feature is more similar to those inside the segment than to cells outside it. The splitting is repeated iteratively until every cell forms a feature of its own. The feature hierarchy therefore represents the vector field in different levels of detail, i.e., the balance between abstraction and resolution of the initial data differs from level to level of the hierarchy. In the lowest level of the hierarchy, every data point is a cluster of its own, and therefore the finest resolutions is achieved on the cost of not abstracting the data at all. For the highest level the balance is vice versa.

There are two major classes of classic hierarchical clustering methods, the agglomerative and the divisive one. They only differ in the technique, how the overall all data set is treated. The agglomerative ones start at the atomic level and merge the elements until there is only one cluster left, containing all data points, and the divisive ones work vice versa. As a result the input data is represented as a dendrogram, a tree-like representation consisting of all merges or splits. The segmentation is influenced by the so called chaining phenomenon. An agglomerative hierarchal clustering algorithm tends to add new elements to the already found clusters instead of creating new ones. This phenomenon influences the result and is, in general, regarded as a disadvantage of this clustering class. However, in this certain scenario, the chaining phenomenon actually is an advantage by providing coherent clusters. The CLINK [40] algorithm applies the complete-link criterion, and the segmentation results are good.

The features situated around flow singularities are very difficult to select, because they are formed by just a few cells. This is especially difficult for clustering algorithms based on sample techniques. For instance, CuRe [59] samples the data and builds a hierarchal representation. For a naive sample of the vector field the obtained results were not good because of the problematic sample choice. For a steadily increasing sample, the results became better. Indeed, good results are achieved for sample sizes of more than 90% of the data set. A smarter way to sample the data is to search the flow singularities first and make sure those are included in the sample. This search is not straightforward, hence in general the flow singularities are not identical to data points of the vector field. Sampling this way is too inefficient for large data sets, because flow singularities in divergence and rotation must be found first by the Hodge-Helmholtz decomposition.

BIRCH [153] was developed to cluster very large data sets efficiently. By building a balanced CF-Tree make optimal use of the available memory to speed up the clustering. The experiments showed that the results of the segmentation are good and the overall computation is faster than for the classic hierarchal clustering algorithms. Therefore, we use BIRCH to cluster the data set.



(a) The feature vector consists of the velocity vector only

(b) All components of the feature vector are used

Figure 3.9: Clustering for the generic backward step by BIRCH, computed for different subsets of the cell's feature vector.

Figure 3.9 provides a first example for the presented workflow. The clustering is based on the identical generic backward step data set, only the number of components of the cell's feature vectors have been change. The clustering in Figure 3.9(a) is based on the plain cell centered representation of the vector field, which is not including divergence, rotation or other derived information, while for the clustering in Figure 3.9(b) the complete feature vector is used.

The optimal number of clusters was selected manually for both settings. When comparing the results of the clustering with the initial setup of the generic backward step shown in Figure 3.4, the results for the complete cell's feature vector is better. Including the additional flow properties increases the segmentation, especially for the part of the homogeneous flow at the top and the curl in the lower right corner. The clustering for the partial cell's feature vector segments the homogeneous part into two and incorrectly combines the mid with the curl. This is probably caused by the same magnitude of the velocity vectors of the data points inside and outside the curl. Nevertheless, by including the additional flow properties, the segmentation quality is increased.

3.2.2 Visual Feature Extraction

As mentioned already, vector field analysis is traditionally performed visually. In the following the most relevant techniques are applied on the same data set as the automated feature extraction by clustering.

In Figure 3.10, glyph visualization is shown and additionally a segmentation of the vector field by clustering the glyphs applying k-means [67] in Figure 3.11. Visualizing the given vector field by glyphs only provides the user with an overall impression of its structure. The rapid changes in region F can be identified and the primary curl A as well. However, it fails to separate region A and F, and the secondary curl in C is hardly visible. Clustering the glyphs makes the small structure in region F more visible, but due to the similar structure of A and B, both are identified as the same cluster.

Contour lines are drawn in our data set in Figure 3.12. We used the magnitude of the velocity vectors to separate the regions. Because of the rapidly changing flow values in region F, the contour lines are dense and one cannot distinguish between the individual lines. The primary curl A and the secondary curl B have very similar velocity vectors and therefore are not clearly separated by contour lines. To make the secondary curl in region C more visual distinct additional contour lines have to be added, which would making the lines in region F even denser.

Path lines are not explicitly segmenting the vector field into different parts, but the movement of the vector fields particles is highlighted. In Figure 3.13(a) we visualized the flow by path lines placed at the shown line. All structures are clearly separated and identifiable, except the secondary curl in B. This is caused by the seeding problem of the path lines. We provide two additional examples, Figure 3.13(b) and Figure 3.13(c), for the seeding problem. The seeding of the path lines is identical, except the seeds are placed just in two adjacent cells along x direction.

3.2.3 Comparison of the Results

The comparison of the results is rather difficult because the overall goal of the visualization techniques is to make features and structures more distinct on a visual level than their surrounding. Additionally, visualization techniques are user guided, i.e., they need a significant amount of user input to produce good results. Because of those two aspects, the results of the visual feature extraction cannot be compared easily to those of the automated extraction approach by clustering. In fact, the automated approaches are the exact opposite, they are executed with none to little user input

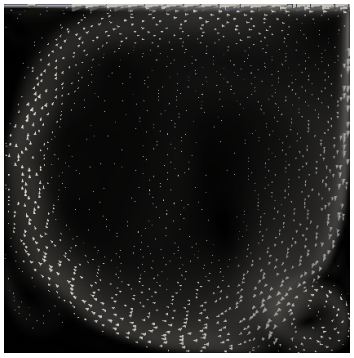


Figure 3.10: Simple glyph visualization

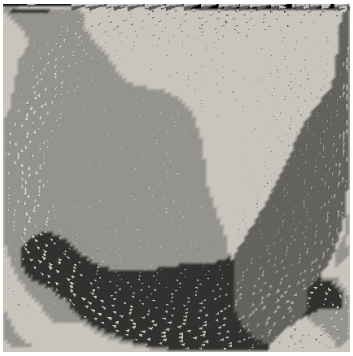


Figure 3.11: Clustered glyph visualization

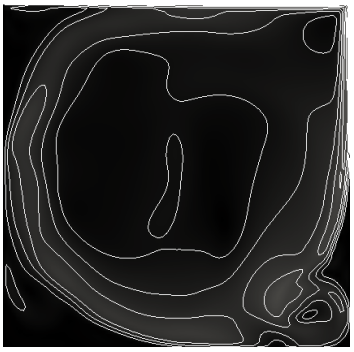
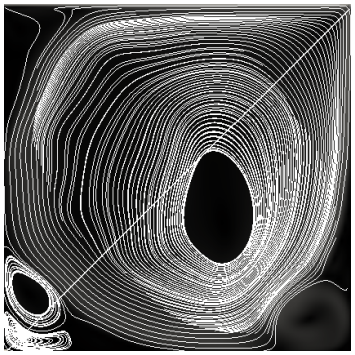
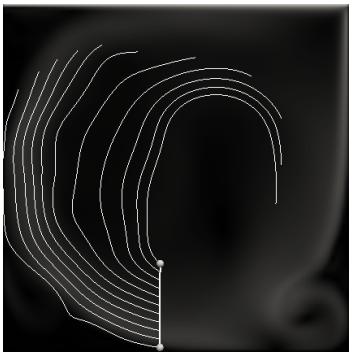


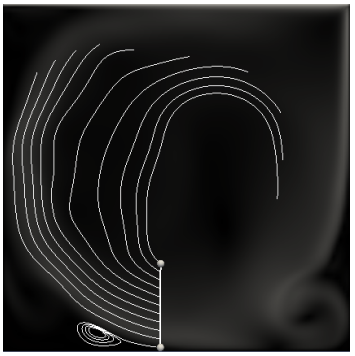
Figure 3.12: Contour line visualization



(a) Global placing of seeds



(b) Simple seeding



(c) Seeding for secondary curl

Figure 3.13: Path line visualizations for the lid-driven cavity simulation at its steady state. Different strategies demonstrate the influence of the seeds on the resulting path lines.

and the produced results are represented by numeric values rather than nice images. Nevertheless, the comparison will take both aspects into consideration, the quality of the feature extraction process and the necessary runtime for both, the visualization and the automated feature extraction by clustering.

Segmentation Quality

In contrast to clustering techniques, for which quality measurements are common [62], the quality of the visualization techniques is mostly judged manually by the user and thus Kirby et al. [83] stated the need for tools to describe the quality of the obtained results.

The segmentation for BIRCH is shown in Figure 3.14. In the columns, the number of used properties in the cell's feature vector 3.1 is constant. In the first column, the

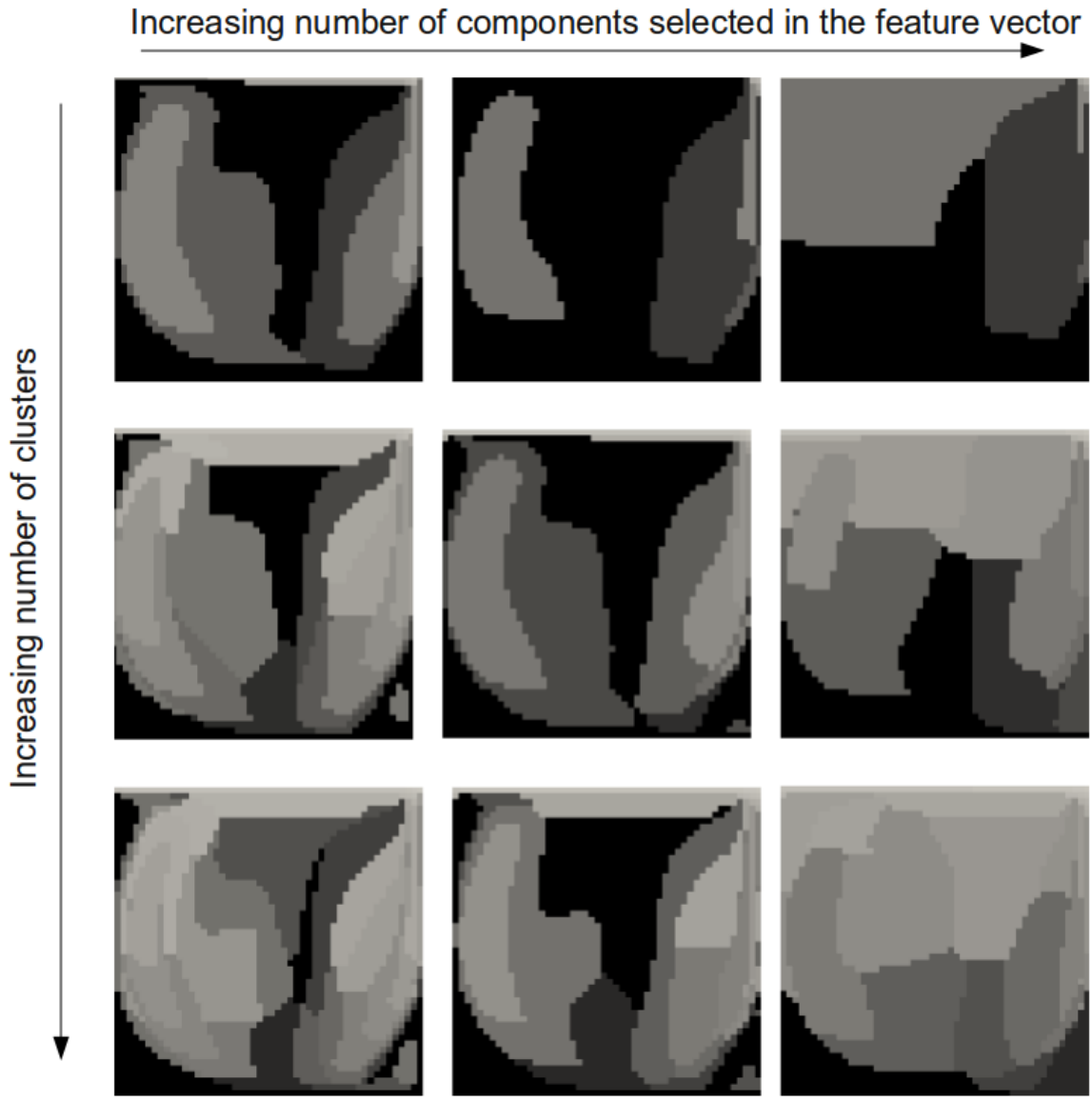


Figure 3.14: The lid-driven cavity simulation at its steady state is segmented by BIRCH according to the number of clusters and the number of the components selected in the feature vector, respectively.

clustering is only based on the velocity vector, i.e., those results use the identical data set as the visualization. As a second step, the derived flow properties are added. And finally the third column shows the results for the complete cell's feature vector, especially including the cell centers. In the rows, we increase the number of extracted segments.

Choosing the optimal number of clusters for the segmentation of the input data set is not trivial. Actually, it is still an open research question which will be subject of the next chapter-along with possible validation techniques and quality measures for clustering. Because the feature extraction is an intermediate result, we will skip the optimal number of clusters for now and give possible solution for it in context with the complete workflow.

First, we compare the proposed segmentation by clustering to the traditional visualization techniques as shown in Figures 3.10-3.12. The overall structure of the vector field is visible for all, however even the clustering with the fewest clusters gives more detailed insight than Figure 3.11, which has a similar number of segments. By increasing the number of clusters, the level of detail for the segmentation by clustering is raised.

Second, the additionally derived flow properties help to improve the segmentation quality. In the last row, the segmentation based on the velocity vectors is more cluttered than the one including all flow properties. This is visible best in the left part of the primary curl A. Here the additional flow properties prevents the segmentation by magnitude of the velocity vectors only. Therefore, the additional flow properties help to interpret the flow dynamics. By taking all cell's feature vector components into account, this effect is becoming even more obvious.

Runtime Comparison

Unlike clustering techniques, visualization methods require lots of user interaction. First, the best fitting visualization technique for every scenario must be chosen to extract the desired features and to obtain good results. Second, the data set must be explored in order to set the parameters for the visualization correctly. Iteratively, the user adapts the settings, e.g., the seeding for the path lines, and steadily improves the results of the visualization. Many specialized visualization frameworks, e.g., ParaView [2] or VisTrails [3], provide the user many standard filters and extraction techniques. The user just need to drag and drop his pipeline together and the visualization framework

[2] ParaView: http://www.paraview.org/
[3] VisTrails: http://www.vistrails.org/

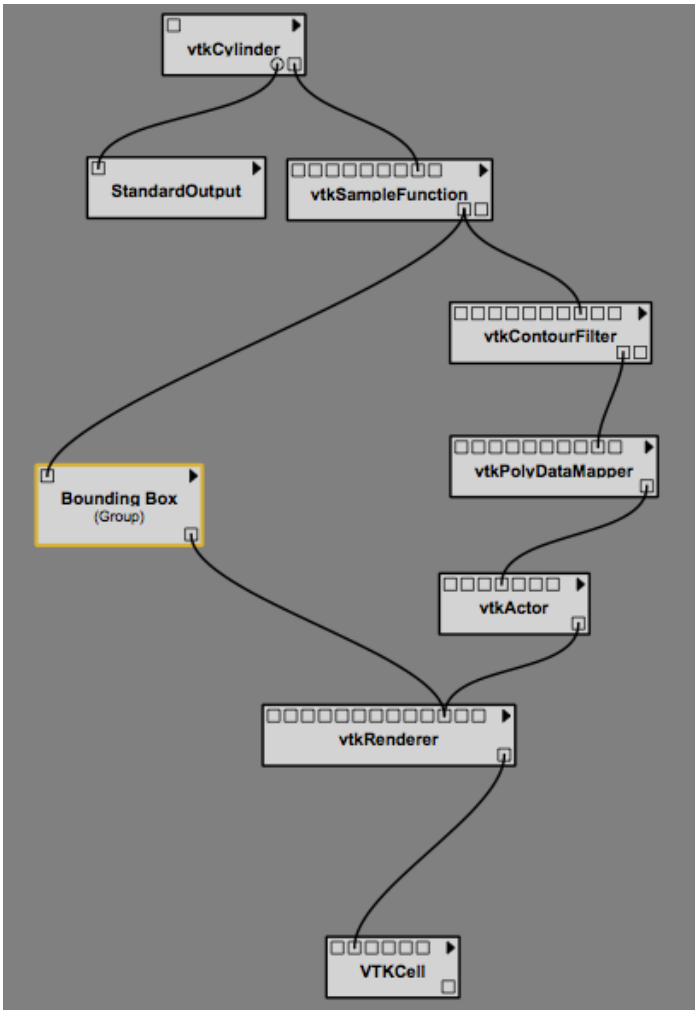


Figure 3.15: VisTrails GUI with pipeline for contour lines. Several modules are necessary for quite a simple visualization. Taken from the official documentation of VisTrails [7].

supports him to explore the parameters. Nevertheless, the user needs to construct a series of visualizations to convey the main aspects and trends of the data set.

To give the user an impression of a visualization pipeline, Figure 3.15 provides an example. In the gray-shaded center a pipeline for computing contour lines is constructed by filters. As set of modules is included in the pipeline, which are necessary for the correct rendering and managing of the data flow. The pipeline branches, the left branch computes a bounding box and the complete right one is necessary to entire the contour lines. Each of those modules has a couple of parameters and settings, which are not visible here. We do not provide additional details on the selection of the modules or the user interface as this example is just supposed to give the reader an impression on how many modules are involved in a rather simple pipeline.

The complete process requires a lot of time, in which the user is tuning the visualization manually. Therefore, the actual computation of the visual entities, e.g., path lines, is quite short in comparison to the overall time the user spends to obtain a comprehensive visualization. This aspect also influences the execution of the simulation workflow. The simulation runs automatically on large computers specialized on numerical computation, i.e., their infrastructure fits the demands of the floating point computation the best. The visualization process on the other side is user driven and requires computers capable of human computer interaction and with powerful graphics. As a result, the workflow is typically split into two iterative processes. First, the simulation is computed and, as a second step, the results are visualized on another computer to free the resources of the first for numeric simulations.

For the clustering algorithms, the situation is totally different. After specifying the input parameters, the execution is user independent and the overall runtime can be measured easily. Additionally, cluster algorithms favor the same infrastructure that is necessary for the simulation as well, i.e., the clustering can be executed automatically on the same infrastructure as the simulation which is typically very powerful.

Algorithm 1: Clustering the input vector field

```
   Data: normalized vector field
   Result: clustering
 1 for every cell c of the vector field do
 2     p = getCornerPoints(cell c);
 3     fv1 = cellCenter(coordinates of p);
 4     fv2 = cellFlow(velocity vectors of p);
 5     fv3 = cellCenterData(additional data of p);
 6     fv4 = cellType(number of p);
 7     fv5 = cellVolume(cooridnates of p);
 8     fv6 = cellDivergence(cooridnates of p, velocity vectors of p);
 9     fv7 = cellRotation(cooridnates of p, velocity vectors of p);
10     feature vector of c = (fv1, ... fv7);
11 clustering = BIRCH(features vectors of c)
```

Algorithm 3.2.3 summarized the previously defined cell's feature vector (Definition 3.1). Details on its components have been discussed above. The process is linear and depends on the number of cells c inside the vector field, i.e., $O(c)$. The computation of the individual components of the cell's feature vector is dependent on the number of corner points p forming the cells. Although this parameter can vary for unstructured grids, it can be seen as constantly smaller than eight for hexahedrons. The computation of the individual components is built on basic arithmetic operations

number of cells	10000	20000	40000
deriving data	12	22	62
clustering	2	5	12
sum	14	27	86

Table 3.2: The measured runtime in seconds for the complete feature extraction, including pre-processing and deriving of additional data, e.g., divergence or rotation among others.

like addition and multiplication. This allows a fast overall computation. Only the computation of divergence and rotations scales slightly worse in $O(clogc)$, because it requires more details on the grid and not only on the individual cell.

The BIRCH algorithm has been developed for large data sets and it consequently performs very efficiently $O(c)$ [153]. We performed runtime experiments on a series of lid-driven cavity simulations. The parameters of the individual simulations are identical except for the z dimensions, i.e., the test chamber is extended into depth to include more cells, but keep all other parameters constant. Those experiments confirmed the assumed complexity. Therefore, the presented approach scales nicely with the number of cells. Furthermore, the pre-processing can be parallelized easily, because the cell's feature vector is computed independent of other cells.

In Table 3.2, the runtimes for the elements of the lid-driven simulation series are summarized. The costs for deriving the additional data is actually higher than for the clustering but as shown previously the quality of the segmentation can be significantly improved. Furthermore, those experiments were performed on a normal desktop computer to be comparable to runtimes for visualization techniques. However, in contrast to visualization techniques, which require user interaction, the presented automated feature extraction process can be run on the same infrastructure as the simulation being more powerful than the desktop computer. The computation of the additional data is constructed to be parallelized easily potentially reducing the runtime significantly.

As mentioned already, comparing the runtime of an user guided and an automated approach is impossible, because of the human factor influencing the visualization pipeline. This factor is impossible to be measured as exactly as the runtime of an individual computer. Image yourself as a user, trying to build a visualization pipeline. First, an appropriate visualization technique must be selected. After this decision, a simple pre-processing needs to be done, e.g., normalization and filtering. On the pre-processed data, the visualization techniques are applied and they compute the

visual entities. Then, the user must validate the results. If the results are not good enough or do not visualize the desired structures, the parameters must be improved by the user. If the results are still not good enough, the user needs to reconsider his choice of visualization technique and re-enter this iterative loop. Inside the loop, the results are evaluated manually, which results in a significant user impact on the visualization.

Even in the very best case - when everything is optimal the first try - the user would have 90 seconds to setup the pipeline in order to compete with our approach (see Table 3.2). For comparison, each of the visualizations in the Figures 3.10 - 3.13 took around five minutes, the path line visualization (Figure 3.13) even longer. For more complex data sets than the lid-driven cavity, the difference between the automated extraction and the visual one becomes even more extreme. The runtime for the visualization depends on the model complexity, the automated extraction only on the size of the data set.

3.3 Summary and Discussion

In this chapter we defined the cell's feature vector (Definition 3.1) which is necessary for the application of clustering techniques. The clustering is the key in our automated feature extraction. In particular we have shown and demonstrated the following points

- Although features are defined in our context as regions of interest rather than points of interests, the definition is comparable to established ones. The cell's feature vector includes and summarizes the different definitions into one general description of features, which is used for the concluding clustering algorithm.

- Although the lid-driven cavity test case seems to be easy to cluster on first sight, the choice of the clustering algorithm is surprisingly difficult. Because the data set is not formed by a loose set of points, the relation of the points and the grid this formed grid needs to be taken into consideration for the clustering process. Also, the sensitiveness against noise and outliers is important here, because critical points have often been identified as noise and consequently been ignored.

- The comparison between the user guided visualization techniques and our automated feature extraction based on clustering is double-edged. In general visual feature extraction is much faster, but the user needs to run it several times to

improve the parameters iteratively. Furthermore, the construction of the visualization pipeline is heavily user biased by the choice of the technique and the parameter setting. In contrast, feature extraction by clustering is slower. It is user independent, but can still be adapted to personal preferences a posteriori, which will be shown in the next chapter.

- Including more properties of the features results in a better segmentation quality, which justifies the increased runtime necessary for the computation of those properties. This novel comprehensive comparison cannot be performed by the classic visualization technique. These techniques are limited in their abilities by the human visual system. The human eye can only resolve three-dimensions. Furthermore, it can qualitatively depicts differences but not quantitative judging them.

Chapter 4

Graph Construction

In the previous chapter, discrete vector fields were segmented into their features, such that each segment consists of coherent cells exposing similar properties and behavior. While the static properties are well represented by the segmentation, the dynamic behavior is not resolved, yet. Thus, the obtained clustering stands for a static snapshot of the discrete vector field and does not convey any information on the relation between the features. However, this relation between the features is essential to understand the dynamic behavior and the flow properties of vector fields.

Each cells of the vector field's grid is labeled by its properties, e.g., its position and size, and the behavior of the flow over this cell. Both information have already been included in the clustering, but only the static properties are described by the segmentation. Therefore, the representation must be extended to include the dynamic flow properties as well as the static cell properties.

In the raw data, every data point is labeled by an attached velocity vector indicating the flow at this point. Like every vector, the velocity vector is also specified by its origin, direction and magnitude. The velocity vector can be imagined as a glyph, which is attached to the corresponding feature. Those glyphs point away from the feature and towards others. Thus, we intuitively obtain a graph representation for the vector field.

The graph construction fulfills two major functions and extracts valuable information from the existing data. First, the graph sets the extracted features in context of each other and indicates their relation, which conveys the vector field's dynamics. As a result, a comprehensive visualization is obtained. In contrast to the established techniques, it is solely based on the vector field data and preserves this information throughout the workflow. Second, the graph representation links two different research domains, visualization and graph theory. Visualization deals with the qualita-

tive analysis and representation of complex and high-dimensional data sets to extract trends and structures and highlights them for the user. The research field of graph theory provides a variety of tools and techniques to analyze graphs quantitatively on a very abstract level. The link between those two domains is established by the presented graph representation of vector fields. Furthermore, it enables the transfer of techniques from one domain to the other. In particular, the former qualitative visual analysis of vector fields can now be enriched by the additional information, which is gained during the feature extraction and graph construction. The results are quantitative statements on the development of vector fields, which are represented by its graph.

If the pair of nodes is ordered, the graph becomes a directed graph, which complies with our initially derived concept for the representing graph. In fact, the representation of a vector field by graph-like structure is so natural, that it is utilized by visualization specialists already. The topological graph [68] connects the extracted critical points with path lines. The obtained graph-like structure is formed by visualization entities only and all numeric properties of the input vector field are stripped away with the extraction of the critical points. Indeed, the resulting graph is directed, but misses any additional information on the features and their relation to each other.

Theisel et al. [132] add a more precise feature description to the topological graph. They distinguish between more different features and consequently utilize more visual entities for the features categorization. The constructed topological skeleton is a highly sophisticated visualization, built upon a large set of visual entities, e.g., saddles and connectors. Those entities are, especially for non visualization specialists, difficult to understand and interpret. Furthermore, the risk of occlusion is increased by the additional visualization entities bearing the danger of overseeing important structures. The Figure 2.7 in the background section shows a topological skeleton built upon critical points, LIC planes and saddle connectors. All those visualization entities have different characteristics and properties, which inhibits a uniform description for all involved objects. Although the topological skeleton is an improvement over the very basic topological graphs, they still miss a uniform and comparable description of the extracted features.

The previous segmentation splits the vector field into parts, but it groups data points of similar properties and behavior together as well. Thus, the features can be used to approximate the vector field on a more abstract level with reduced complexity. To complete the description, the properties of the data points inside the feature can be passed on to it. Thus, the vector field is simplified from several thousand

data points to just a couple of extracted features, which are still preserving the main characteristics of the input vector field. In contrast to traditional visualization techniques that perform a similar reduction complexity without preserving the exact numeric properties.

Instead of complicating the representation further by more specific visualization entities, we utilize the concept of a graph-like representation for vector fields, but it is re-built from its basis upon a different basic feature definition (Definition 2.6). The features used in this work are defined for application to many scenarios and are described directly by the properties of the data points and cells forming them. The properties can only be carried over because the features are defined as coherent regions of interest of similar cells. As for every cell of the vector field the cell's feature vector (Definition 3.1) precisely specifies the cell's properties and behavior, all cells are describe identically. Therefore, the cell's feature vector of the corresponding cells can be joined to describe the complete feature. As a result, a uniform feature description is derived. Because the automated feature extraction preserves the properties, the graph can be enriched by those labels to obtain a labeled geometric graph.

Definition 4.1. *Geometric Graph*
The labeled directed geometric graph $G = (V, E, L)$ consists of a finite set of nodes V, a finite set of edges E connecting the nodes pairwise and a set of labels L. L contains labels for every node $n \in V$ including the node's spatial coordinates and every edge $e \in E$.

This basically defines the standard graph. It has been already mentioned in the background chapter, but because of its importance in this chapter, we repeat it here once more. The set of labels L is added to describe the properties of the nodes and the behavior of the edges. Because in this context the geometric graph represents the vector field's flow, we will refer to them as *flow graphs* or *geometric flow graphs*. In contrast to the visual approaches, our approach simplifies the representation to an abstract flow graph, which reduces the risk of occlusion to a minimum. Additionally, graphs are more intuitive and easier to understand for the viewer.

Thus, the flow graph displays the vector field and the derived data transparently and clearly. However, the major advantage of the flow graph representation is merely a nicer visualization, but the key for a quantitative analysis of vector fields. The uniform feature description, which is introduced along with the flow graphs, enables such an analysis. Furthermore, in the next chapter, we will show, that the introduced flow graphs link vector field analysis with graph theory. This newly established link

allows the transfer of powerful techniques to improve the vector field analysis. In the following, the sets of the geometric graph are extraction or constructed, respectively. First, the nodes, that are basically the features of the discrete vector field. However, the optimal number of them must be defined, which is equivalent to find the optimal cutting level *ocl* for the feature's hierarchy. In the second step, the edges connecting the nodes are constructed describing the dynamic behavior of the vector field. Since the glyphs are not connecting the features and just point in some direction, the construction of the edges between the features requires a more detailed examination. After the graph's nodes and edges are defined the labels are added to obtain a complete geometric graph representation. The components of the cell's feature vector are passed on to the feature to describe it. This description is based directly on the properties that have been derived during the feature extraction.

Because of the new definition of features alongside their changed description, no comparable algorithms for the graph construction process exist. Therefore, the functionality of this process is shown in the experimental evaluation by a proof of concept. In the experimental evaluation, the advantages of the graph representation over traditional visualization techniques are demonstrated.

4.1 Nodes

The segmentation computed by hierarchical clustering algorithms is represented by a hierarchical structure, the so-called dendrogram, in which every level stands for a specific segmentation into a distinct number of clusters or features. In the highest level of the hierarchy, the entire vector field is represented by one complete segment. In the next lower level this segment is split in two, such that every cell of the feature is more similar to cells inside this segment than to cells outside it. The splitting is iteratively repeated until every cell forms a feature of its own on the lowest level of hierarchy.

Definition 4.2. *Feature Hierarchy*
Discrete vector fields are decomposed by hierarchical clustering into a hierarchy of features $C(n)$. For every level h the hierarchy consists of individual features $C(h) = \{c_{h,1}, c_{c,2}, ..., c_{h,h}\}$. The highest level $C(1)$ consists of exactly one feature $c_{1,1}$ and the lowest level $C(n)$ is formed by the n individual cells of the discrete vector field.

The feature hierarchy $C(h)$ is a multilevel resolution of the vector field. By cutting the hierarchy at a certain level, the level-of-detail for the representation is selected.

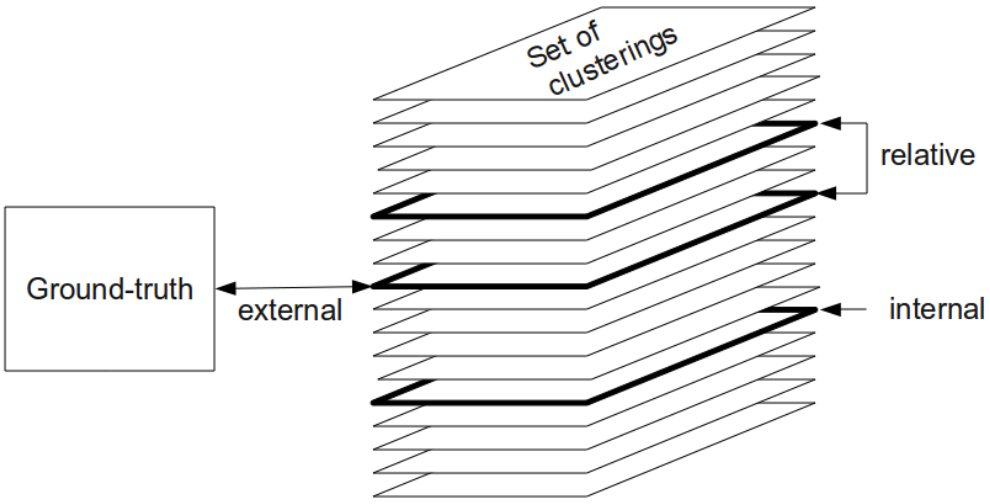


Figure 4.1: Schematic overview of different classes of cluster evaluation.

For every level of the hierarchy, the balance between abstraction and resolution shifts. For instance in the lowest level, the abstraction is the lowest one as well, because every cell forms an individual feature, which results in the highest resolution. And in the highest level of the hierarchy, the ratio between abstraction and resolution is inverted. The optimal cutting level of the hierarchy is found at the level in which both of these aspects are balanced and consequently the optimal vector field representation is obtained.

Definition 4.3. *Optimal Cutting Level*
The feature hierarchy $C(n)$ is formed by n levels. The hierarchy's level that represents the input vector field the best is defined as:

$$ocl := \max_{h}(\textit{quality of the clustering});\ h \in [1, ..., n] \tag{4.1}$$

The quality of the clustering estimates the balance between simplification, which reduces the number of segments, and details, which are available for more segments.

The optimal clustering level *ocl* is the one for which the clustering quality is the best. However, validating the clustering's quality is rather difficult and can only be answered a posteriori. Basically, there are three categories of cluster validation techniques [62], which are shown in Figure 4.1 as well. External criteria compare the obtained clustering with an expected structure. In general, the necessary ground-truth information is unknown in this specific scenario. Internal criteria validate the clustering only on the extracted structures and do not compare them to expected

structures or other clusterings. Finally, relative criteria compare the structures of two given clusterings and judge which of them results in a better representation of the input data. Only the last category is directly applicable to this scenario to find the optimal cutting level for the cluster hierarchy.

Traditionally, two characteristic values express the quality of clustering as well as the compactness and separation of the found clusters. The compactness evaluates the redistribution of the input data by the clustering algorithm into the computed clusters [66]

$$compactness := \frac{1}{n} \sum_{i}^{n} \frac{v(c_i)}{v(X)} \tag{4.2}$$

where n is the number of computed clusters for the data set X and the variance $v(\cdot)$. Smaller values indicate a higher average compactness of the computed clustering. Indeed, a higher average compactness is not equivalent to a better quality of the clustering. The separation needs to be considered, too. It expresses the similarity of the computed clusters [66]

$$separation := \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j=1, i \neq j}^{n} exp(-\frac{d^2(x_{c_i}, x_{c_j})}{2\sigma^2}) \tag{4.3}$$

with σ as a Gaussian constant, $d(\cdot)$ as the distance used to compute the clustering and $d(x_{c_i}, x_{c_j})$ as the distance between the centroids of the clusters c_i, c_j. The smaller the separation, the larger are the differences between the clusters. Because compactness and separation are opposing values, the best clustering quality is achieved for a clustering in which both aspects are balanced, e.g., for the weight $\beta \in [0; 1]$ [66]

$$quality\ of\ the\ clustering = \beta \cdot compactness + (1 - \beta) \cdot separation \tag{4.4}$$

The Dunn's index [45] approximates the compactness and separation of the clusters $\{1, ..., k\}$ by finding the set distance $\delta_{p,q}$ between clusters $p, q, \in \{1, ..., k\}, p, \neq q$, s.t.

$$DI = \min_{p,q}\left\{\frac{\delta_{p,q}}{diam}\right\} \tag{4.5}$$

$$\delta p,q = \min_{x\in p}\left\{\min_{y\in q}\|x-y\|\right\} \tag{4.6}$$

$$diam = \max_{l}\{diam_l\} \tag{4.7}$$

with the diameter $diam_l$ of the l^{th} cluster, which is defined by the maximum distance between two items within the cluster. A larger Dunn's index DI indicates a better clustering. Variants of the Dunn's index can be constructed by altering the definition of the set distance $\delta_{p,q}$ and the $diam_l$ [19]. The well-known Silhouette Criterion [114] examines the compactness and separation of clusters, too. For every item belonging to any cluster $x \in p$, the silhouette is computed by:

$$s_x = \frac{\|z-x\| - \|y-x\|}{\max\{\|z-x\|, \|y-x\|\}} \tag{4.8}$$

for $x, y \in p$ and $z \in q \neq p$, i.e., the object x is compared to all other objects y of cluster p and all objects z outside cluster p. For every data point $x \in N$, the silhouette is computed as described in Equation 4.8 and the arithmetic mean gives the Silhouette Criterion

$$SCC = \frac{1}{N}\sum_{x} s_x. \tag{4.9}$$

Larger SCC coefficients indicate a better clustering quality. Variants of the coefficient can be constructed by changing the Equation 4.8 [73] and applying a different distance function.

Langfelder et al. [88] present an algorithm which extracts the optimal cutting level for a given dendrogram from a hierarchical clustering. The clusters are decomposed and recombined iteratively until the number of clusters becomes stable. This process is initialized with few large clusters computed by the static tree cut. For each cluster, the joining heights in the dendrogram are analyzed. If the clusters exhibits a fluctuating, characteristic pattern, they are split again, because a sub-cluster structure is assumed. Very small clusters are initially merged with larger, neighboring ones, which prevents an over-splitting. Furthermore, Langfelder et al. [88] proposed an improvement to the algorithm by adding additional information on the similarity of the data points within the clusters, which basically combines two strategies to judge the clustering quality.

At the beginning of the section a computed dendrogram is required, that had to be generated by the previous clustering during the feature extraction. Although the best clustering quality was achieved by the application of hierarchical clustering in our specific context, the complete workflow is constructed to have an interchangeable clustering algorithm. This modification might be utilized, if the user prefers a different algorithm. Furthermore, segmenting the data set by hierarchical clustering is usually more costly than other clustering techniques, because it generates a complete hierarchy rather than a single cluster. In the case that hierarchical clustering, in our particular setting Birch, is replaced by another clustering technique, which does not build a dendrogram, such must be computed afterwards. This subsequent computation is cheaper, because the hierarchical clustering is computed for the initially extracted clusters to obtain the dendrogram. This is not only necessary for choosing the level of representation a posteriori, but also in order to construct the edges connecting the nodes efficiently.

4.2 Edges

Although, the conceptual idea of just connecting the extracted features by their glyphs sounds straight forward, many problems and challenges are involved in the edge spanning procedure. First, the flow direction of a feature needs to be defined. The data points with the corresponding velocity vectors are similar, but not identical, i.e., the velocity vector of the feature's data points may point in different directions. Second, the ambiguity of the target needs to be solved. If the feature's velocity vector is pointing not exactly to one other feature, which is the most likely case, the target of the edge is not unique. In two dimensions, this problem can be solved reasonably by vector analysis, but for three dimensions, a different technique is necessary in order to avoid considering several spatial angles.

The second most obvious construction starts at a randomly chosen cell and integrates the flow to build path lines. This idea is borrowed from the construction of topological graphs and skeletons, respectively. For every path line, the cells forming it need to be checked, if they are member of a different cluster than the previous cell. In the positive case, both clusters are connected by an edge. This search needs to be processed until every cell is either a starting point or member of a path line, which is not efficient and only works in the best case scenario. In the most likely case, a cell cannot be clearly identified as a member of the path line, because the flow direction is not aligned with the grid's geometry. Therefore, two or more cells are candidates

for the path line, which makes the propagation of the line ambiguous and dependent on the starting points.

Other geometric approaches face similar problems and therefore, we evolve from a geometric understanding of the edge construction process towards a more abstract one. Basically, an edge between two nodes indicates a transportation process between both, and this process can occur at adjacent surfaces only. This physical interpretation transfers the edge construction from a geometric problem to finding adjacent surfaces as their are present between neighboring cells. Querying for neighboring cells was used already for the computation of divergence and rotation in the feature extraction and can be re-utilized here.

The brute force approach strategy I just tests if any two nodes are connected. For a given hierarchy level $C(h)$ each pair of features (c_i, c_j); $c_i, c_j \in C(h)$ needs to be compared, to determine whether c_i and c_j share common points. The comparison must be performed on point basis, as by definition 2.3, one individual cell cannot be element of two different clusters. Therefore, if, and only if, two features are connected with each other they share at least one common point, which is a corner point of both cells. Indeed, cells can also share more than one point, as seen in Figure 2.1.

Algorithm 2: Check if two features are neighboring each other

Data: feature A, feature B
Result: number of common points
1 PA = all points forming cells of A;
2 PB = all points forming cells of B;
3 compute the intersection of PA and PB;
4 return result;

If the function 4.2 returns a positive result, the input clusters are connected and an edge between the nodes is drawn. Computing all edges for the clustering $C(h)$ consisting of h clusters requires:

$$h + (h-1) + \cdots + 2 + 1 = \sum_{c=1}^{h} c = \frac{h(h+1)}{2} = \frac{h^2 + l}{2} \tag{4.10}$$

checks, which is the Gaussian Summation equation. By this brute force algorithm, good results are obtained, i.e., the edges are correct and no false edges are found. The runtime can be shortened by including information on the feature hierarchy, which is already available as it has been computed during the clustering process.

Strategy II minimizes the number of checks between the h features of $C(h)$. Instead of pairwise checking the connectivity of the nodes, the clusters are compared by their similarity. Iteratively, the two most similar clusters are merged until, after h steps, only one large cluster representing the whole vector field is left. Essentially, this equals a second clustering step, exploiting the dendrogram of the first hierarchical clustering, respectively.

The dendrogram is either parsed top-down or bottom-up, which makes no difference here and without loss of generality we describe the process bottom-up. The two most similar clusters are merged and regarded as one in the next iteration and an edge between both of them is stored. This step is repeated h-times and, if a cluster is joined to two already merged ones, the edges between all three are stored.

Algorithm 3: Connecting nodes by dendrogram

```
Data: dendrogram
Result: Adjacency matrix
while There is more than one cluster do
    merge two clusters according to the dendrogram;
    if both are cluster then
        change their entries in the adjacency matrix;
    else
        look-up the edges of the compound cluster;
        add edges from the newly merged cluster to all members of compound;
return adjacency matrix;
```

Again, for h features $\frac{h^2+h}{2}$ comparisons are required. These checks are cheaper and are computed faster than the search for common points, because they are solely performed on the dendrogram. Unfortunately, this strategy produces too many false edges, because it connects every feature with every other one. Therefore, both strategies are combined to achieve a correct result in the shortest possible time. The resulting strategy III is a hybrid version of both I and II. In strategy II, a connectivity check, as used in strategy I, is added to reduce the number of checks compared to strategy I.

By adding the information of the dendrogram, the number of checks is reduced in dependence of the connectivity among the clusters. In the worst case scenario, if the nodes are fully connected, $\frac{h^2+h}{2}$ are still needed. However, for linear connected nodes, the number of checks is reduced to h.

Algorithm 4: Connecting nodes by dendrogram

```
Data: dendrogram
Result: Adjacency matrix
while There is more than one cluster do
    merge two clusters according to the dendrogram;
    if both are cluster then
        change their entries in the adjacency matrix;
    else
        look-up the edges of the compound cluster;
        check, if every cluster of the compound cluster is connected to newly
        merged cluster;
        change entries of adjacency matrix accordingly;
return adjacency matrix;
```

4.3 Graph Labeling

So far, the extraction of the nodes and the spanning of the edges was described. At this point, the representing graph is as powerful as the topological graph or the topological skeleton, respectively. In the following, labels are added to the graph entities enriching the graph representation with a precise and uniform description of nodes and edges. Because the topology of the representing graph is not sufficient to describe it in detail, additional information is derived from the input vector field to label the graph's nodes and edges. We distinguish between structural properties and flow properties. Structural properties describe the shape and the appearance of the represented feature, while the flow properties give details on the relation between these features. Therefore, the structural properties are attached to the nodes and the flow properties to the edges that represent the interaction between two nodes.

4.3.1 Structural Properties

The structural properties describe the static appearance of the vector field and are derived from the cells forming the corresponding feature directly. Vector fields from different simulations may be represented on different scales and in different reference systems. In order to achieve comparability between two representing graphs comparable, we derive properties, which are invariant to such transformations.

Center

Every feature $c_{i,h} \in C(h)$ is represented by an individual node $n_i \in \mathcal{N}$. The features have a spatial volume, which is shrunk to a point represented by the node. Therefore, the feature's cells are projected to the center of mass, which can be interpreted as the feature's center. Hence the cells $\mathcal{C}_v$ inside the feature are equally weighted, the center of mass is computed by the arithmetic mean of the cell's coordinates, i.e.,

$$\vec{cm}_{n,i} = avg(\mathcal{C}_v),\ \forall\ \mathcal{C}_v \in c_{i,h} \in C(h) \tag{4.11}$$

The obtained vector $\vec{cm}_{n,i} \in \mathbb{R}^n$ defines the position of every cell in space. This property is not invariant to any transformation of the given vector field, i.e., features of two identical simulations, which are represented in different reference frames, have different centers. This property is used to visualize the graphs, but it also extends the representing graph to a geometrical flow graph. Therefore, the property can also be used to compare two graphs with identical reference frames, which is the case for time steps within a single simulation.

Volume

The clustering process is based on an initially defined feature vector for every cell (Definition 3.1), on which the grouping is performed. We included the volume of every cell in the cell's feature vector already. Therefore, the volumes of every individual cell $\mathcal{C}_v$ of the feature can be summed up easily to obtain the volume of the feature, i.e.,

$$V(c_{i,h}) = \sum_{\mathcal{C}_v \in c_{i,h}} V_{\mathcal{C}_v},\ \forall\ \mathcal{C}_v \in c_{i,h} \in C(h) \tag{4.12}$$

Since this property depends directly on the size of the cells, it is not invariant to scaling, but it is still rotation and translation invariant. By dividing the feature's volume by the overall vector field volume, the property becomes also invariant to transformation, because the obtained ratio reflects the importance of the feature inside the vector field with respect to its size.

Feature's Average Flow

This property is considered as structural, because it only belongs to a single node and does not describe any interaction between different nodes. Similar to the center

of the feature, the average flow of the node is computed. The arithmetic mean over all velocity vectors of every element inside the feature is computed.

$$\gamma(c\vec{m}_{n,i}) = \sum_{\mathcal{C}_v \in c_{i,h}} \gamma(\mathcal{C}_v),\ \forall\ \mathcal{C}_v \in c_{i,h} \in C(h) \tag{4.13}$$

The major flow of the node is distinguished by this properties and depends exclusively on the simulation itself being invariant to any transformation.

Divergence and Rotation

Both values are of interest to categorize the feature's type. Helman's et al. feature definition [68], which is shared among all topological extraction methods in visualization, is based on the divergence and the rotation value for every point inside the vector field. By the divergence, sinks and sources can be identified inside the vector field and determined whether a feature is influenced by one of those structures. Similarly, the rotation identifies if the feature is influenced by a curl and its direction of the rotation. Figure 2.2 gives details on their feature definition.

At this point, the construction of the vector field over a set of cells becomes important (Definition 2.3). Different to Helman et al. [68], divergence and rotation are not computed by partial derivatives, but by integrals over the cells. Partial derivatives would require recomputing on every level of hierarchy. By using an integration process, the divergence value for every cell inside the corresponding feature can be summed up in order to obtain the divergence value for the complete feature and respectively for the rotation, respectively.

During the integration process, the surface integral for every face of the cell is computed. Doing this, the cell's corner points must be traversed in a distinct order, mathematically positive with respect to the face's normal. Two adjacent cells share a common surface, but for the two cells the surface's normals point in opposite directions, and therefore the integral values for this surface sum up to zero. See Figure 3.3 for the integration process over two adjacent cells.

$$div(c_{i,h}) = \sum_{\mathcal{C}_v \in c_{i,h}} div(\mathcal{C}_v),\ \forall\ \mathcal{C}_v \in c_{i,h} \in C(h) \tag{4.14}$$

$$rot(c_{i,h}) = \sum_{\mathcal{C}_v \in c_{i,h}} rot(\mathcal{C}_v),\ \forall\ \mathcal{C}_v \in c_{i,h} \in C(h) \tag{4.15}$$

$div(\cdot)$, $rot(\cdot)$ is the divergence or rotation, value for every cell and the feature. Both values are invariant to any transformation applied on the input vector field.

4.3.2 Flow Properties

So far, the derived properties describe the appearance of the individual feature, but no information about the interaction between two features is conveyed. This section focuses on this relation, which are represented by a node each, and therefore the following properties label the edges connecting the nodes and can be interpreted as the dynamic properties of the vector field. The strategies mentioned previously only compute undirected graphs, in which the connection between the nodes is not labeled with a direction. Therefore, the first property describing the edge is its direction. Subsequently, we will multiply it with a weighting which depends on the flow magnitude over the border cells to indicate the strength of the flow.

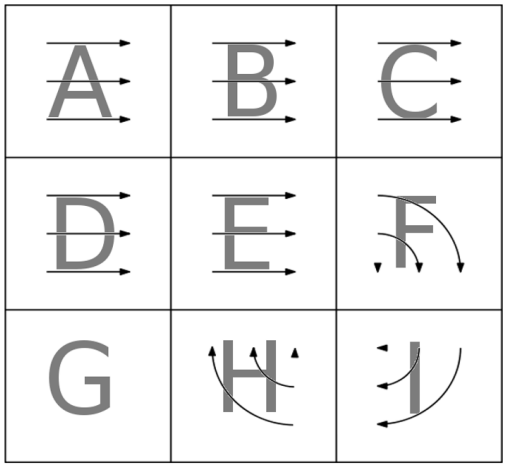
Unfortunately, the flow direction does not have to be parallel with the direct connection between the cluster centers, and also not orthogonal to the cluster borders. Therefore, the cosine-similarity is applied to judge the flow's direction.

$$\cos(\theta) = \frac{(c\vec{m}_i - c\vec{m}_j) \cdot \gamma(0.5 \cdot (c\vec{m}_i - c\vec{m}_j))}{\|(c\vec{m}_i - c\vec{m}_j)\| \|\gamma(0.5 \cdot (c\vec{m}_i - c\vec{m}_j))\|} \tag{4.16}$$

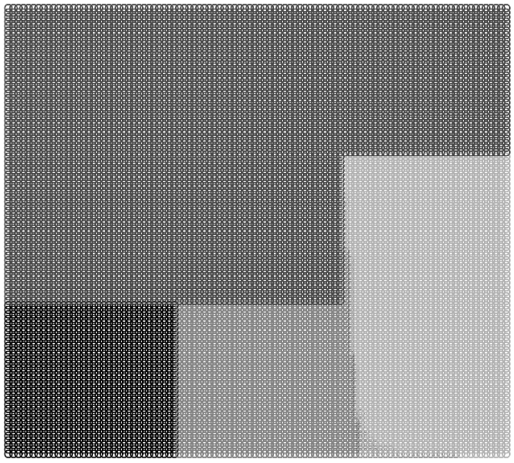
For $\theta < 90$ the edge is pointing from edge $c\vec{m}_i$ to $c\vec{m}_j$, and for $\theta > 90$ reversed. For $\theta = 90$ the cosine-similarity vanishes, because the flow is orthogonal to the line $(c\vec{m}_i - c\vec{m}_j)$ and no edge is drawn between both nodes. The cosine-similarity computes only the angle spanned by the two vectors. To obtain a meaningful numeric value, the magnitude of the flow $\gamma(0.5 \cdot (c\vec{m}_i - c\vec{m}_j))$ and the number of cells forming the border between the clusters $c\vec{m}_i$ and $c\vec{m}_j$ are multiplied to the computed cosine similarity.

4.4 Experimental Evaluation

The evaluation focuses on two aspects. First, a proof of concept demonstrates the functionality of the proposed technique. It is performed on the fully synthetic backward step, because ground-truth information is available for this data set. The same information is also used to validate the feature properties, which are derived from the data points. In the second part, the algorithm is applied on real simulations to demonstrate the usability and to enrich their visual representation. In this part, we



(a) Simulation setup and ground-truth information. The velocity vectors are symbolized by glyphs for nine different regions (A-I). There is no flow in region G and therefore, no glyphs are drawn.



(b) Features extracted by BIRCH. The four clusters correlate with the parts specified by the setup.

Figure 4.2: Generic backward step data set used for the edge validation. The images shows the projection of the three-dimensional data set onto the $x - y-$plane.

also discuss the number of extracted features, which are forming the nodes of the flow graph.

4.4.1 Proof of Concept

The focus for the proof of concept lies on the edges and their correctness. During the development of the algorithm, strategy I was already discarded, for its of the obviously incorrect edges. Now, the correctness of the chosen algorithm is demonstrated on a synthetic data set with ground-truth information. We use the generic backward step, which has been discussed previously for the feature extraction. Figure 4.2 shows the setup once more and the extracted features for which the edges are constructed.

In general, an edge represents the transport process between two features over a common surface. Indeed, a common surface does not guarantee a transportation process between the corresponding features. Several different scenarios must be considered for two adjacent feature $c_{i,h}, c_{j,h}$ and the flow over the surface γ_s:

a) There is no flow over the surface, i.e., $|\gamma_s| = 0$. Consequently, no edge between $c_{i,h}$ and $c_{j,h}$ is drawn.

b) The flow is parallel to the surface, i.e., the dot product $\gamma_s \cdot \overrightarrow{c_{i,h}c_{j,h}} = 0$ vanishes, because the flow is orthogonal to the connecting vector.

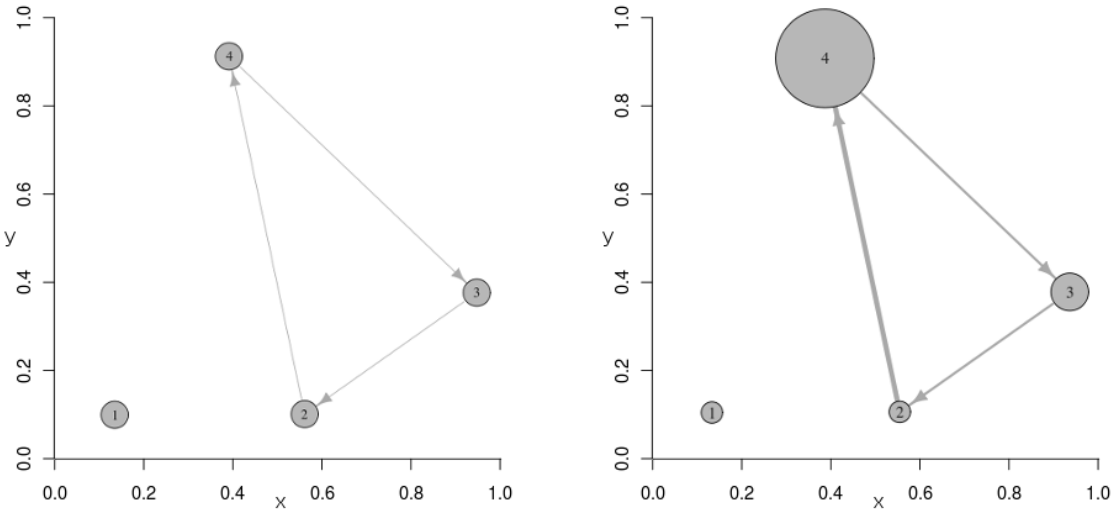
c) The flow is clearly directed from one feature to the other, i.e., the dot product is not zero and the velocity vectors are continuous at the surface.

d) The flow is not clearly directed and the velocity vector is not continuous, i.e., the absolute value of the velocity vector $\gamma_{c_{i,h}}$ and $\gamma_{c_{j,h}}$, which are computed for the sides of the surface of thickness δ, is $|\gamma_{c_{i,h}} - \gamma_{c_{j,h}}| > \epsilon$

Those four cases must be considered to validate all possible scenarios. In Figure 4.2(a) the velocity vectors are indicated by arrows, at the border surfaces the flow is continuous, which is not visible in the figure. The generic data set is constructed to be continuous and therefore the border regions of the different segments are manufactured to have a continuous translation rather than shocks. The situation of a) is found in the generic backward step between features G and H, where no flow streams over the border surface. Scenario b) is found between C and F, where the flow is aligned with the border surface. Case c) is actually the most common and standard and occurs at all other border surfaces. Finally, case d) is very special and not covered in the generic backward step. However, this scenario will not occur in any other CFD simulation. The simulations are constructed to be continuous, which explicitly excludes scenario d).

For the extracted features of Figure 4.2(b) the corresponding flow graph is constructed by strategy III. Because the results shown in Figure 4.3 are constructed basing on the properties of the features, we provide coordinate axis. Note that the graph is actually three dimensional, but the results are projected onto the $x-y-$plane. The simple graphs use just the coordinates of the features to place the nodes in space, while the enriched flow graph includes additional properties to improve the visualization as well. The nodes in the enriched flow graph are scaled by the relative size of the corresponding features which they are representing, i.e., the size is specified by the ratio of the individual feature volume over the total volume of the vector field. This means for the flow graph of Figure 4.3(b), that nodes 1 and 2 are scaled identically, while node 3 has the double size of them and node 4 is even larger than the other nodes combined. The width of the edges is computed similarly, the individual flow along the edge is divided by the complete flow inside the graph to obtain the ratio.

To validate the edges, the scenarios a) to c) are now discussed case-by-case. Scenario a) is fulfilled, because no edge is drawn between the nodes 1 and 2. The case c) is handled correctly and an edge $(3 \rightarrow 2)$ connecting nodes 2 and 3 is computed. The last remaining scenario b) is not handled correctly as obviously as the other two scenarios. Basically, the edge $(4 \rightarrow 3)$ must be drawn in, because there is a flow from



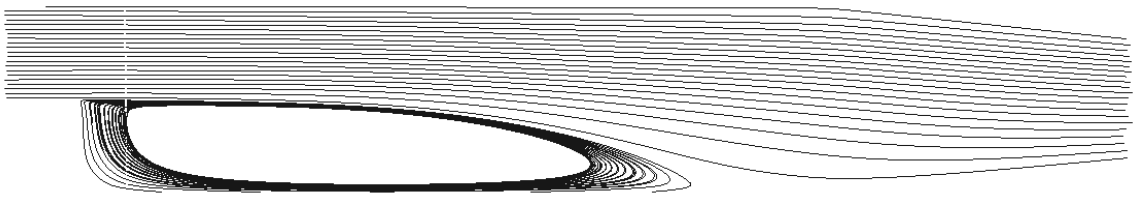
(a) Simple flow graph without weighting of nodes or edges.

(b) Enriched flow graph with weighting of nodes and edges. The sizes of the nodes are scaled corresponding to the volume of the vector field they are covering. The edges are weighted by their impact on the overall flow.

Figure 4.3: Flow graphs for the generic backward step. The graphs have been constructed for the extracted features shown in Figure 4.2(b).

region E, which is part of feature 4 represented by node 4, and region F, which is part of feature 3 represented by node 3. Only the properties of this particular edge can prove its correctness. The widths of edges $(3 \rightarrow 2)$ and $(4 \rightarrow 3)$ are identical, i.e., their properties are the same and the transport process between nodes 3 and 2 is identical to the transport process between nodes 4 and 3. Suppose the case that there is a flow between region C and F, the edge $(4 \rightarrow 3)$ would not be identical to $(3 \rightarrow 2)$. And so, by contradiction, we proved, that the transportation process causing the edge $(4 \rightarrow 3)$ occurs at the surface between regions E and F and not C and F, which validates scenario c). This argumentation is not applicable to the edge $(2 \rightarrow 4)$, which is consequently wider than the other two edges. This is caused by the different flows at the border surface between region E and H. This argumentation justifies to scale the edges as we did. If the scaling did not include the sum of all flows, the width of the edges would be misleading. For instance, if we only consider the out-flow of one node which has only one outgoing edge. This edge would be scaled as 1, because 100% of the flow stream out of the node over this edges. This way, the edges are not comparable for their impact inside the flow graph.

(a) Heat-map of the velocity vectors' magnitudes. Brighter colors represent larger magnitudes.



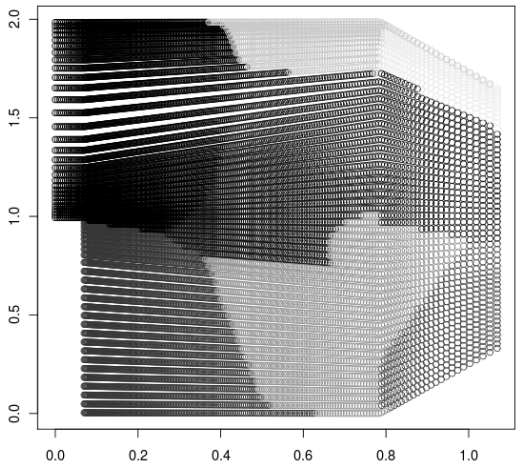
(b) Path line visualization for the backward-step. The path lines are seeded at the entrance of the tube.

Figure 4.4: Backward step data set. The flow streams in the tube from the left and faces the backward facing step. The tube is conical to minimize unwanted interactions. The images show clipping planes of the three-dimensional test chamber.

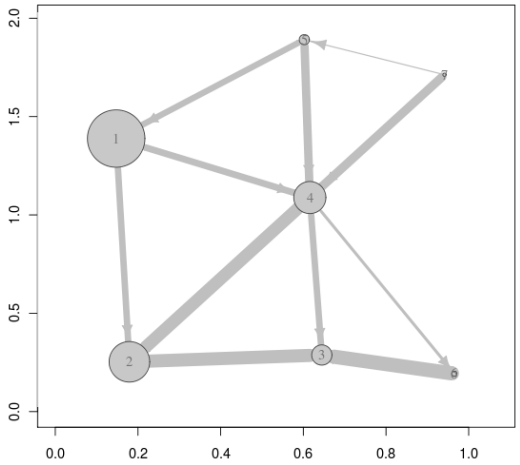
4.4.2 Application

So far, the correctness of the algorithm was proven on a generic data set only. To demonstrate that the algorithm is useful and effective, real simulations are more expressive. First, the algorithm is applied to a simulation of the backward step to show that the generic data set examined previously is not totally artificial. Subsequently, the flow graph for the lid-driven cavity data set is constructed. On this data set, the number of the features to be extracted is discussed, because it is more likely to be familiar to the reader than the backward step scenario.

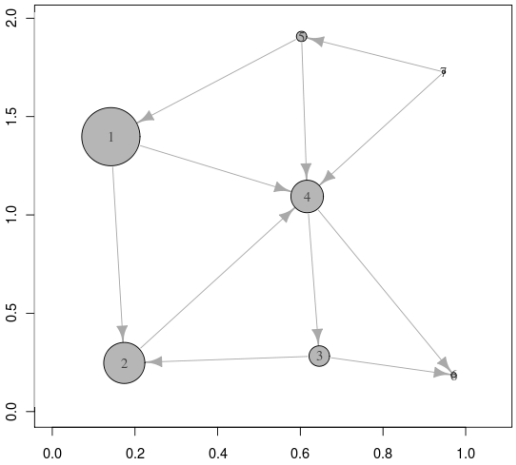
The setup for the generic backward simulation (Figure 4.2(a)) and its real counterpart are almost identical. The one in Figure 4.4 differs in its dimensions and the conical end section of the tube only. This section guarantees the correct and continuous flow in the front part without any unwanted interactions between the flow and the virtual walls of the tube. The flow streams in from the left and faces the backward step. Depending on the fluid's viscosity, a stationary curl forms here while the upper part of the flow streams to the end section on the right. For the visualization of the velocity vectors, a heat-map was chosen in Figure 4.4(a) in which white indicates the larger magnitudes of the vectors and black the lower ones. This techniques allows an overview of the complete vector field, while the path lines in Figure 4.4(b) provide more details in particular on the formed curl at the side of the backward facing step.



(a) Features extracted by BIRCH.



(b) Enriched flow graph with weighting of nodes and edges. The sizes of the nodes are scaled according to the volume of the vector field, which they are covering. The edges are weighted by their impact on the overall flow.



(c) Partly enriched flow graph with weighting of the nodes. Their sizes are scaled according to the volume of the vector field they are covering.

Figure 4.5: Flow graphs for backward step data set constructed according to the feature extraction by BIRCH.

For the backward step, seven features were extracted. This number of extracted features was chosen to simplify the comparison between the backward step and its generic counter-part. Despite the similar choice of the number of features the results for both data sets look quite differently. The part of the homogeneous flow, which is formed by the upper regions of the generic backward step (regions A-E in Figure 4.2(a)), is split into five individual features (feature 1, 4, 5, 6 and 7) for the backward step (Figure 4.5(b) and Figure 4.5(c)). We provide both, the simple and the enriched flow graph, to give the reader a complete picture of the properties as well as the structure of the flow. The Figures 4.5 were scaled to be the same sizes as the generic backward step for a better comparison. The different appearance is caused by the over-simplification of the generic data set, in which the regions A-E are identical. In contrast to that, the data points represented by the feature 1, 4, 5, 6, and 7 of the non-generic scenario are just similar, but not identical. This is caused by the more complex mathematical model simulation in comparison to the very simple setting of the generic case. Actually, the latter is not based on any mathematical model it has been constructed, such that it mimics the simulation in its appearance. The simulation includes the friction of the walls, though, the test chamber's bend, which causes a higher pressure in the end section, and many more influences, which are responsible for the non-identical properties of those regions. Nevertheless, the discussed regions are similar in their position and properties, such that the generic data set can be seen as an approximation of the simulation, which is not oversimplified. Upon examination of the curl formed in both simulations this becomes clearer. This curl is formed by the features 2, 3 and 4 in the simulation, compare Figure 4.5(b) and Figure 4.5(c). The same structure is also found in the generic case, which was discussed previously as well. Overall, the generic case really can be seen as a valid simplification of the simulation. This strengthens the proof of concept and demonstrates the correctness of the approach.

The flow graph is constructed correctly although some of its properties seem to be wrong on first sight. For instance, we look at the edges $(7, 5)$ and $(5, 1)$, which are directed opposing the general flow direction. This is caused by the geometry of the border between the corresponding features. For the features 5 and 7, the border surface is concave. While there is a flow from feature 5 to 7 at $x = 0.8$, the major flow is exchanged at the border surface, which is parallel to the test chamber's wall. Here the velocity vectors of the data points inside feature 7 are pointing towards the wall, but to this wall, a volume belonging to feature 5 is adhered. Therefore, the edge pointing from feature 7 to 5 is actually correct. The same situation appears

between the nodes 5 and 1, where the geometry of the border surface causes an edge in the opposing direction. This issue could be solved by a better placing of the nodes representing the feature inside the flow graph. At the moment, the nodes are placed at the coordinates of the center of mass of the corresponding feature. By shifting this center according to the general flow direction, the visualization would become clearer and the previous issue would not appear. Because the efforts of this work are to reduce the amount of visualization in vector field analysis, the mentioned changes are ignored since they are not improving the general results, but only their visualization in one specific case.

After the correctness of the algorithm has been shown, its advantage over the traditional visualization techniques are demonstrated. The applied heat-map in Figure 4.4 indicates the presence of a curl formed by the backward step. Except the magnitudes of the velocity vectors, which are coloring the representation, no quantitative assumptions can be stated for it. In contrast to the visualization, the properties of the curl can easily be estimated by the extracted features and the flow graph. The curl is stimulated by two sources, the features 1 and 3. While feature 1 contributes to only one third of the curl, the major stimulus is caused by feature 3, because this is the edge of the major flow and twice as thick as the edge $(1, 2)$. To estimate the strength of the curl, the border surface between feature 2 and 4 is examined. This is the narrowest expansion of the curl and by measuring this border surface, the curl is approximated along with its flow strength, which is the flow over this short border surface. This quantity can also be estimated at the key node 4, where the flow branches into the curl and the flow streaming the major direction to the right and nodes 6 and 7. In Table 4.1, the properties for the flow graph's features are given. The adjacency matrix with the corresponding percentage of the flow over this edge is shown in Table 4.2. Both tables provide the properties necessary to compute the previous example.

The derivation of Table 4.1 and Table 4.2 was explained in the previous conceptual sections. Deriving those tables based on visualization techniques to compare them with our tables, is not possible, because visualization techniques solely focus on a qualitative analysis. This analysis focus mainly on highlighting significant structures inside the vector field, and not on their exact and quantitative description by a defined set of properties. Furthermore, visualization techniques are limited by the human visual system to two dimensions. For instance, the shown path lines in Figure 4.4 are drawn on a two-dimensional plane, which is parallel to the $x - y-$ plane and is positioned at the mid of the test chamber in $z-$ direction. The flow graph, in

node	x_1	x_2	u_1	u_2	V
1	0.15	1.39	9.39	-0.63	34
2	0.18	0.23	-0.05	0.14	24
3	0.65	0.31	2.17	-0.34	12
4	0.62	0.27	7.43	-0.45	19
5	0.61	1.92	6.2	-0.09	6
6	0.96	0.19	4.1	0.047	3
7	0.96	1.74	5.98	-0.54	2

Table 4.1: Nodes' properties for the flow graph of the backward step (Figure 4.5). The spatial coordinates (x_1, x_2), the velocity vector (u_1, u_2) and the volume (in percent) are given for the individual nodes.

	1	2	3	4	5	6	7
1	0	7.69	0	16.46	0	0	0
2	0	0	0	7.38	0	0	0
3	0	14.35	0	0	0	10.09	0
4	0	0	15.2	0	0	1.25	0
5	7.41	0	0	3.43	0	0	0
6	0	0	0	0	0	0	0
7	0	0	0	7.14	9.6	0	0

Table 4.2: Weighted adjacency matrix for the flow graph of the backward step (Figure 4.5). The weight shown for any edge represents the percentage of the overall flow via this edge.

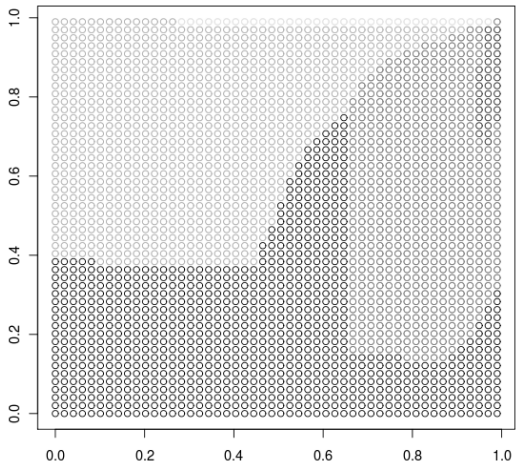
contrast to the visualization, is based on all three dimensions and the properties are computed by considering all dimensions as well. Therefore, the flow graph representation is a comprehensive analysis of vector fields, which includes all properties and dimensions of the raw data. Because the focus of the analysis lies on a comprehensive and general description, the visual representation of the flow graph is less appealing than many visualization techniques. We discussed some issues to improve the visual representation already, but those issues are adapted for one specific scenario. For instance, two virtual nodes could be added at the front and at the back of the test chamber standing for the in- and outflow of the system.

After constructing the flow graphs for the two backward step test cases, the flow graph for the lid-driven cavity case is constructed. It is actually simpler than the simulated backward step in terms of model complexity, but the order of their appearance was changed to have a better transition from the generic backward step to the simulated one. Because the functionality of the concept has already been demonstrated, the focus for the lid-driven cavity data set lies on the correct number of extracted

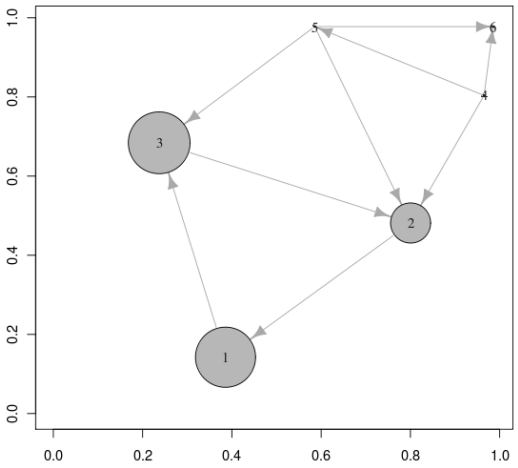
features. Nevertheless, the constructed flow graphs for the lid-driven cavity scenario are discussed first to familiarize the user with the data set and the graphs.

In Figure 4.6, different segmentations of the vector field are shown along with their corresponding flow graphs. We chose the simple flow graphs, because in the following discussion the focus lies on the segmentation and the flow's structure rather than the flow properties. The constructed flow graphs (Figure 4.6(b), 4.6(d), 4.6(f)) look very different, but they actually expose many similarities. First, in the upper right corner the extracted features are the smallest, which is caused by the cells properties changing rapidly, as described during the feature extraction process already. This is caused by the simulation's setup of the moving top lid and the reflection at the fixed wall on the right. Turbulences are formed here causing the chaotic flow. While the flow graph for 27 nodes is too cluttered, in the graphs for 6 and 13 nodes a curl is visible. In Figure 4.6(b) it is formed by nodes $2 \rightarrow 1 \rightarrow 3 \rightarrow 2$ and by the nodes $8 \rightarrow 5 \rightarrow 2 \rightarrow 1 \rightarrow 4 \rightarrow 8$. In the flow graph with 6 nodes, only the primary curl is extracted, but by the additional nodes in Figure 4.6(d), the secondary curl is indicated as well, which is formed by the nodes $2 \rightarrow 3 \rightarrow 5 \rightarrow 2$. Because the stable state of the lid-driven cavity was used as input, almost every feature is affected by the main flow direction. The only exception is node 6 in Figure 4.6(d), which in consequence is very small and the outgoing edges indicate a flow opposite to the main direction.

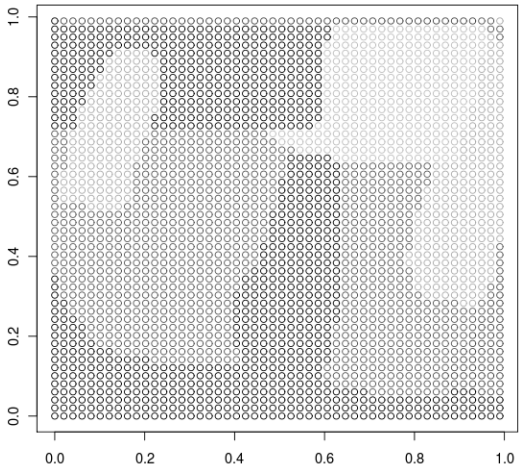
Choosing the optimal number of clusters for a given data set is still an open research topic. Consequently, the results for the optimal number of clusters for the introduced clustering evaluation techniques differs significantly. While the algorithm of Langfelder et al. [88] approximates the optimal cutting level $ocl = 6$, the evaluation by the silhouette coefficient computes $ocl = 4$ for the steady state of the lid-driven cavity. Because of this difference in judging the optimal cutting level, the experiments are performed for the complete time interval to test the evaluation methods for different settings. Although the appearance of the vector fields changes heavily, the optimal number of clusters computed almost remains identical. The approximated cutting level computed by the method of Langfelder is in the interval $[4; 6]$ and the approximation by the silhouette and the Dunn's index results in a smaller number of clusters. Therefore, in Figure 4.6, several flow graphs are shown for different time steps and different numbers of extracted features. This can be interpreted as the level-of-detail of the representation. The larger the number of features is, the finer the smaller structures are resolved. And in addition, for a smaller number of features the overall structure is portrayed better. The flow graph constructed on 6 nodes



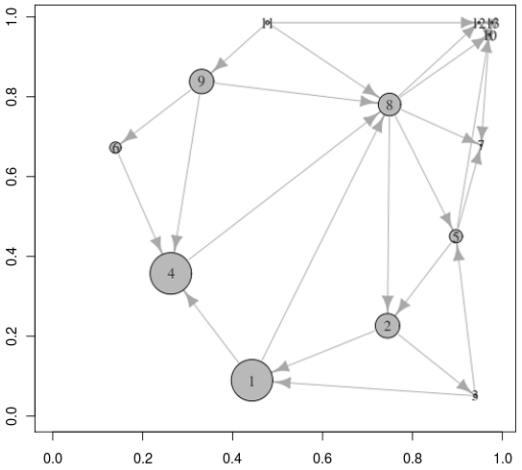
(a) 6 extracted features



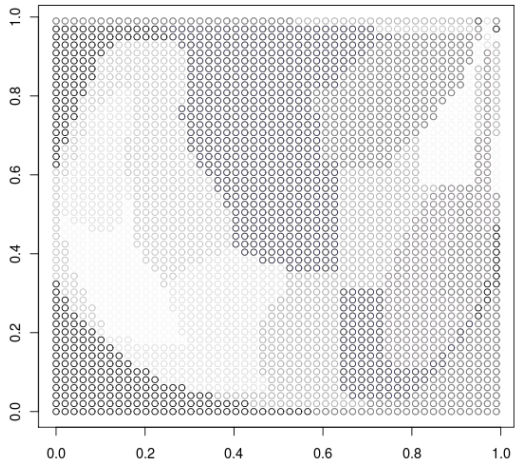
(b) Flow graph for 6 extracted features



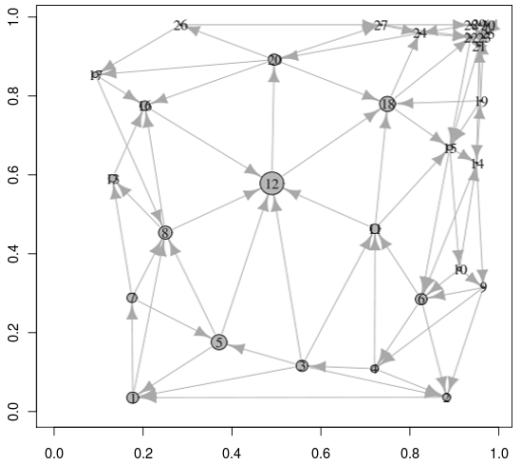
(c) 13 extracted features



(d) Flow graph for 13 extracted features



(e) 27 extracted features



(f) Flow graph for 27 extracted features

Figure 4.6: Lid-driven cavity simulation: The extracted features are shown side-by-side with their corresponding flow graphs.

(Figure 4.6(b)) hides too many details and in consequence not a good representation. The flow graph with 13 nodes (Figure 4.6(d)) was selected, because it balances the ratio between overview and details best. If choosing too many nodes, e.g., 27 nodes in Figure 4.6(f), the representation becomes too cluttered.

Because the computed levels of the clustering evaluation techniques do not represent the vector field good enough, the decision is left to the user, who can choose the best representation level a posteriori. The construction process is build to parse the dendrogram beginning at a cutting level, which is larger than the suggested optimal cutting level of the evaluation techniques. This way, we make sure that the user can pick the desired level, which is smaller than the threshold, interactively without any re-computation of the graph structure. In the next chapter, we will develop a novel approach to compute the optimal cutting level of the hierarchy, which consists of the optimal number of clusters. In advance, we need to introduce some other techniques. Nevertheless, the construction of the flow graphs and the expressed relation among the nodes allows the reduction of the extracted nodes in comparison to the simple feature extraction (compare Figure 3.14). This reduction simplifies the representation by conveying the same information with less entities. Furthermore, the flow graphs are simpler and easier to understand, even for such a simple visualization, the coloring of the features is challenging to make everything stand out, which we actually failed for a full color representation as well as for black and white in Figure 4.6(a), 4.6(c), 4.6(e).

Finally, the runtime of the graph construction process is examined on the series of lid-driven cavities, which was also used to estimate the runtime of the feature extraction. Strategy II performs faster than the others, because it is a simple look-up of the dendrogram. The other two strategies are slower, because querying the neighboring cells is very expensive. Therefore, reducing the number of checks is crucial for the overall runtime. Even for small data sets, i.e., several ten thousands cells, the costs for checking dominates the runtime. Therefore, the complexity is dependent on two parameters, the number of cells and the number of extracted features.

The dependency on the number of features can be observed, when the runtime of strategy I is compared to that of strategy III, which requires 30% less comparisons for the same graph. For more features, this difference even increases. We observed a linear grow in spent time for an increasing number of features., i.e., $O(h)$ for h features, while more input cells result in a quadratic increase in runtime, i.e., $O(n^2)$ for n input cells.

number of cells	10000	10000	10000	10000	10000
	deriving add. data	10	20	30	40
time (sec.)	12	3	5	11	23
number of cells	20000	20000	20000	20000	20000
	deriving add. data	10	20	30	40
time (sec.)	22	8	18	37	67
number of cells	40000	40000	40000	40000	40000
	deriving add. data	10	20	30	40
time (sec.)	74	17	34	69	118

Table 4.3: Runtime for incorporating different numbers of cells, different numbers of nodes (10, 20, 30, 40) and for deriving additional data, e.g., divergence, rotation, aso.

4.5 Summary and Discussion

In this chapter, flow graphs were defined and an effective construction for them was presented. The geometric flow graphs are the major link between vector field analysis and graph theory. Although graph-like representation has been published already, those approaches do not include a precise description of the nodes and edges, which is essential for a quantitative analysis. In particular, the following points have been examined:

- Geometric graphs were chosen as an appropriate representation for vector fields. Their definition was transfered to our specific context as flow graphs.
- The properties of the cell's feature vector (Definition 3.1) were transfered from the individual cells to the corresponding features, which are describing the properties of the flow graph precisely.
- Representing the vector field by the flow graph, the number of necessary nodes could be reduced. The graph representation extracts the secondary curl with fewer clusters than the simple clustering technique in the previous section does.
- The functionality of the proposed technique was shown on a generic data set, which covers all possible scenarios for the edges' construction. Furthermore, it was shown, that the generic data set is close to a real one and the results were conveyed to the simulated data set of a back ward step.
- The optimal number for the extracted features was discussed briefly. Because of the ambiguity in the optimal number, this algorithm is constructed such that the user decides for the best number.

- The graph representation was compared to traditional visualization techniques and the advantages and disadvantage have been discussed.

To conclude this chapter, the flow graph construction process is summarized:

For the level h of the built feature hierarchy $C(h)$ the flow graph construction fcg maps the extracted features onto the geometric flow graph $G(V, E, L)$, $fcg: C(h) \rightarrow G$. Every feature $c_{i,h} \in C(h)$ is mapped onto exactly one graph's node $n_i \in V$ and the aggregated properties of $c_{i,h}$ specify the node's label. The candidates for the graph's edges are extracted from the higher levels of the feature hierarchy $C(l),\ 1 \leq l \leq h$. The feature $c_{t,l-1}$ is formed by the features $c_{i,l}, c_{j,l}$ of the next lower hierarchy level. All features $c_{s,l}$ sharing common points with $c_{t,l-1}$ are reviewed if they also have common points with one of the features $c_{i,l}, c_{j,l}$. If so, the edges $e(n_s, n_i)$ respectively $e(n_s, n_j)$ are added to E and the weighted flow for this edges is added to L.

Chapter 5

Graph Evolution and Comparison

In the first part of the presented workflow the features were extracted and, based on those flow graph was constructed. The features form the set of nodes of the geometric flow graph, which stands for the static appearance of the vector field. Its dynamics are indicated by the edges connecting the nodes. The edges symbolize a transportation process between the nodes and, consequently, an exchange between them. Furthermore, the nodes and the edges can be labeled with properties to obtain a geometric flow graph. Thus, the flow graph represents the vector field and the graph's properties can be used to analyze the vector field directly. This analysis can easily include a uniform description of the flow graph along with its properties as discussed in the previous chapter. Now, the focus of the analysis is on the comparison of different vector fields from different simulations, which are represented by their flow graphs. In contrast to established comparisons, our method is not built on the visualization, but on the representing graphs, which allows for the application of graph matching and similarity techniques. This analysis is a second knowledge extraction part in our workflow, which directly follows the first one.

Traditionally, vector fields are analyzed visually and so the comparison of vector fields is based on their visualization. The user needs to build a visualization pipeline for every vector field before he is able to compare them. The comparison usually measures the similarity of distinct properties of the visualization, e.g., the vectorial properties of the glyphs [67]. Thus, the visualization pipelines must include the same techniques. Additionally, the parameters inside the pipeline must be identical. For instance, if using glyphs to describe the vector field, the placing of the glyphs must be identical in order to be able to compare the glyphs between different vector fields.

To overcome those limitations, visualization toolkits, e.g., VisTrails or ParaView , provide the possibility to run batch jobs, which execute a previously defined visu-

alization pipeline for a set of input files. This is a convenient feature for the user, but it bears some danger, too. For instance, as already shown previously, for poorly seeded path lines the quality of the visualization suffers heavily. The seeding problem becomes even more troublesome for a general placement of the seeds for several vector fields. Therefore, the visualization pipeline defined initially must be built carefully, otherwise the results are not optimal and the comparison may lack details. Despite those problems, there are comparison techniques specialized on different visualizations. For instance, Jiang et al. [79] assume vortexes to be the most significant features of vector fields. They extract those vortexes and base their vector field's description on them. There are also other comparison methods, which are specific for one visualization technique and its corresponding feature definition, e.g., critical points [130], glyphs [67] or Lagrangian coherent structures [115]. Because those comparisons are always based on a distinct visualization, the extracted properties rather describe the visualization entities than the actual vector field, i.e., the link between the input and the representation is not as direct as for our presented technique. Also, the significant amount of user input which is necessary to build one visualization pipeline for all the vector fields to be compared is a major drawback of the traditional comparison methods. This problem becomes even more pronounced for simulations with increasing complexity.

The feature extraction techniques introduced previously are fully automated, which increases the throughput of the workflow. The user can now process the complete time sequence of vector fields that is generated by the simulation automatically. Furthermore, the applied definition of features (Definition 2.6) is less restrictive and in consequence, better applicable than more specialized definitions like those used for visualization techniques. Instead of defining specific features, e.g., vortexes or critical points, the presented feature extraction is based on regions of interests which are composed of data points exposing similar properties and behavior. These elementary properties and their behavior are derived from the points and cells forming the region. The approach presented in this thesis keeps all available information from the atomic level of data representation up to the abstract level of representation by flow graphs. The method of comparison derived in the following exploits all the available elementary information of the description, in contrast to the traditional visualization, which constructs complex descriptions derived from just a subset of this information.

Three different comparison scenarios are possible which are to be distinguished. In the first case, vector fields of one simulation are compared with each other. In particular, the quality of the representation for different cutting levels of the hierarchy

is examined, which leads to a better technique for extracting the optimal cutting level *ocl* (Definition 4.3). As well, this case describes the evolution of vector field for t time steps of the simulation. By the analysis of the evolution information on the simulation itself can be obtained, e.g., the changing rate between two consecutive time steps to occur specifies characteristics on simulated fluid or medium. To compare the results of different simulations, either this changing rate can be used or the similarity of vector fields of different simulations is computed directly. The latter is the second case, when two simulations, which are based on different mathematical models, are compared, e.g., backward step and lid-driven cavity.

In the following section, we briefly summarize the related work on vector field comparison and distinguish between the evolution of graphs and their comparison. The two classes are differentiated by the applied reference frame of the simulation. Suitable graph matching and similarity techniques are assigned to these classes. In the experimental evaluation the profit of the novel techniques is demonstrated and additional information on the simulation extracted. The chapter is concluded by a short summary and a discussion.

5.1 Related Work

If one compared two vector fields point-by-point, the result would be a vector field again, which is as difficult to analyze as the initial set. Therefore the direct comparison of two vector fields is not reasonable due to the sheer size and complexity of the input data. Thus, instead of examining the changes of the overall vector field, the analysis is simplified by the feature extraction. The features represent either distinct points, e.g., critical points, or stand for a group of points, if they are defined as regions of interest. In both cases the similarities or differences of the vector fields are described by these features and their properties.

For a visualization based on critical points, the trajectories for these points are computed to describe the changes in the vector field. This approach is common and often used to describe the evolution of vector fields [[121], [30], [130], [145]]. The analysis based on critical points can also be enriched by adding additional visualization entities. For instance, Chen et al. [30] add orbits to indicate the influence of the critical points on their surrounding. Shi et al. [121] visualize the influence sphere of the critical points by sampling data points close to them and computing their trajectories over the time steps. Figure 5.1 is taken from Theisel et al. [130] and visualizes the critical points along with their trajectories. The lowest level shows the starting time

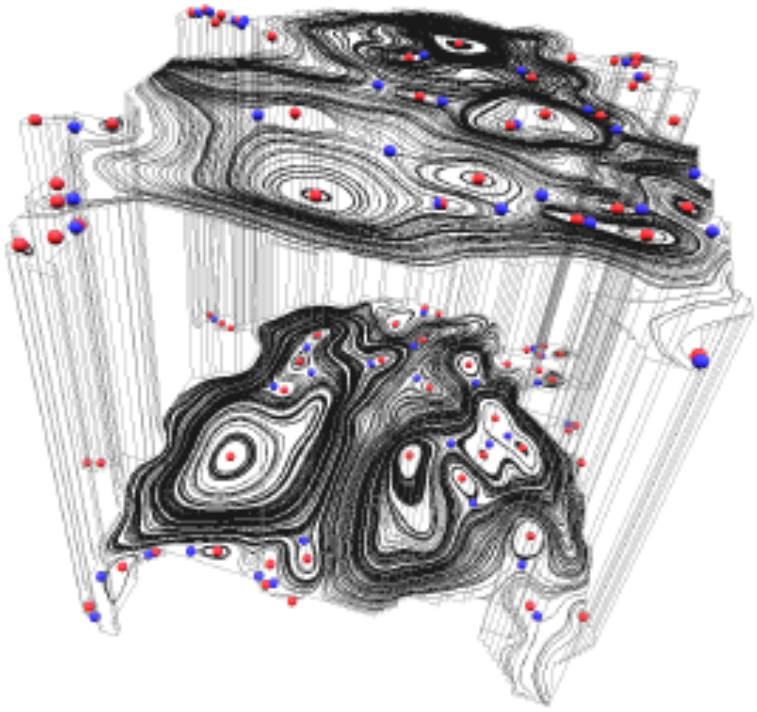


Figure 5.1: Critical points and their trajectories over several time steps. Trajectories describe the evolution of the vector field. Taken from Theisel et al. [130].

step and the highest level the vector field for a later time step. Those in between have been omitted and only the positions of critical points are shown. The example demonstrates the complexity and problems of a visualization with more than two dimensions. Here, the underlying vector field is just two-dimensional and the time is portrayed on the third dimension. Thus, including another dimension is impossible. This shows the limitation of the traditional visualization techniques once more, which becomes even more restrictive for portraying the evolution of vector fields over time.

Consequently, a more abstract method that is not based on visualization is necessary to analyze the evolution and compare vector fields. Nevertheless, there are techniques trying to solve those issues visually. Verma et al. [139] apply path lines as primary visualization. They compute path lines for the two vector fields to be compared. The seeding of the lines is identical and the differences in the flow properties are represented by the path lines. For their comparison complex geometric structures must be defined, which express the differences of the path lines. In Figure 5.2, four examples of those geometric structures are applied to two path lines of two different vector fields. This comparison considers only the position of the critical points. For regions of interest, one specific data point must be selected, which represents the

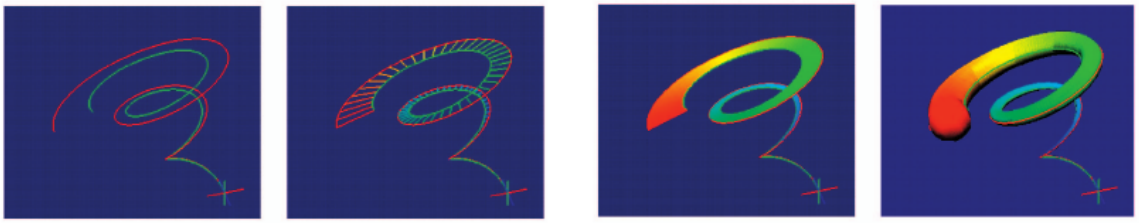


Figure 5.2: Geometric structures expressing the differences between two path lines. Taken from Verma et al. [139].

regions or volumes, respectively. Tracing or tracking the volume over time is the first step to describe the development of the vector field, because it not only describes the features' positions in spatial space, but also examines their evolution [[125],[84], [126]]. Not only does the features' positions differ from time step to time step, but the properties are changing, too.

In the simplest case the number of features stays constant over time, which is very unlikely for an evolving vector field, in particular if the complete time interval of the simulation is examined. The different scenarios for the feature's evolution are nicely summarized by Samtaney et al. [117]. They describe the features as amorphous regions and are observing their evolution as it is shown in Figure 5.3. During the simulation, features can dissipate, bifurcate, amalgamate or be created. All of these processes effect the total number of features representing the vector field and must be considered to describe the vector field's evolution, in particular for a quantitative description.

Another approach for the comparison is the utilization of flow graphs as presented in this thesis. So far, there is very little work on the evolution and comparison of vector fields within the visualization community, which is probably caused by problems and disadvantages of visualization techniques, e.g., the significant user bias on the results and the applied complex structures, which are used for the visual feature extraction but remain difficult to describe numerically.

The newly developed flow graphs provide not only a general and comprehensive representation of vector fields. They allow the description and comparison of the vector fields' properties and their behavior, as well. The representation is based on the extracted features and differences between them are the key for examining the similarity of the initial vector fields. Because of the link between visualization and graph theory established by the proposed flow graphs, techniques for graph comparison and similarity can be applied to examine vector fields.

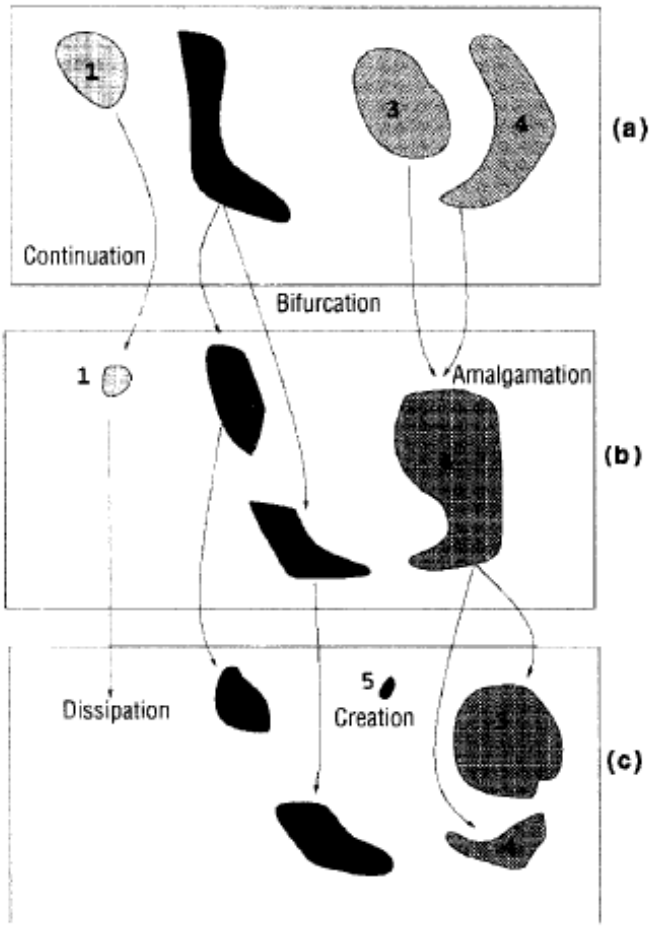


Figure 5.3: Scenarios for feature evolution. Regarding the complete time interval of the simulation, different cases of feature evolution can occur. Features are defined as regions of interest resulting in a more complex evolution process than for points of interest. Taken from Samtaney et al. [117].

Graph matching, which includes the similarity and comparison of graphs, is well studied and many different surveys exist. Here, we follow the work of Armiti [14]. Geometric graphs $G = (V, E, L)$ are formed by three sets of entities, namely nodes V, edges E connecting the nodes and their corresponding labels L (see Definition 4.1). Finding correspondences between two graphs is based on the similarity of these entities, but graph matching approaches treat the importance of matching differently. Two graphs can be matched by their nodes, by their structure consisting of the nodes and the edges or by the complete description, including the labels. For two extracted flow graphs G_1, G_2, we distinguish three different scenarios:

I Graph Evolution: The two given flow graphs G_1, G_2 have been computed by an identical simulation based on the same mathematical model. Therefore, both graphs expose very similar structural properties and their major differences are differently labeled nodes.

II Graph Comparison: The two given flow graphs G_1, G_2 have been computed by different simulations solving different mathematical models. Therefore, they differ significantly in their structural properties.

III Graph Simplification: G_1 is a subgraph of G_2. G_2 is possibly built by several similar substructures like G_1. Those structures may have different properties but are structurally similar.

The last scenario differs significantly from the other two, because it is not only matching two graphs onto each other, but also searching for multiple occurrences of subgraph G_1 has to be carried out. We assume an initially defined subgraph G_1, but this graph can also be detected and matched automatically. In general, this task can be solved by similar techniques as the other two cases, but the focus there is on the comparison and less on the search for subgraphs. Note the difference between the terms similarity, comparison and matching. Similarity measures the differences between the nodes and/or edges. Based on the computed similarity the graphs are compared and a matching is computed by the correspondences between the graphs, such that the similarity is maximal.

Each of the three scenarios requires a different view on the graphs and their properties. The most simple approaches for comparison are solely built on the similarity of the nodes, e.g., link prediction [31], web search [20] and frequent subgraph discovery [104]. Only the similarity of the nodes is taken into account, i.e., the structure of the graph specified by the edges is ignored. This matching strategy suits the evolution of graphs, for which little differences between the flow graphs are expected. If the graphs differ more, the precision and the quality of the matching can be increased by considering more details of the graphs while computing their similarity. This could be the case for two flow graphs constructed for two non-consecutive vector fields. More complex approaches take the node's neighborhood into consideration, either just locally by adding the node and its incident edges [155] or globally by including the node, its incident edges plus the neighboring nodes below a certain distance [140].

Another way to extend the scope of the comparison is to include the structure of the graph in the similarity computation. Umeyama [135] includes the structure encoded by the weighted adjacency or weighted Laplacian matrices, in which the

weights stand for the distances of the edges. Then the spectra of the graph are computed by the Eigendecomposition of the matrix. The Eigenvectors define the corresponding spectral feature for every node. To measure the similarity of two given graphs, the distance of the spectral features is computed. Since all nodes are included in the adjacency matrix, this approach is considered to be global. In general, global approaches lack in flexibility comparing graphs which are formed by different numbers of nodes. Zhu et al. [156] solved this issue by truncating the special features and keeping the ones with the most dominant Eigenvalues only. Another global approach was proposed by Cheong et al. [32]. They select a subset of the graph's node as landmarks and compute the shortest path from every graph's nodes to these landmarks. Additionally, they suggested the usage of landmarks as a bounding box for the graph, i.e., the landmarks are chosen to be the most outside nodes. So far, the major aspects for the comparisons are the structural properties of the graph and the metric distance of the nodes, but no information of the labels has been considered for the algorithms mentioned previously. Therefore, structural graph matching seems to suit the comparison of two flow graphs from different simulations.

In contrast to global approaches, the local ones offer greater flexibility and are more robust to changes in the structure of the graph especially to those in the number of nodes. The histogram-based approach is one of the oldest based on local features [[54], [75], [82], [129]]. A histogram is computed by the spatial properties of every node and its direct neighborhood. The result is a two-dimensional histogram in which the spatial distance between edges and the angles between the attached edges are stored. The histograms of two graphs can now be compared by the Bhattacharyya distance [28] to compute their similarity.

Very little can actually be found on matching geometric graphs. We assume that this is the case because of the graph's distinct properties, which are unique to them. However, some local-based feature approaches can be modified to fit to the specific properties of geometric graphs. The connectivity of the nodes is limited to a certain diameter, which decomposes the graph into subgraphs. Subgraph features have the highest selectivity power, but finding the optimal matching between two subgraphs is as complex as the graph matching itself. Therefore, the subgraphs are simplified to a single node with its attached edges, which is called vertex signature [14], clique [122], local structure [112], start [152] or subgraph [110]. The similarity of the graphs is now computed by the necessary affine geometric transformations to map all vertex signatures of one graph onto those of the other one [42]. However, this matching technique is still sensitive to the number of nodes in the graphs.

Of the numerous possible graph or node matching techniques each has its own advantages and disadvantages and not every technique is suitable for every scenario. The choice of the appropriate method depends strongly on the data set and the matching scenario. We distinguish between those mentioned previously: comparing two graphs of the (I) same or (II) different simulations and (III) finding correspondences inside a graph.

Flow graphs are formed by the nodes, which represent the vector field's features, and edges, which symbolize the relation of the nodes. Both graph entities are labeled with numeric values specifying their properties. The similarity measure can take one or all of these components into consideration. Therefore, the correspondences between the graphs can be found by either finding similar nodes of the graphs, by matching the graphs' spectra or by a combination of both. Note that, since the nodes' coordinates are included in the labels (Definition 4.1), those are necessary for the comparison.

In scenario I the vector field evolves steadily over the time interval of the simulation. For every time step the mathematical model is solved, some of the solutions are stored as vector fields, others are dropped as intermediate results. At the end of the simulation a sequence of vector fields has been generated. In these vector fields structures can slowly occur, change their properties and disappear. The processes typically do not happen suddenly, because the simulation is typically set up to convey this evolution by as many time steps as necessary. Therefore, the assumption of a steadily evolving vector field is valid. Regardless, this will be an aspect of the experimental evaluation. Furthermore, by examining the similarity of the flow graphs for different cutting levels of the hierarchy, an *ocl* is derived.

Inside the vector field similar structures may be formed, e.g., primary and secondary curls for the lid-driven cavity simulation. Because these structures are formed under the same circumstances and by the same conditions of the vector field, their properties are very similar and their structure is congruent, i.e., the structures can be mapped onto each other by affine geometric transformations. Therefore, the labels are less interesting than in the previous scenario, because the numeric properties are influenced directly by the transformation. Different are the spectra of those structures, which stay the same for affine transformations.

Scenario II is the more challenging scenario comparing flow graphs from different simulations. Because of the different setups the references frames for both simulations may be different and consequently the labels, as well. This requires either a normalization to a common reference frame, which is unpractical for high-dimensional data, or a relative comparison of the labels. However, the third scenario (III) is the most

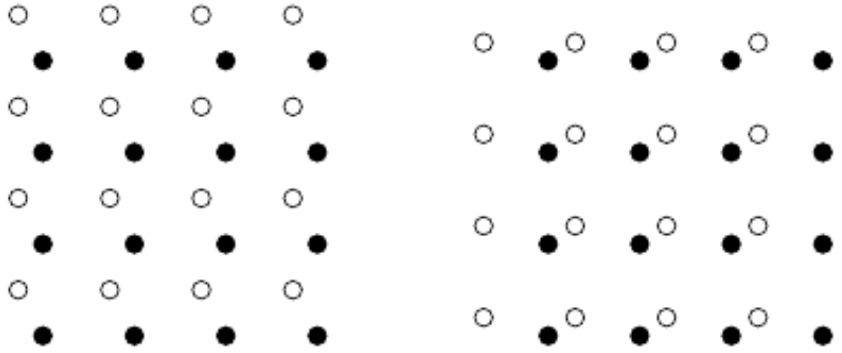


Figure 5.4: Grids of black and white dots shown for consecutive time steps. This setting is the worst case scenario for motion estimation, because the correspondence problem cannot be solved in a unique manner. Taken from Jähne [77].

difficult one, because it includes the extraction of repetitive structures inside individual flow graphs. This detection is beyond the scope of this thesis and we simplify this scenario by comparing pre-defined, prototypical graphs, i.e., the actual detection is omitted and replaced by a set of initially defined graphs. In the following, we will discuss the evolution and the comparison of graphs and appropriate techniques for each of them.

5.2 Graph Evolution

This scenario is the most practically relevant in terms of increasing the throughput of the analysis workflow and the quality of representation. The evolution of the flow graphs describes the development of the vector field for the complete time interval of the simulation. It consequently represents the complete simulation rather than a single time step. Furthermore, the last missing information is gained from the evolution process to remove the remaining user input from the presented workflow.

Because the nodes are not uniquely labeled or named and the nodes properties may vary, the correspondences between two flow graphs are not established easily. Actually, all possible combinations of correspondences must be tested in order to find the optimal matching. This so-called correspondence problem is found in many other applications which are trying to match two objects. For instance for motion estimation in image processing, the corresponding object must be found in different images to compute its displacement [77]. If there are several objects in one image, which are very similar to each other or even identical, the correspondence cannot be

computed uniquely anymore. In the worst case shown in Figure 5.4, the movement of the individual dots in relation to the others cannot be computed anymore at all. To resolve this ambiguity, an additional constraint is added, namely that the displacement between the frames is assumed to be small. We also add this constraint to the description of the evolution process and assume steadily, but slowly changing graphs, i.e., the spatial displacement of the nodes and the change of the properties is small. Assuming a small displacement between the time steps is equivalent to restricting the node's neighborhood which is considered for the matching to a certain diameter [140].

For two time steps t, $t+h$ of a simulation, the flow graphs G_t, G_{t+h} are constructed for the corresponding vector fields. The time difference h between the vector fields is assumed to be small, i.e., the flow graphs are not necessarily constructed for consecutive time steps. The nodes of flow graph G_t, which, without loss of generality, consists of fewer nodes than G_{t+h}, are matched onto the nodes of flow graph G_{t+h}, i.e., if $\#(G_t) < \#(G_{t+h})$ at least one additional feature has been created during the time span h.

For the correct execution of Algorithm 5.2, which computes the matching $M(G_t, G_{t+h})$, a small displacement of the flow graphs' nodes between the two time steps is assumed. The result is a locally optimized solution for the matching problem and the difference between the flow graphs. The assumption of a small displacement guarantees that the locally found optimal solution is also correct globally. It is an inexact matching, which minimizes the graph distance 5.2. This solution to the matching and the correspondence problem is also found in other applications, e.g., the algorithm of Lucas and Kanade [94]. In the experimental evaluation we demonstrate that the algorithm computes the optimal and correct solution.

The main matching criteria is the distance between the nodes' labels. Those labels are formed by the spatial coordinates and the flow properties. They are defined as an attached vector for each individual node of the flow graph (Definition 2.12). If a node a from G_t is matched with the correspondent node b of G_{t+h} the distance is computed for the matching $m(a, b)$ by Euclidean distance of the vector $labels$, which include the node's coordinates and properties. In analogy to the graph edit path, the distances for all matches is summed up to compute the matching distance $md(\cdot)$ between both graphs.

Algorithm 5: Computing corresponding nodes

Data: Flow graphs G_t, G_{t+h}
Result: Matching $M(G_t, G_{t+h})$ and the distance $(G_t, G_{t+h})_2$

```
if #(G_t) < #(G_{t+h}) then
    N_A = list of nodes G_t ;
    N_B = list of nodes G_{t+h} ;
else
    N_A = list of nodes G_{t+h} ;
    N_B = list of nodes G_t ;
difference of grahs = 0 ;
init matching as empty list ;
while N_A is not empty do
    a = first element of N_A;
    difference of nodes = 0 ;
    forall b in N_B do
        if ||labels_a, labels_b||_2 < difference of nodes then
            difference of nodes = ||labels_a, labels_b||_2 ;
            candidate for matching = b ;
    add element (a, candidate for matching) to matching;
    difference of graphs = difference of graphs + ||label_a, label_b||_2;
    remove a from N_a ;
    remove b from N_b ;
forall b in N_B do
    difference of graphs = difference of graphs + ||label_b||
return(matching, difference of graphs) ;
```

Definition 5.1. *Matching Distance*
For the matching $M(G_t, G_{t+h})$ of two graphs G_t, G_{t+h} the matching distance $md(G_t, G_{t+h})$ is defined as the sum of the distances between the matched nodes $m(a, b)$

$$md(G_t, G_{t+h}) := \sum_{m(a,b)\in M} \|label_{m,a}, label_{m,b}\|_2 \tag{5.1}$$

The distance $md(\cdot)$ fulfills the constraints of a metric. It is non-negative by construction, because it adds differences and addition is the only involved operation, which is non-negative itself. The proposed graph metric is zero, if, and only if, both flow graphs are identical, which is necessary for the identity of indiscernibles. Since the smaller graph is always matched to the larger one, the symmetry is fulfilled in all

cases except for graphs of the same size. For the latter case the same argumentation as for the non-negativity applies, because the addition is the only involved operation, which is symmetric itself. The triangle inequality is proven in analogy. The matching distance measures the difference between the matched nodes of both graphs, but does not consider the creation or dissipation of nodes. To handle the resulting change in the number of nodes, the graph distance $gd(\cdot)$ adds the labels of the unmatched nodes as penalty to the matching distance.

Definition 5.2. *Graph Distance*
For two graphs G_t, G_{t+h} their graph distance $gd(G_t, G_{t+h})$ is defined as the matching distance $md(G_t, G_{t+h})$ of the matching $M(G_t, G_{t+h})$ plus the sum of the labels, which are not matched

$$\begin{aligned} gd(G_t, G_{t+h}) &:= md(G_t, G_{t+h}) + \sum_n \|label_n\|_2 \\ n \in G_{t+h} &\quad , \quad n \notin M(G_t, G_{t+h}) \end{aligned} \tag{5.2}$$

Because the main matching criteria are the nodes' labels, the presented algorithm requires an identical reference frame. This requirement is obviously met by the vector fields of one simulation, but for different simulations the reference frame is likely to be different. If two complete sequences of vector fields are supposed to be compared to each other by their flow graphs, the differing reference frames do not influence the result, because all properties are scaled equally. For both sequences, a similarity is computed, that expresses the evolution of the vector field over time. This corresponds to the definition of the Reynold's number and describes the overall changes of the vector field. For simulations based on the same setup, but slightly modified parameters, the similarity already expresses the impact of the modifications on the results.

If comparing two single flow graphs of different simulations, the graphs need to meet two requirements to allow the application of Algorithm 5.2. They need to be similar and the reference frames must either be identical or mapped onto each other. The mapping transfers one data set into the reference frame of the other one, but still, both vector fields need to be similar in order to fulfill the constraint of a small displacement between the flow graph's entities.

5.3 Graph Comparison

Since the requirements on the input data limit the application of Algorithm 5.2 to a specific scenario, namely scenario I, some less restrictive comparison methods need to be developed being applicable in more general settings as in the scenarios II and III. For flow graphs derived from vector fields of the same simulation or similar simulations the assumption of identical reference frames is met, either directly or by a mapping of the reference frames. However, for flow graphs of two vector fields from a different origin, the corresponding graphs' labels are described according to the individual reference frame of each vector field. Consequently, the labels cannot be used anymore as major criteria for the comparison. Even if a mapping between both reference frames can be defined, which is more difficult than for similar simulations, the second constraint of a small displacement between the graphs entities is not met.

Without any labels, the flow graphs are reduced to directed graphs. Directed graphs are described by their nodes and edges. Therefore, only their properties are available for a comparison, i.e., the major comparison criteria is the graph's structure described by the edges connecting the nodes.

To solve the matching problem on this limited set of information, Umeyama [135] proposed a spectral matching for graphs solely based on their structure. Each graph is represented by its adjacency matrix for which the Eigenvectors are computed. Those Eigenvectors span the Eigenspace in which the graph's nodes are embedded. For the two adjacency matrices of the graphs A_1, A_2, the graph matching is defined as optimization problem

$$\min_P \|\|PA_1P^t - A_2\|\|^2 \tag{5.3}$$

to find the optimal permutation matrix P, which minimizes the objective function with respect to the Frobenius norm. The permutation matrix is of the same size as the adjacency matrices and all its entries are element of the set $\{0; 1\}$. It specifies the permutation of the nodes between both graphs. In this basic version of the algorithm the graphs must have the same number of nodes, but modified versions exist, which can handle graphs of different sizes [156]. Another disadvantage of this algorithm is the multiplicity of the Eigenvectors, which may lead to incorrect matchings. Umeyama orders the Eigenvalues according to their magnitudes before using the Hungarian algorithm [86], which computes the actual matching.

Singh et al. [127] developed a matching algorithm for proteins, which are represented by graphs. Their algorithm is formed by two major stages. In the first, each possible matching between nodes of the two graphs is scored. This score R_{ij} is used in the second stage to match the two graphs G_1, G_2 and considers the local similarity of the nodes $i \in G_1, j \in G_2$, i.e., not only the nodes themselves, but their neighborhoods are considered in the computation [127], as well.

$$R_{ij} = \sum_{u \in N(i)} \sum_{v \in N(j)} \frac{1}{\|N(u)\|\|N(v)\|} R_{uv} \tag{5.4}$$

where $N(i), N(j)$ are the neighboring nodes for the corresponding nodes i, j. Since the score R_{ij} is dependent on the score R_{uv} of the neighboring nodes, non-local effects are also considered in the computation. By computing the score for all nodes and transferring it into matrix form, the problem can be interpreted as Eigenvalue problem

$$\begin{aligned} A[i,j][u,v] &= \begin{cases} \frac{1}{\|N(u)\|\|N(v)\|}, & if (i,u) \in \text{edges of } G_1 \text{ and } if (j,v) \in \text{edges of } G_2 \\ 0 & otherwise \end{cases} \\ R &= AR \end{aligned} \tag{5.5}$$

The Eigenvalue problem is now solved iteratively by the power method [58]. Once R has been computed, the matching nodes are extracted. Interpreting R as encoding for a bipartite graph and finding the maximum-weight bipartite matching leads to the optimal matching [103]. Each side of the bipartite graph consists of the nodes of one of the input graphs G_1, G_2, from which the matching nodes are now extracted by maximizing the score. This is possible because of the special properties of bipartite graphs. They consist of two sets of nodes such that no nodes of the same sets are connecting with each other by an edge.

Both algorithms belong to the spectral matching algorithms, because they use different strategies to select the Eigenvalues. Thus, we can verify them with each other. The techniques mentioned previously only consider the graphs' structures, an intentional feature, since their labels are dependent on the reference frames. In order to extract additional properties on graphs, we review those and the dependence on the reference frames. The only independent property is the volume of the feature, as it is defined by the ratio of the feature's volume $V(f_i)$ over the volume of the complete

vector field $V(\mathcal{V})$. Based on this ratio a weighting matrix W is calculated. The entries $w_{ij} \in W$ represent the similarity of the feature's volume which is approximated by:

$$w_{ij} = 1 - \left\| \frac{V(f_i)}{V(\mathcal{V}_1)} - \frac{V(f_j)}{V(\mathcal{V}_2)} \right\|_2 \tag{5.6}$$

The last scenario (III), the graph simplification, can be seen as a generalization of the graph comparison. The flow graph is searched for similar structures, e.g., the primary and secondary curls inside the lid-driven cavity. Solving this problem can be realized by a two step searching strategy. First, the nodes and their local neighborhood must be described in an appropriate way to make those structures comparable to each other. The description used cannot be based on the labels, because the structure likely differs in size and other properties. For instance, the primary curl is larger than the secondary one, but also spins in the opposite direction with a different speed. The structure is not sufficient for the comparison as well, because the previously described graph comparison algorithms, which are based on the graph's structure, just take the connections of the nodes into account. Therefore, the structure must be described more precisely, e.g., by the spatial angles and distances between the nodes. Asyer [14] introduced vertex signatures to solve this problem, but only for two-dimensional geometric graphs. We refer to his work for more details on this specific scenario.

5.4 Experimental Evaluation

The evaluation is split into two. In the first part the evolution of graphs is examined, i.e., scenario I. For the graph evolution the flow graph's labels are compared to examine and eventually measure the similarity and the differences between two vector fields being represented by their flow graphs. Typically, those vector fields have been computed by the same simulation for two chronologically close or even consecutive time steps of the time interval observed in the simulation. Therefore, the graphs' structures are very similar and the major changes are expected to occur in the nodes' labels, which describe the properties and the flow behavior of the cells forming the corresponding feature. To describe the graph evolution, an optimization technique is modified to fit the requirements of this particular scenario. This technique is found in other applications, too. Nevertheless, a proof of concept demonstrates its correctness for our setting. Following that, the complete sequence of vector fields computed by

one simulation is examined to describe the evolution of the constructed flow graphs. The evolution gives further insight into the simulation and its characteristics.

The circumstances of the comparison of graphs (scenario II) differ significantly from the graph evolution (scenario I), because here the changes are expected in both, the graphs' structures and the labels. Most of the label's components are given for the specific reference frame of the simulation and are not invariant to any translation. Indeed, for two flow graphs which were computed by different simulations, their reference frames probably differ and consequently, the labels cannot be considered for the comparison. The last remaining information on the graphs is their structure specified by the nodes and the connecting edges, which is represented by the corresponding adjacency matrices. By neglecting the graph's labels geometric flow graphs are trimmed to direct graphs, for which many comparison techniques exist already. The proof of concept is therefore omitted and the focus of the evaluation lies on the knowledge which can additionally be extracted by comparing graphs with each other.

5.4.1 Graph Evolution

The evaluation is structured in similarity to the previous ones. Because there are actually no comparable techniques for this particular context, the evaluation starts with a proof of concept to validate the proposed algorithm and the correctness of the computed results. The focus lies on the assumption of a small displacement h and the range from which h can be chosen.

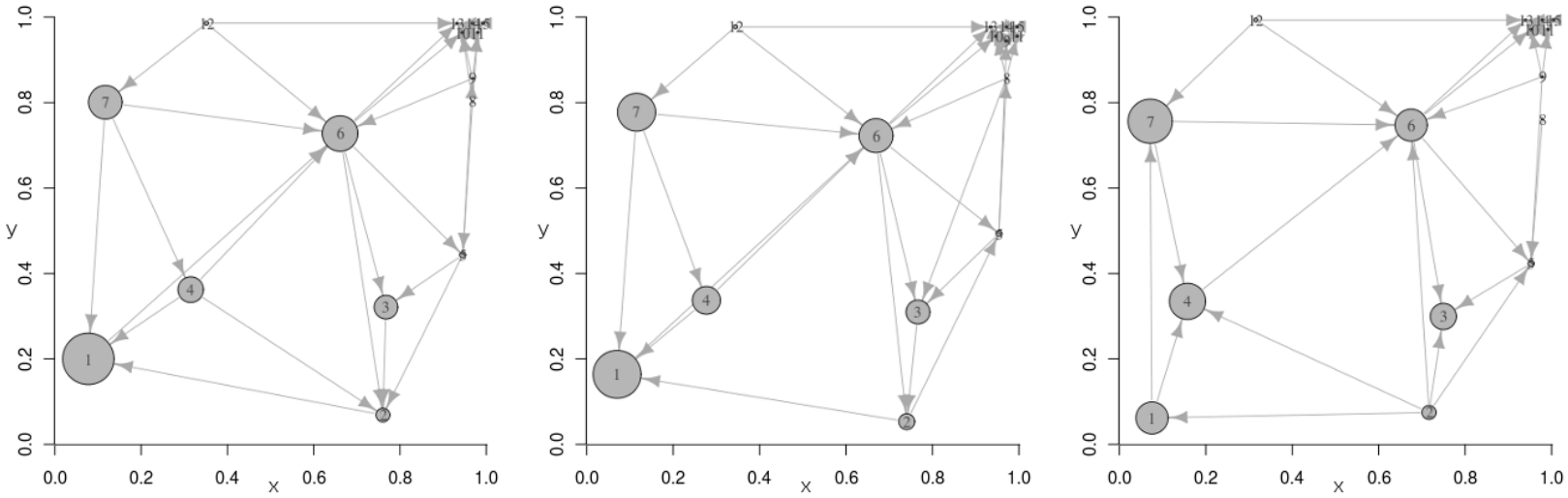
The algorithm is applied on the lid-driven cavity scenario to examine what kind of additional information can be extracted by observing the graph's evolution. In particular, we discuss three different settings: (a) the evolution of the flow graph constructed for the identical vector field but with varying numbers of nodes, (b) the evolution of the flow graphs for the complete sequence of vector fields generated by one simulation and (c) the evolution of the flow graphs for the same simulation, but with varying input parameters.

The evaluation will be performed on the data sets generated previously with focus on the lid-driven cavity case. This simulation is more stable against changes in the input parameters. While small changes in the fluid's viscosity result in a chaotic flow in the backward step case, the flow in the lid-driven cavity case changes consistently. This is crucial to understanding and interpreting the measured similarities and differences of the flow graphs, in particular if with respect to the assumption of small displacements of the nodes between two constructed flow graphs.

(b) Sequence of one simulation

In this particular scenario, the concept of the algorithm is proven. The principle of the proposed algorithm, which compares chronologically close time steps to each other, is found in other domains, too. For instance, in image processing to estimate the optical flow [94]. Therefore the concept of the algorithm is assumed to be consistent. As already shown, the mathematical requirements are fulfilled by the distance function. In the following, the correctness is demonstrated experimentally on the lid-driven cavity data set.

The distance function is changed from application to application and is typically constructed for or adapted to one particular scenario. Because the matching distance (Equation 5.1) is constructed based on the Euclidean distance, the mathematical constraints on a distance are already fulfilled as it has been mentioned previously. In Figure 5.1 the flow graphs and their properties for three time steps are provided. For the sake of simplicity, not all properties are explicitly given and the same cutting level for the hierarchy is chosen. In consequence the matching distance is equivalent to the graph distance. As expected, the requirements are fulfilled by the distance in the experiments, as well.



(a) Time step t=13 (b) Time step t=14 (c) Time step t=15

Figure 5.5: Flow graphs for the lid-driven cavity simulation representing consecutive time steps of the single simulation.

node	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Time step t=13															
x	0.234	0.798	0.804	0.43	0.951	0.716	0.266	0.97	0.97	0.95	0.98	0.46	0.97	0.99	0.99
y	0.261	0.139	0.373	0.411	0.487	0.751	0.818	0.82	0.87	0.97	0.98	0.99	0.99	0.99	0.99
u1	-0.278	-0.737	-0.375	-0.249	-0.373	0.592	0.219	0.003	0.086	2.398	1.462	6.194	6.813	6.451	4.362
u2	0.302	-0.226	-1.111	1.060	-2.156	-0.062	0.328	-2.886	-1.532	-0.841	-1.788	0.059	-0.241	-0.831	-0.562
V%	26	7	12	13	3	18	17	1	1	0	0	2	0	0	0
Time step t=14															
x	0.237	0.782	0.803	0.405	0.956	0.725	0.274	0.97	0.97	0.95	0.99	0.46	0.94	0.97	0.99
y	0.247	0.146	0.38	0.406	0.548	0.757	0.807	0.88	0.96	0.97	0.97	0.99	0.99	0.99	0.99
u1	-0.348	-0.805	-0.353	-0.142	-0.294	0.615	0.247	0.103	1.418	2.402	0.884	6.197	6.815	6.452	4.362
u2	0.311	-0.267	-1.133	1.103	-2.302	-0.110	0.350	-1.521	-2.370	-0.845	-1.462	0.059	-0.241	-0.831	-0.562
V%	24	8	12	14	3	17	19	1	0	0	0	2	0	0	0
Time step t=15															
x	0.275	0.768	0.794	0.338	0.95	0.736	0.272	0.97	0.97	0.95	0.98	0.46	0.94	0.97	0.99
y	0.149	0.161	0.364	0.396	0.475	0.768	0.776	0.78	0.87	0.97	0.97	0.99	0.99	0.99	0.99
u1	-0.534	-0.734	-0.423	-0.112	-0.381	0.625	0.270	-0.023	0.087	2.403	1.462	6.196	6.815	6.451	4.361
u2	0.226	-0.199	-1.106	0.999	-2.131	-0.151	0.385	-2.927	-1.543	-0.846	-1.787	0.059	-0.241	-0.83	-0.561
V%	16	7	13	18	3	16	22	1	1	0	0	2	0	0	0

Table 5.1: Flow graph properties for three consecutive time step: the node's position (x, y), its average flow direction (u_1, u_2) and the ratio of its volume over the complete volume ($V\%$, rounded).

The flow graphs for the time steps are hard to distinguish visually, but the changes can be clearly seen in the graph properties. Furthermore, the differences are precisely described by the numeric values and are attached to the corresponding features. This localizes the changes by the spatial coordinates of features. By combining those two informations, the evolution can be precisely perceived by the user even for high-dimensional data sets, e.g., the interaction and the evolution of nodes $1, 4, 7$. In the three shown time steps, node 4 steadily moves upwards into the space of nodes $1, 7$. At the same time its volume and also the volume of node 7 increases, while the node's volume of 1 is reduced by the same amount. This can be interpreted as the evolution of the primary curl. For this, the more stationary parts of the flow are reduced, e.g., node 1., because they are mobilized by the energy brought into the system. Furthermore, the similarity and the differences between the time steps can be measured precisely, e.g., $gd(t = 13, t = 14) = 0.4$ and $gd(t = 14, t = 15) = 0.5$, i.e., time step $t = 14$ is more similar to time step $t = 13$ than to time step $t = 15$. On the other hand $gd(t = 13, t = 15) = 0.1$, which seems to be incorrect. This finding underlines the necessity of restricting the displacements and the time steps to be small, because the obtained result probably stems from incorrect matching.

In analogy to motion estimation we initially discussed that the displacement of the features between the individual time steps must be small. This restriction is set to solve the correspondence problem uniquely, which is essential to compute an optimal matching. Before the translation of an object between two images can be computed, the corresponding object must be identical in both time steps first. In the case of image processing the consecutive time steps are represented by images, which are formed by a discrete number of densely distributed pixels. The images are compared pixel-wise to find correspondences, and as soon as one pixel of the first image has two correspondences in the second image, the solution is not unique anymore. To solve this ambiguity, the displacement is required to be small enough [94], such that only one correspondence exists, or alternatively an optimization problem is formulated expressing the quality of the matching [72]. The quality is measured by the error caused by incorrect matchings between the two images. As can be seen from this example, the correct solution for the correspondence problem depends on the representation of the input and the objects' translation.

The lid-driven cavity simulation is steadily evolving over time, and in particular, no states re-occur until a stable steady state is reached. As a result, the flow graphs for chronologically close time steps are expected to be more similar than graphs representing graphs of time steps that are further apart in time. Under this assumption we

measure the graph distances between every vector field computed during the simulation. The generated 25 time steps t are compared to each other in a pairwise manner, i.e., $gd(t_i, t_j),\ i, j \in [1; 25]$. The distances between the representing flow graphs are expected to increase steadily with increasing h between the time steps. Therefore, as long as $gd(t_s, t_k) > gd(t_i, t_l),\ k > l$, the computed distances are assumed to be correct. Over the complete interval of 25 time steps, the time difference for which this assumption holds is $|k - l| = h \in [2; 7]$, which was examined in this experiment. Between time steps, in which significant changes take place inside the vector field, e.g., the creation of the secondary curl, the time span is smaller than for time steps close to the steady state. Therefore, the time span for two consecutive vector fields is chosen correctly for this simulation. In simulations in which $|k-l| > 2$ for all time steps, the number of vector fields can be reduced because the evolution is encoded in the flow graph and its evolution. This information can be used after the simulation to reduce the stored data or, if the simulation is re-run, to reduce the produced amount of data by writing fewer vector fields to storage. On the other hand, if $|k-l| < 2$, additional time steps need to be calculated, because the differences between the existing are is too large to portray the evolution correctly. This can efficiently be done, because every comparison is computed in $O(k\ log\ k)$ for the k nodes. This yields a total of $O(k^2)$ for the pairwise comparison, because not the all graphs have to be compared with each other. We omit a quantitative runtime analysis here, because the spent time on the comparison is significantly small in comparison to initial feature extraction, e.g., this previous example took about one second on a standard computer.

(a) Vector field with varying numbers of nodes

After demonstrating the correctness and the efficiency of our proposed algorithm, the computed distances between the flow graphs are now used to extract additional information about the vector field's evolution process. The starting point not only for the moment, but also for other applications, is the hierarchy's optimal cutting level *ocl* (see Definition 4.3). Replacing the user chosen cutting level by the *ocl* eliminates the last user input in the workflow and ensures the best representation of the vector field by its flow graph. In the previous chapter we already presented possible techniques to evaluate the quality of the feature extraction, and by the observation of the evaluation process an additional technique is added.

The evolution of the vector field's flow graph is examined for every time step. Instead of comparing flow graphs of different time steps, the number of nodes, respectively the number of extracted features, for one time step is varied and the similarity

among those graphs is examined. Close to the optimal cutting level, the properties of the flow graph are expected to be stable. Thus, the flow graph constructed for the optimal cutting level (ocl) and its extracted feature is more similar to the flow graphs of the next higher level ($ocl - 1$) or the next lower ($ocl + 1$) level than any other three succeeding levels of the hierarchy. This can be interpreted as finding a local minimum of distances between flow graphs for an identical vector field dependent on the number of extracted features.

The computed optimal cutting level ocl, which is extracted from the evolution, is compared to the technique by Langfelder et al. [88]. The distance in the evolution is converging towards zero for lower levels of the hierarchy, because the overall sizes of the cluster become smaller and smaller and so their differences, too. Therefore, for examining the evaluation of the clustering by the evolution of the flow graphs, a threshold tcl must be set. This threshold is the lowest level that is considered in the search for the ocl.

The presented approach is more complex, though, because every level of the hierarchy the flow graph needs to be compared with the next lower and higher one. This results in $O(n\ log\ n)$ for n extracted features of the tcl hierarchy levels. In the experiments, the algorithm by Langfelder et al. performs in linear time on the extracted features $O(n)$ and faster than our approach. Yet, as discussed previously, it sets the ocl too low. In contrast our presented approach computes $ocl = 13$ and $ocl \in [9; 13]$ for all time steps. The optimal cutting level shown here represents the results better and is applied consequently. After the optimal number of extracted features is approximated automatically, the workflow is now fully un-supervised and the complete functionality can be used to analyze the simulation's vector fields. The user's impact on the results is minimized and guarantees comparable representations of the vector fields by their corresponding flow graphs. We can now start comparing vector fields from different simulations, because this comparison is independent of any user bias and objective. Before two individual vector fields from different simulations are compared, a special case is examined first being of particular interest for the application.

(c) Comparison of different sequences

Typically, the simulation workflow is executed several times while the settings and parameters are slightly changed for every run, e.g., the viscosity of the fluid is modified. To generate the lid-driven cavity data set, which is used for the evaluation, the simulation was executed several times and the viscosity was modified. This was

Reynolds number	3571.43	7142.86	14285.71
evolution coefficient	0.24	0.27	0.28

Table 5.2: Reynolds numbers for three different lid-driven cavity simulations and the computed evolution coefficients, which correlates with the fluid's viscosity set in the simulation.

necessary to obtain at least one sequence of vector fields, in which secondary curls evolve. For each of the generated sequences the evolution coefficient is computed.

Definition 5.3. *Evolution Coefficient*
For the given sequence of vector fields $S(\mathbb{V}) = \{\mathbb{V}_1, ..., \mathbb{V}_T\}$*, which consists of* T *vector fields written for the corresponding time steps of the identical simulation, the evolution coefficient* $ec(S(\mathbb{V}))$ *is defined as the arithmetic mean of the graph distances* $gd(\cdot)$ *between the flow graphs* $FG(\mathbb{V}_t), FG(\mathbb{V}_t + 1)$*, which represent two consecutive time steps* $t, t + 1$

$$ec(S(\mathbb{V})) := \frac{\sum_1^{T-1} gd(FG(\mathbb{V}_t), FG(\mathbb{V}_{t+1}))}{(T-1)} \tag{5.7}$$

This coefficient expresses the changing rate of the flow graphs during the sequence. In the experiments, we could correlate this coefficient with the Reynolds number, which had been defined initially for each simulation. The Reynolds number specifies the fluid's properties, e.g., the viscosity. For simulations in which the fluid is less viscose, the changing rate is higher than for those using a more viscose fluid. In Table 5.2 the Reynolds numbers for three lid-driven cavity simulations are given and the computed evolution coefficients. The Reynolds number does not directly correlate with the evolution coefficient. However, for larger Reynolds numbers that indicate a less viscose fluid, the evolution coefficient is larger, i.e., $Re \propto ec$. This is the anticipated result, because the more fluid the medium is the larger are the changes between the single flow graphs are expected. Examining the proportion between both values could be subject of further research, because the concept seems promising.

5.4.2 Graph Comparison

The evolution of graphs examines the similarities and differences of flow graphs, which were computed for identical reference frames. This is the case for vector fields computed by the same simulation. Nevertheless, the derived graph evolution could also be applied to compare vector fields defined in different frames but of the same type.

	generic backward step	backward step	lid-driven cavity
generic backward step	0	1.2	2.9
backward step	1.2	0	2.7
lid-driven cavity	2.9	2.7	0

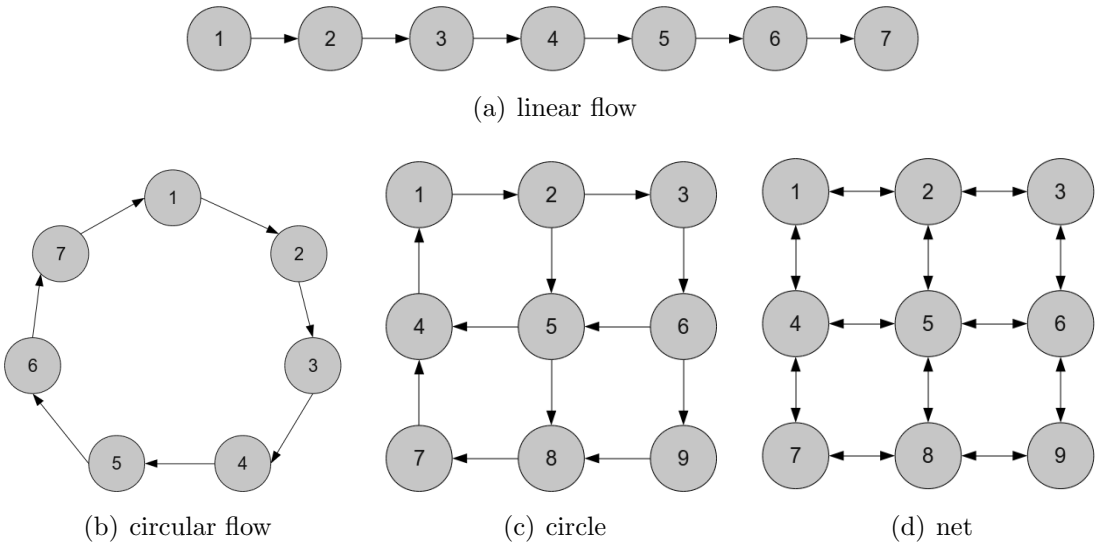
Table 5.3: Graph distances computed by the algorithm of Umeyama [135]. The flow graphs are formed by the optimal number of nodes computed previously (generic backward step (4), backward step (7) and lid-driven cavity (13)).

	generic backward step	backward step	lid-driven cavity
generic backward step	0	1	2.9
backward step	1	0	3.3
lid-driven cavity	2.9	3.3	0

Table 5.4: Graph distances computed by the algorithm of Singh et al. [127]. The flow graphs are formed by the optimal number of nodes computed previously (generic backward step (4), backward step (7) and lid-driven cavity (13)).

The derived evolution coefficient indicates the changing rate and can be used to compare the simulations with each other. The setting for the comparison of graphs is different than for the previous evolution of graphs. For the comparison only two vector fields are needed represented by their flow graphs. Because the comparison is only based on the graphs' structure, two spectral algorithms have been chosen. The algorithm of Umeyama [135] and Singh et al. [127] belong to the spectral graph matching techniques and will be used in two modifications here, the classic and the weighted version.

Starting points for the experiments are the data sets introduced previously, see Table 5.3 and Table 5.4. The comparison is solely based on the adjacency matrix of the graphs without further information, e.g., labels. Obviously, the distances between identical flow graphs is zero and the computed distances are symmetric. The number of nodes for the flow graphs representing the vector field were computed as described in the previous section and are the optimal number for the representation. For both algorithms, the generic backward step data set is more similar to the backward step than to the lid-driven cavity data set. In the concluding experiment the additional information of the feature volumes was included by adding the weighting matrix W. The computed graph distances are the same as before. As a result, the computed matchings are correct, because otherwise by adding the weighting matrix, the matching would haven be shifted by favoring matchings of nodes with large values of w_{ij}.



(a) linear flow

(b) circular flow (c) circle (d) net

Figure 5.6: Prototype graphs used to categorize the flow graphs and describe their main flow direction.

	linear flow	circular flow	circle	net
backward step	1.1	1.4	1.8	2.3
lid-driven cavity	3.8	3.7	3.5	3.8

Table 5.5: Graph distances computed by the algorithm of Umeyama [135]. The flow graphs are formed by the optimal number of nodes computed previously (backward step (7) and lid-driven cavity (13)) and are compared to pre-defined graphs.

In the experiments, the similarity between the generic and real backward step was verified once more to show the merits of the graph comparison. It is applied in the following to extract additional properties of the vector field. We categorize those and their flow graphs into three classes as shown in Figure 5.6. They either expose a linear flow from one side to the other, a circular flow or a net-like connections between the nodes.

The flow graphs of the backward step and the lid-driven cavity are now compared to those constructed graphs to judge, what their major flow property presents itself. The linear and the circular flow are generated automatically to have the same number of nodes as the graphs they are compared to, i.e., seven for the backward step's flow graph and 13 for the lid-driven cavity's flow graph. The other two graphs are formed by nine nodes each. The results of the comparison in Table 5.5 indicate that the backward step can actually be categorized as linear flow, although a curl is formed at the backward step. The flow graph of the lid-driven cavity cannot be categorized as clearly as the backward step's graph. However, it is more similar to a circular flow

time step	1	2	3	4	5	6	7	8	9	10	11	12	13
linear flow	2.5	3.5	3.0	2.9	3.2	3.4	3.5	3.3	3.3	2.7	2.9	2.2	3.1
circular flow	2.2	3.4	2.9	2.8	3.1	3.1	3.0	3.2	3.2	2.8	2.8	2.1	3.0

time step	14	15	16	17	18	19	20	21	22	23	24	25
linear flow	2.5	3.3	3.3	3.0	3.6	3.1	3.2	3.3	3.0	3.2	3.4	3.8
circular flow	2.4	3.0	3.1	2.7	3.5	3.0	2.9	3.0	2.7	3.0	3.1	3.7

Table 5.6: Graph distances computed by the algorithm of Umeyama [135]. The flow graph for every time step of the simulation is compared to a linear and a circular one.

than a linear flow or a simple net. This is also clarified by comparison to the second circle graph, which is constructed to be similar to the lid-driven cavity's flow graph. Furthermore, the graph comparison also verifies the proposed graph construction process. The constructed flow graphs may look like a network, in which the nodes are connected to all their neighbors. Indeed, the flow graphs convey the distinct behavior of the vector field as demonstrated by the comparison to representatives of different classes.

The comparison with prototypical graphs can also be applied to express the structural evolution of the flow graphs over the complete time interval of the simulation. In Table 5.6, the extracted flow graph for every individual time step of the simulation is compared to a linear and a circular flow graph. The similarity is again measured by the graph distance and expresses the differences between them. The measured distances are increasing steadily, which indicates the increasing complexity of the lid-driven cavity's flow graph. This graph becomes more and more complex and consequently less similar to the prototypical graphs. Over the complete time interval, the constructed flow graphs expose more of a generally circular overall appearance than a linear one. As a result, the lid-driven cavity can be interpreted as a circular flow loosing this circular appearance over the time interval while the graphs' complexity increases.

The algorithm of Umeyama [135] has a cubic runtime $O(n^3)$, where n is the number of nodes. Normally, this is considered to be too slow for application. Nevertheless, we can utilized this algorithm in our context, because the graphs are quite small and the majority of computation time is spent by the feature extraction. Even for flow graphs of the lid-driven cavity simulation consisting of many nodes ($n = 30$), the execution time was less than a second.

The comparison to prototypical graphs can be seen as the intermediate result towards examining scenario III, which is the graph simplification by identifying iden-

tical sub-structures. Instead of extracting such structures first and comparing them to the original graph to find re-occurrences, we initially defined these structures and compare them to the graph.

5.5 Summary and Discussion

In this chapter, vector fields were compared via their flow graphs. We distinguished between different scenarios, the graph evolution (scenario I) and the graph comparison (scenario II). While the graph evolution describes the development of the flow graph over time and extracts information on the initial simulation's parameters, the graph comparison matches the structure of any two flow graphs. Both methods are applied to extract further information on the vector fields and the simulation, which has been impossible by using traditional techniques.

- The description of evolution was used to approximate an optimal cutting level of the hierarchy *ocl*. For this level, the balance between simplification and resolution of the initial vector field is optimal. Furthermore, the workflow can now be executed completely un-supervised, because the very last user specified parameter, the hierarchy's cutting level, is replaced, as well.

- The evolution of the flow graphs gives insight on the development of the vector field over time. The changes between two consecutive graphs are not only highlighted visually, but the changes are also described precisely. The user is now able to analyze, in which particular nodes or edges the changes occur and how large they are.

- The complete simulation can be analyzed automatically and efficiently. This includes the analysis of the sequence of generated vector fields, which can be described by the evolution coefficient expressing the changing rate of the simulation's time steps. It can either be used to compare simulations of the same type to each other or to draw conclusions on the initial parameter set of the simulation, e.g., the fluid's viscosity.

- The analysis of the distances between the extracted flow graphs allows from a step-width control for the simulation. This aspect is more of theoretical importance, because the step-width is normally controlled directly during the simulation. However, the gained knowledge can be used to sample the sequence of vector fields appropriately. For two time steps, for which the representing

flow graphs are similar, one can be omitted to reduce the stored data value and to focus on significant time steps.

- The comparison of graphs is used to distinguish to which class the simulation belongs. We compared prototypical graphs in order to the extracted flow graphs to distinguish whether the flow is mainly circular or linear. This can form the basis for scenario III, i.e., matching of repetitive sub-structures inside a flow graph. For now, this scenario has been simplified by omitting the detection of those structures.

Chapter 6

Conclusions and Future Work

Simulations are steadily replacing real world experiments in almost every application in science and industry. With the increasing performance of every new computer generation the simulations are becoming more and more complex. While the generated vector fields include more data points and dimensions than several years ago, the analysis of the simulated data still relies on traditional visualization techniques. Those are limited by the human visual system and biased easily by the user defining them. Because for every data set the visualization pipeline must be manufactured individually, this visual analysis is the bottleneck in the simulation workflow. In the presented work we overcome this limitations by removing the human influence from the workflow. The result is an abstract and comprehensive graph representation, which is extracted and constructed un-supervised. The newly developed flow graphs allow for a quantitative analysis and the comparison of vector fields by their corresponding graphs.

6.1 Summary

In this thesis, the visualization in the classic simulation workflow has been replaced to improve the throughput of the workflow and to reduce the user's impact on the results. In the classic workflow the iterative loop between the user and the visualization is the key knowledge extraction step. This loop requires a lot of user input to be executed and consequently suffers from a significant user impact on its resulting visualization. By replacing the visualization with an automated feature extraction, this loop is broken down. The proposed analysis workflow is based on clustering of the input data to obtain groups of similar data points. This process relies solely on statistics and the obtained results are not dependent on any user input. Furthermore,

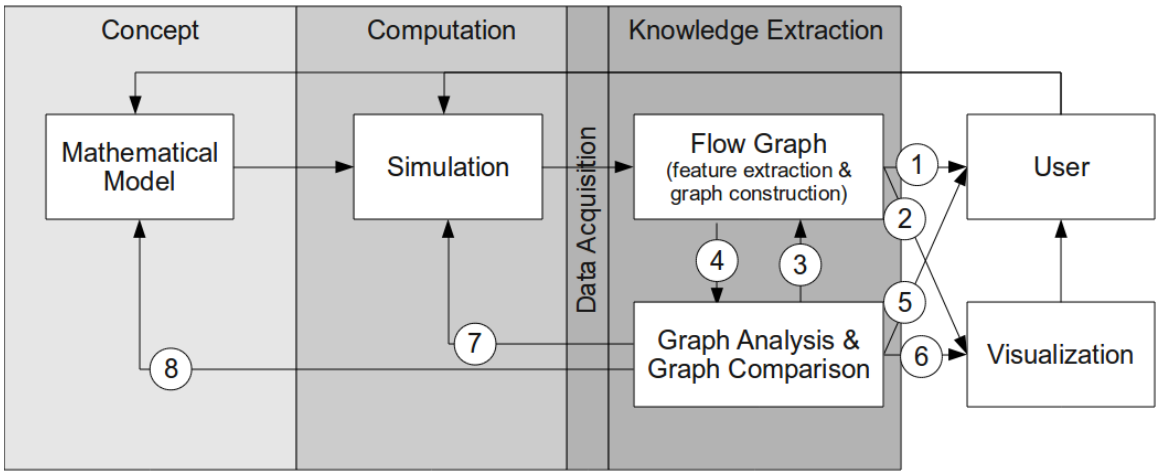


Figure 6.1: Our novel workflow. The labels indicate the issues elaborated in this thesis. 1: Flow graphs describe the vector fields in a comprehensive manner. 2: Flow graphs enable an abstract representation and and a simplified visualization. 3: The optimal cutting level of the hierarchy is found by a similarity of the flow graphs for a single vector field exposing different numbers of extracted features. 4: The uniform description enables a quantitative analysis. 5: Vector fields can be categorized with their flow graphs into prototypical classes, e.g., those of linear or circular flow. 6: The difference between two flow graphs for consecutive time steps can be visualized and described precisely. 7: The changing rate between the flow graphs of subsequent time steps can be exploited to control the step-width of the simulation. 8: Simulations of the same type are distinguished by the evolution coefficient describing their basic properties.

geometric flow graphs have been introduced to represent vector fields using the extracted features and their relation to each other. Our presented flow graphs describe the vector field in a uniform way. The vector fields can be compared in order to evaluate the clustering, to understand the vector field's evolution over time and to identify differences and similarities between several simulations. The newly developed workflow is summarized in Figure 6.1 and will be discussed in the following.

The replacement of visualization by clustering within the workflow demanded a novel definition of features in vector fields. While they are traditionally defined as points of interest, clustering algorithms extract regions of interest. The smallest possible features in our context are individual cells, thus the cell's feature vector was introduced to describe the properties and the behavior of the cells or the features, respectively. To extract the features according to our definition, an optimal clustering algorithm was chosen.

For the extracted features the geometric flow graph was introduced, which represents the vector field. The flow graphs display the features and their relation and

can be used as an abstract visualization (Figure 6.1 item 2). Although this is built on very few entities, and consequently does not suffer from many disadvantages of classic visualization techniques, the major improvement achieved by the flow graphs is the uniform and comprehensive description of vector fields (Figure 6.1 item 1). This description enables a quantitative analysis of the vector field by using the flow graph and its components (Figure 6.1 item 4).

Furthermore, the graph representation links the domains of visualization and graph theory. Techniques for graph similarity and matching are now applicable to vector fields and to their flow graphs, as well. This novel connection is used to evaluate the quality of clustering as feature extraction (Figure 6.1 item 3) in order to examine the evolution of vector fields over the complete time interval of the simulation (Figure 6.1 item 8) and to compare vector fields of different simulations (Figure 6.1 item 5). Indeed, evolution and comparison can be described precisely by the extracted evolution coefficient (Figure 6.1 item 7) and the similarity to prototypical graphs (Figure 6.1 item 6), respectively. All these scenarios could only be examined because the traditional simulation workflow is modified and the components of visualization are replaced. The novel workflow is run independent from the user, which enables a higher throughput and the processing of complete sequences of vector fields, the same way they are generated by simulations. This provides the automatic control of the step-width (Figure 6.1 item 7).

Throughout this thesis, the presented techniques were applied to different data sets to validate them and to demonstrate their effectiveness and efficiency. The data sets were selected from flow simulations and are well described in theory. The feature extraction was compared to established visualization techniques, because they share comparable approaches. Yet, for the graph comparison a proof of concept was provided to validate its correctness.

6.2 Future Work

There are still open issues of course, for example, how the presented approach could be improved or extended. We mention five of them here. The first is a technical issue and discusses the shortening of runtime by parallelizing our approach. The next issue focuses on the usability and shows how the feature extraction and graph comparison can be made available for a larger community. Finally, the last three issues discuss the extension to vector fields from other origins and the extension of the graph comparison.

- **Parallelization.** The construction of the flow graphs for a sequence of time steps can be parallelized easily. In contrast to classic visualization techniques, which requiring user input, our proposed feature extraction and graph construction is executed unsupervised. Thus, the complete feature extraction and the graph construction for every individual time step can be performed independently on individual computers. The results must only be merged, if a final comparison of the series of flow graphs is desired. The construction of one flow graph can also be parallelized, especially the costly feature extraction process and the included pre-processing. Because the cells are pre-processed independently of each other, this step can be parallelized easily, as well. For the computation of divergence and rotation, the parallelization requires an allocation of the grid's cells on different computers with a little overhead of some duplicated cells. For instance, the grid can be segmented into blocks of equal size and processed individually. Only the borders of the block need to be stored twice, to ensure that all neighboring cells are found for every cell inside the block.

- **Improved visualization.** During the evaluation of graph construction we focused our approach on the description instead of the visualization of the vector field, already. Without implementation into a visualization toolkit, the display of the flow graphs can be improved easily by superior virtual placing of the nodes. This has been discussed for the backward step's flow graph. Furthermore, the proposed techniques are constructed, such that it requires no user interaction. Consequently, the major focus of its design was functionality and not usability. Visualization toolkits, e.g., VisTrails or ParaView, unite many different visualization techniques and make them available for a large community of users. They are designed to be easy to learn and to be utilized by users with little knowledge in visualization. Those profit from an abstract representation of the input data the most. To them, the implementation into a visualization toolkit makes our approach available. In this environment, the vector field's flow graph provides an overview and helps the user during data exploration. In a second step, the user can combine the flow graph with other techniques to build a visualization focused on specific aspects.

- **Evolution coefficient.** The evolution coefficient was introduced to describe the changing rate of the flow graphs for sequences of vector fields computed by a single simulation. So far, we have proven the impact of the viscosity on

evolution by the coefficient. This was just a basic example. However, we believe that more information on the vector fields, their representing flow graphs and their evolution can be extracted and described by the evolution coefficient.

- **Finding similar structures inside the flow graph.** In the section about the comparison of graphs, different scenarios were discussed. We distinguished the cases by the origin of vector fields. The most challenging scenario, of finding similar structures inside the flow graph, was omitted and we refer to the work of Armiti [14]. There, the matching and the similarity of geometric graphs is discussed in detail. In general, the developed techniques are applicable to our scenario. They are constructed for 2D graphs only, but could be extended to 3D graphs as well.

- **Extension to other types of vector fields.** For the evaluation we used linear vector fields only. Non-linear vector fields can be described by modifying the definition of the feature vector and including feature properties, which are specific to non-linear vector fields, e.g., shocks. Depending on the construction of the vector field's grid, the computation of the rotation and the divergence must be changed. If the vector field is not linear for a cell, their values cannot be computed by the integral. In the more likely case of the grid being aligned with non-linearity, only the interpolation of the functional values for the path integral must be modified. The focus of this thesis is on flow fields. Vector fields represent other fields, too, e.g., magnetic fields. Again, the definition of the feature vector can be modified to include the specific features of these scenarios as well.

Bibliography

[1] Delta wing visualization. http://www.eng.utah.edu. Accessed: 2012-09-16.

[2] European weather forecast. http://www.wetter.net/kontinent/europa.html. Accessed: 2011-07-06.

[3] Formula 1 season 2010. http://www.formula1.com/results/team/2010/. Accessed: 2014-12-14.

[4] Indian gp: Why ferrari are struggling. http://www.bbc.com/sport/0/formula1/20091800. Accessed: 2015-06-02.

[5] Mercedes-benz passion blog. http://blog.mercedes-benz-passion.com. Accessed: 2012-09-16.

[6] Spline interpolation. https://en.wikipedia.org/wiki/Spline/_interpolation. Accessed: 2015-03-15.

[7] Vistrails documentation. http://www.vistrails.org/usersguide/v2.2/html/ VisTrails.pdf. Accessed: 2015-05-16.

[8] Wind channel visualization. http://www.mech.kth.se. Accessed: 2012-09-16.

[9] Elke Achtert, Christian Bohm, Peer Kroger, and Arthur Zimek. Mining hierarchies of correlation clusters. In *Scientific and Statistical Database Management, 2006. 18th International Conference on*, pages 119–128. IEEE, 2006.

[10] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. *Automatic subspace clustering of high dimensional data for data mining applications*, volume 27. ACM, 1998.

[11] Mark S Aldenderfer and Roger K Blashfield. Cluster analysis. sage university paper series on quantitative applications in the social sciences 07-044. 1984.

[12] Herbert Amann, Joachim Escher, Silvio Levy, and Matthew Cargo. *Analysis III*. Springer, 2009.

[13] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: Ordering points to identify the clustering structure. In *ACM SIGMOD Record*, volume 28, pages 49–60. ACM, 1999.

[14] Ayser Armiti. *Geometric graphs*. PhD thesis, Heidelberg, Univ., Diss., 2015, 2015. Zs.fassung in engl. und dt. Sprache.

[15] Chris Bailey-Kellogg and Feng Zhao. Qualitative spatial reasoning extracting and reasoning with spatial aggregates. *AI Magazine*, 24(4):47, 2003.

[16] Egon Balas and Chang Sung Yu. Finding a maximum clique in an arbitrary graph. *SIAM Journal on Computing*, 15(4):1054–1068, 1986.

[17] Rajesh Batra and Lambertus Hesselink. Feature comparisons of 3-d vector fields using earth mover's distance. In *Proceedings of the conference on Visualization'99: celebrating ten years*, pages 105–114. IEEE Computer Society Press, 1999.

[18] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is nearest neighbor meaningful? In *Database TheoryICDT99*, pages 217–235. Springer, 1999.

[19] James C Bezdek and Nikhil R Pal. Some new indexes of cluster validity. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 28(3):301–315, 1998.

[20] Vincent D Blondel, Anahí Gajardo, Maureen Heymans, Pierre Senellart, and Paul Van Dooren. A measure of similarity between graph vertices: Applications to synonym extraction and web searching. *SIAM review*, 46(4):647–666, 2004.

[21] Rita Borgo, Johannes Kehrer, David HS Chung, Eamonn Maguire, Robert S Laramee, Helwig Hauser, Matthew Ward, and Min Chen. Glyph-based visualization: Foundations, design guidelines, techniques and applications. *Eurographics State of the Art Reports*, pages 39–63, 2013.

[22] E Brandt Heiberg. Automated feature detection in multidimensional images–a unified tensor approach, 2001.

[23] Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[24] Charles-Henri Bruneau and Mazen Saad. The 2d lid-driven cavity problem revisited. *Computers & Fluids*, 35(3):326–348, 2006.

[25] Horst Bunke, Pasquale Foggia, Corrado Guidobaldi, Carlo Sansone, and Mario Vento. A comparison of algorithms for maximum common subgraph on randomly connected graphs. In *Structural, Syntactic, and Statistical Pattern Recognition*, pages 123–132. Springer, 2002.

[26] Tibério S Caetano, Julian John McAuley, Li Cheng, Quoc V Le, and Alexander J Smola. Learning graph matching. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(6):1048–1058, 2009.

[27] Edwin Catmull. A subdivision algorithm for computer display of curved surfaces. Technical report, DTIC Document, 1974.

[28] Sung-Hyuk Cha. Comprehensive survey on distance/similarity measures between probability density functions. *City*, 1(2):1, 2007.

[29] Chun-houh Chen, Wolfgang Karl Härdle, and Antony Unwin. *Handbook of data visualization*. Springer Science & Business Media, 2007.

[30] Guoning Chen, Konstantin Mischaikow, Robert S Laramee, Pawel Pilarczyk, and Eugene Zhang. Vector field editing and periodic orbit extraction using morse decomposition. *Visualization and Computer Graphics, IEEE Transactions on*, 13(4):769–785, 2007.

[31] Hung-Hsuan Chen, Liang Gou, Xiaolong Luke Zhang, and C Lee Giles. Discovering missing links in networks using vertex similarity measures. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 138–143. ACM, 2012.

[32] Otfried Cheong, Joachim Gudmundsson, Hyo-Sil Kim, Daria Schymura, and Fabian Stehn. Measuring the similarity of geometric graphs. In *Experimental Algorithms*, pages 101–112. Springer, 2009.

[33] TP Chiang, WH Sheu, and Robert R Hwang. Effect of reynolds number on the eddy structure in a lid-driven cavity. *International journal for numerical methods in fluids*, 26(5):557–579, 1998.

[34] Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.

[35] Sam Collins. Wirth research breaks new ground. *Racecar Engineering (IPC Media)*, 2010.

[36] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. Thirty years of graph matching in pattern recognition. *International journal of pattern recognition and artificial intelligence*, 18(03):265–298, 2004.

[37] Luigi Pietro Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. An improved algorithm for matching large graphs. In *3rd IAPR-TC15 workshop on graph-based representations in pattern recognition*, pages 149–159, 2001.

[38] Richard Courant and Kurt Otto Friedrichs. *Supersonic flow and shock waves*, volume 21. Springer, 1976.

[39] Richard Courant and Herbert Robbins. *What is Mathematics?: an elementary approach to ideas and methods*. Oxford University Press, 1996.

[40] Daniel Defays. An efficient algorithm for a complete link method. *The Computer Journal*, 20(4):364–366, 1977.

[41] Arthur P Dempster, Nan M Laird, Donald B Rubin, et al. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal statistical Society*, 39(1):1–38, 1977.

[42] Mukund Deshpande, Michihiro Kuramochi, Nikil Wale, and George Karypis. Frequent substructure-based approaches for classifying chemical compounds. *Knowledge and Data Engineering, IEEE Transactions on*, 17(8):1036–1050, 2005.

[43] Reinhard Diestel. *Graphentheory*. Springer, 2000.

[44] Richard O Duda, Peter E Hart, et al. *Pattern classification and scene analysis*, volume 3. Wiley New York, 1973.

[45] Joseph C Dunn. Well-separated clusters and optimal fuzzy partitions. *Journal of cybernetics*, 4(1):95–104, 1974.

[46] Julia Ebling and Gerik Scheuermann. Clifford convolution and pattern matching on vector fields. In *Visualization, 2003. VIS 2003. IEEE*, pages 193–200. IEEE, 2003.

[47] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, volume 96, pages 226–231, 1996.

[48] Vladimir Estivill-Castro. Why so many clustering algorithms: a position paper. *ACM SIGKDD Explorations Newsletter*, 4(1):65–75, 2002.

[49] Tom E Faber. *Fluid dynamics for physicists*. Cambridge University Press, 1995.

[50] Gregory Falkovich. *Fluid mechanics: A short course for physicists*. Cambridge University Press, 2011.

[51] Usama Fayyad, David Haussler, and Paul Stolorz. Mining scientific data. *Communications of the ACM*, 39(11):51–57, 1996.

[52] Otto Forster. Analysis 3. *Vieweg, Braunschweig*, 1996.

[53] Z Fu, A De Pennington, and A Saia. A graph grammar approach to feature representation and transformation. *International Journal of Computer Integrated Manufacturing*, 6(1-2):137–151, 1993.

[54] Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis and applications*, 13(1):113–129, 2010.

[55] Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman, 2002.

[56] Paul L George. *Automatic mesh generation: applications to finite element methods*. John Wiley & Sons, Inc., 1992.

[57] Steven Gold and Anand Rangarajan. A graduated assignment algorithm for graph matching. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 18(4):377–388, 1996.

[58] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

[59] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: an efficient clustering algorithm for large databases. In *ACM SIGMOD Record*, volume 27, pages 73–84. ACM, 1998.

[60] Qinghong Guo, Mrinal K Mandal, and Micheal Y Li. Efficient hodge–helmholtz decomposition of motion fields. *Pattern Recognition Letters*, 26(4):493–501, 2005.

[61] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

[62] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17(2-3):107–145, 2001.

[63] J Han, M Kamber, and J Pei. Data mining: concepts and techniques. *Morgan Kaufmann series in data management systems (*, 2012.

[64] Charles D Hansen and Chris R Johnson. *The visualization handbook*. Academic Press, 2005.

[65] Helwig Hauser, Robert S Laramee, Helmut Doleisch, Frits H Post, and Benjamin Vrolijk. The state of the art in flow visualization, part 1: Direct, texture-based and geometric techniques. In *Proceedings IEEE Visualization*, pages 19–26, 2003.

[66] Ji He, Ah-Hwee Tan, Chew-Lim Tan, and Sam-Yuan Sung. On quantitative evaluation of clustering systems. In *Clustering and information retrieval*, pages 105–133. Springer, 2004.

[67] Bjoern Heckel, Gunther Weber, Bernd Hamann, and Kenneth I Joy. Construction of vector field hierarchies. In *Proceedings of the conference on Visualization'99: celebrating ten years*, pages 19–25. IEEE Computer Society Press, 1999.

[68] James Helman and Lambertus Hesselink. Representation and display of vector field topology in fluid flow data sets. *Computer*, 22(8):27–36, 1989.

[69] H Helmholtz. Ueber integrale der hydrodynamischen gleichungen, welche den wirbelbewegungen entsprechen, crelles j. 55, 25 (1858).

[70] Harro Heuser. Lehrbuch der analysis, teil 2. *BG Teubner, Stuttgart*, 6, 1991.

[71] Alexander Hinneburg and Daniel A Keim. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. 1999.

[72] Berthold K Horn and Brian G Schunck. Determining optical flow. In *1981 Technical symposium east*, pages 319–331. International Society for Optics and Photonics, 1981.

[73] Eduardo R Hruschka, Ricardo JGB Campello, and Leandro N de Castro. Improving the efficiency of a clustering genetic algorithm. In *Advances in Artificial Intelligence–IBERAMIA 2004*, pages 861–870. Springer, 2004.

[74] Ming Huang and Fuling Bian. A grid and density based fast spatial clustering algorithm. In *Artificial Intelligence and Computational Intelligence, 2009. AICI'09. International Conference on*, volume 4, pages 260–263. IEEE, 2009.

[75] Benoit Huet and Edwin R Hancock. Inexact graph retrieval. In *Content-Based Access of Image and Video Libraries, 1999.(CBAIVL'99) Proceedings. IEEE Workshop on*, pages 40–44. IEEE, 1999.

[76] Anette Hulth. Improved automatic keyword extraction given more linguistic knowledge. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pages 216–223. Association for Computational Linguistics, 2003.

[77] Bernd Jähne. *Digitale Bildverarbeitung*. Springer-Verlag, 2013.

[78] Ming Jiang and Raghu Adviser-Machiraju. *A feature-based approach to visualizing and mining simulation data*. Ohio State University, 2005.

[79] Ming Jiang, Tat-Sang Choy, Sameep Mehta, Matt Coatney, Steve Barr, Kaden Hazzard, David Richie, Srinivasan Parthasarathy, Raghu Machiraju, David Thompson, et al. Feature mining paradigms for scientific data. In *SDM*. SIAM, 2003.

[80] Leonard Kaufman and Peter Rousseeuw. *Clustering by means of medoids*. North-Holland, 1987.

[81] Johannes Kehrer and Helwig Hauser. Visualization and visual analysis of multifaceted scientific data: A survey. *Visualization and Computer Graphics, IEEE Transactions on*, 19(3):495–513, 2013.

[82] Duck Hoon Kim, Il Dong Yun, and Sang Uk Lee. Attributed relational graph matching based on the nested assignment structure. *Pattern Recognition*, 43(3):914–928, 2010.

[83] Robert M Kirby and Cláudio T Silva. The need for verifiable visualization. *Computer Graphics and Applications, IEEE*, 28(5):78–83, 2008.

[84] Hardy Kremer, Stephan Günnemann, Simon Wollwage, and Thomas Seidl. Nesting the earth mover's distance for effective cluster tracing. In *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*, page 34. ACM, 2013.

[85] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(1):1, 2009.

[86] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[87] Christian B Lang and Norbert Pucker. *Mathematische Methoden in der Physik*. Spektrum, Akad. Verlag, 1998.

[88] Peter Langfelder, Bin Zhang, and Steve Horvath. Defining clusters from a hierarchical cluster tree: the dynamic tree cut package for r. *Bioinformatics*, 24(5):719–720, 2008.

[89] Robert S Laramee, Helwig Hauser, Helmut Doleisch, Benjamin Vrolijk, Frits H Post, and Daniel Weiskopf. The state of the art in flow visualization: Dense and texture-based techniques. In *Computer Graphics Forum*, volume 23, pages 203–221. Wiley Online Library, 2004.

[90] Yingmei Lavin, Rajesh Batra, and Lambertus Hesselink. Feature comparisons of vector fields using earth mover's distance. In *Visualization'98. Proceedings*, pages 103–109. IEEE, 1998.

[91] Richard S. Lindzen. There is no 'consensus' on global warming. *Wall Street Journal*, 2006.

[92] Stuart Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982.

[93] David G Lowe. Three-dimensional object recognition from single two-dimensional images. *Artificial intelligence*, 31(3):355–395, 1987.

[94] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, pages 674–679, 1981.

[95] Hans J Lugt. Vortex flow in nature and technology. *New York, Wiley-Interscience, 1983, 305 p. Translation.*, 1, 1983.

[96] Bin Luo and Edwin R. Hancock. Structural graph matching using the em algorithm and singular value decomposition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(10):1120–1136, 2001.

[97] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, page 14. California, USA, 1967.

[98] James J McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Software: Practice and Experience*, 12(1):23–34, 1982.

[99] John Willard Milnor. *Topology from the differentiable viewpoint*. Princeton University Press, 1997.

[100] Gordon E Moore et al. Cramming more components onto integrated circuits, 1965.

[101] Walter G O'Dell, Christopher C Moore, William C Hunter, Elias A Zerhouni, and Elliot R McVeigh. Three-dimensional myocardial deformations: calculation with displacement field fitting to tagged mr images. *Radiology*, 195(3):829–835, 1995.

[102] Steven J Owen. A survey of unstructured mesh generation technology. In *IMR*, pages 239–267, 1998.

[103] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[104] Odysseas Papapetrou, Ekaterini Ioannou, and Dimitrios Skoutas. Efficient discovery of frequent subgraph patterns in uncertain graph databases. In *Proceedings of the 14th International Conference on Extending Database Technology*, pages 355–366. ACM, 2011.

[105] Lance Parsons, Ehtesham Haque, and Huan Liu. Subspace clustering for high dimensional data: a review. *ACM SIGKDD Explorations Newsletter*, 6(1):90–105, 2004.

[106] Konrad Polthier and Eike Preuss. *Variational approach to vector field decomposition*. Springer, 2000.

[107] Konrad Polthier and Eike Preuss. Identifying vector field singularities using a discrete hodge decomposition. In *Visualization and Mathematics III*, pages 113–134. Springer, 2003.

[108] Jake Porway, Benjamin Yao, and Song Chun Zhu. Learning compositional models for object categories from small sample sets. *Object categorization: computer and human vision perspectives*, 2008.

[109] Frits H Post, Benjamin Vrolijk, Helwig Hauser, Robert S Laramee, and Helmut Doleisch. The state of the art in flow visualisation: Feature extraction and tracking. In *Computer Graphics Forum*, volume 22, pages 775–792. Wiley Online Library, 2003.

[110] Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. A graph matching method and a graph matching distance based on subgraph assignments. *Pattern Recognition Letters*, 31(5):394–406, 2010.

[111] Ronald C Read and Derek G Corneil. The graph isomorphism disease. *Journal of Graph Theory*, 1(4):339–363, 1977.

[112] Kaspar Riesen and Horst Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Computing*, 27(7):950–959, 2009.

[113] Kaspar Riesen and Horst Bunke. *Graph classification and clustering based on vector space embedding*. World Scientific Publishing Co., Inc., 2010.

[114] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[115] Filip Sadlo and Ronald Peikert. Visualizing lagrangian coherent structures and comparison to vector field topology. In *Topology-Based Methods in Visualization II*, pages 15–29. Springer, 2009.

[116] Tobias Salzbrunn, Heike Jänicke, Thomas Wischgoll, and Gerik Scheuermann. The state of the art in flow visualization: Partition-based techniques. In *SimVis*, pages 75–92, 2008.

[117] Ravi Samtaney, Deborah Silver, Norman Zabusky, and Jim Cao. Visualizing features and tracking their evolution. *Computer*, 27(7):20–27, 1994.

[118] Alberto Sanfeliu and King-Sun Fu. A distance measure between attributed relational graphs for pattern recognition. *Systems, Man and Cybernetics, IEEE Transactions on*, (3):353–362, 1983.

[119] Gerik Scheuermann, Hans Hagen, Heinz Krüger, Martin Menzel, and Alyn Rockwood. Visualization of higher order singularities in vector fields. In *Proceedings of the 8th conference on Visualization'97*, pages 67–74. IEEE Computer Society Press, 1997.

[120] Erich Schikuta. Grid-clustering: An efficient hierarchical clustering method for very large data sets. In *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*, volume 2, pages 101–105. IEEE, 1996.

[121] Kuangyu Shi, Holger Theisel, Tino Weinkauf, Hans-Christian Hege, and Hans-Peter Seidel. Visualizing transport structures of time-dependent flow fields. *IEEE computer graphics and applications*, (5):24–36, 2008.

[122] Dongjoe Shin and Tardi Tjahjadi. Similarity invariant delaunay graph matching. In *Structural, Syntactic, and Statistical Pattern Recognition*, pages 25–34. Springer, 2008.

[123] Robin Sibson. Slink: an optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, 1973.

[124] Cláudio T Silva, Erik Anderson, Emanuele Santos, and Juliana Freire. Using vistrails and provenance for teaching scientific visualization. In *Computer Graphics Forum*, volume 30, pages 75–84. Wiley Online Library, 2011.

[125] Deborah Silver and Xin Wang. Volume tracking. In *Visualization'96. Proceedings.*, pages 157–164. IEEE, 1996.

[126] Deborah Silver and Xin Wang. Tracking and visualizing turbulent 3d features. *Visualization and Computer Graphics, IEEE Transactions on*, 3(2):129–141, 1997.

[127] Rohit Singh, Jinbo Xu, and Bonnie Berger. Pairwise global alignment of protein interaction networks by matching neighborhood topology. In *Research in computational molecular biology*, pages 16–31. Springer, 2007.

[128] Alvy Ray Smith. Color gamut transform pairs. In *ACM Siggraph Computer Graphics*, volume 12, pages 12–19. ACM, 1978.

[129] Neil A Thacker, PA Riocreux, and RB Yates. Assessing the completeness properties of pairwise geometric histograms. *Image and Vision Computing*, 13(5):423–429, 1995.

[130] Holger Theisel, Christian Rössl, and Hans-Peter Seidel. Using feature flow fields for topological comparison of vector fields. In *VMV*, pages 521–528, 2003.

[131] Holger Theisel and H-P Seidel. Feature flow fields. In *Proceedings of the symposium on Data visualisation 2003*, pages 141–148. Eurographics Association, 2003.

[132] Holger Theisel, Tino Weinkauf, H-C Hege, and H-P Seidel. Saddle connectors-an approach to visualizing the topological skeleton of complex 3d vector fields. In *Visualization, 2003. VIS 2003. IEEE*, pages 225–232. IEEE, 2003.

[133] Melanie Tory and Torsten Moller. Human factors in visualization research. *Visualization and Computer Graphics, IEEE Transactions on*, 10(1):72–84, 2004.

[134] Julian R Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, 23(1):31–42, 1976.

[135] Shinji Umeyama. An eigendecomposition approach to weighted graph matching problems. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 10(5):695–703, 1988.

[136] Jarke J Van Wijk. Spot noise texture synthesis for data visualization. In *ACM SIGGRAPH Computer Graphics*, volume 25, pages 309–318. ACM, 1991.

[137] Sashikumar Venkataraman, Milind Sohoni, and Vinay Kulkarni. A graph-based framework for feature recognition. In *Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 194–205. ACM, 2001.

[138] Vivek Verma, David Kao, and Alex Pang. A flow-guided streamline seeding strategy. In *Proceedings of the conference on Visualization'00*, pages 163–170. IEEE Computer Society Press, 2000.

[139] Vivek Verma and Alex Pang. Comparative flow visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 10(6):609–624, 2004.

[140] Guoren Wang, Bin Wang, Xiaochun Yang, and Ge Yu. Efficiently indexing large sparse graphs for similarity search. *Knowledge and Data Engineering, IEEE Transactions on*, 24(3):440–451, 2012.

[141] Zhongjie Wang, Janick Martinez Esturo, Hans-Peter Seidel, and Tino Weinkauf. Pattern search in flows based on similarity of stream line segments. In *19th International Workshop on Vision, Modeling and Visualization*, pages 23–30. Eurographics Association, 2014.

[142] Joe H Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.

[143] Tino Weinkauf, Holger Theisel, H-C Hege, and H-P Seidel. Boundary switch connectors for topological visualization of complex 3d vector fields. In *Proceedings of the Sixth Joint Eurographics-IEEE TCVG conference on Visualization*, pages 183–192. Eurographics Association, 2004.

[144] Tino Weinkauf, Holger Theisel, Kuangyu Shi, H-C Hege, and H-P Seidel. Extracting higher order critical points and topological simplification of 3d vector fields. In *Visualization, 2005. VIS 05. IEEE*, pages 559–566. IEEE, 2005.

[145] Tino Weinkauf, Holger Theisel, Allen Van Gelder, and Alex Pang. Stable feature flow fields. *Visualization and Computer Graphics, IEEE Transactions on*, 17(6):770–780, 2011.

[146] C-F Westin, Stephan E Maier, Hatsuho Mamata, Arya Nabavi, Ferenc A Jolesz, and Ron Kikinis. Processing and visualization for diffusion tensor mri. *Medical image analysis*, 6(2):93–108, 2002.

[147] Richard C Wilson and Edwin R Hancock. Structural matching by discrete relaxation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(6):634–648, 1997.

[148] Laurenz Wiskott, J-M Fellous, N Kuiger, and Christoph Von Der Malsburg. Face recognition by elastic bunch graph matching. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):775–779, 1997.

[149] Hui Yang, Srinivasan Parthasarathy, and Sameep Mehta. A generalized framework for mining spatio-temporal patterns in scientific data. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 716–721. ACM, 2005.

[150] Chang Hun You, Lawrence B Holder, and Diane J Cook. Application of graph-based data mining to metabolic pathways. In *Data Mining Workshops, 2006. ICDM Workshops 2006. Sixth IEEE International Conference on*, pages 169–173. IEEE, 2006.

[151] Alan L Yuille, Peter W Hallinan, and David S Cohen. Feature extraction from faces using deformable templates. *International journal of computer vision*, 8(2):99–111, 1992.

[152] Zhiping Zeng, Anthony KH Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. *Proceedings of the VLDB Endowment*, 2(1):25–36, 2009.

[153] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. In *ACM SIGMOD Record*, volume 25, pages 103–114. ACM, 1996.

[154] Yongjie Zhang, Yuri Bazilevs, Samrat Goswami, Chandrajit L Bajaj, and Thomas JR Hughes. Patient-specific vascular nurbs modeling for isogeometric analysis of blood flow. *Computer methods in applied mechanics and engineering*, 196(29):2943–2959, 2007.

[155] Weiguo Zheng, Lei Zou, Xiang Lian, Dong Wang, and Dongyan Zhao. Graph similarity search with edit distance constraint in large graph databases. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 1595–1600. ACM, 2013.

[156] Yuanyuan Zhu, Lu Qin, Jeffrey Xu Yu, Yiping Ke, and Xuemin Lin. High efficiency and quality: large graphs matching. *The VLDB JournalThe International Journal on Very Large Data Bases*, 22(3):345–368, 2013.