Technische Universität München
Fakultät für Informatik
Lehrstuhl für Computer Grafik und Visualisierung

Visualizing the Variability in Ensemble Simulations

Florian Ferstl

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. T. Huckle
Prüfer der Dissertation:	1. Univ.-Prof. Dr. R. Westermann
	2. Univ.-Prof. Dr.-Ing. H. Theisel, Otto-von-Guericke-Universität Magdeburg

Die Dissertation wurde am 28.09.2016 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 24.11.2016 angenommen.

To my family and friends

Abstract

The visualization of numerical simulations that are affected by uncertainty can easily lead to inaccurate or even wrong conclusions. Ensemble simulations account for this problem by performing multiple simulation runs with perturbed input parameters or with different models, allowing us to estimate the uncertainty in a simulation by analyzing the variability in the different outcomes. The introduction of ensembles poses a great challenge for the visualization of simulations since the information of all simulation runs needs to be conveyed simultaneously. Furthermore, the direct coupling of visualizations to simulations is hindered due to substantially longer simulation times. Thus, in order to unfold the full potential of ensemble simulations, new methods for efficient numerical simulation and advanced ensemble visualization techniques need to be developed.

The contributions of this thesis are twofold. In the first part of this thesis, as a possible application for numerical simulation, we present a new method for the efficient simulation of incompressible fluids. We use a finite element discretization on an octree grid to reduce memory requirements by coarsening cells in the interior of the fluid domain, and we employ a special multigrid solver to ensure optimal performance. Despite the irregular grid structure, the resulting method is based on regular computational kernels and is thus easy to parallelize, enabling the simulation of large-scale scenarios and complex domains with a minimum of computational resources.

In the second part of this thesis, we introduce the concept of *variability plots* for the visualization of line based features in ensembles. These are often visualized using so-called spaghetti plots, which are created by simply overlaying all lines of interest in a single view and, thus, are prone to visual clutter. For variability plots, on the other hand, we use statistical analysis and clustering to generate abstract geometric representations that illustrate the major trends and outliers in the data. We present two basic applications of this concept to ensembles of streamlines and iso-contours, and, as a third application, we investigate the visualization of iso-contours in weather forecast ensembles. For this specific use case, we extend variability plots to the time-dependent setting by proposing a special time-hierarchical clustering. We demonstrate the potential of our method in various examples using artificial and meteorological datasets, as well as flow datasets generated with the proposed method for fluid simulation.

Zusammenfassung

Die Visualisierung von numerischen Simulationen, die mit Unsicherheit behaftet sind, kann leicht zu unpräzisen oder falschen Schlussfolgerungen führen. Ensemble Simulationen adressieren dieses Problem, indem mehrere Simulationsläufe mit gestörten Eingabeparametern oder mit verschiedenen Modellen durchgeführt werden, was es uns ermöglicht die Unsicherheit in einer Simulation durch die Analyse der Variabilität in den unterschiedlichen Simulationsergebnissen abzuschätzen. Der Schritt zu Ensembles stellt eine große Herausforderung für die Visualisierung von Simulationen dar, weil die gesamte Information aller Simulationsläufe gleichzeitig vermittelt werden muss. Zusätzlich wird das direkte Koppeln von Visualisierungen an Simulationen erschwert, da sich die Simulationszeiten beträchtlich erhöhen. Um das gesamte Potential von Ensemble Simulationen zu entfalten ist es daher nötig, neue Methoden zur effizienten numerischen Simulation und fortgeschrittene Visualisierungstechniken für Ensembles zu entwickeln.

Die Beiträge dieser Arbeit sind zweigeteilt. Im ersten Teil der Arbeit präsentieren wir eine neue Methode zur effizienten Simulation von inkompressiblen Fluiden als mögliche Anwendung von numerischer Simulation. Wir verwenden eine Finite Elemente Diskretisierung auf einem Octree Gitter um den Arbeitsspeicherverbrauch zu reduzieren, indem wir Zellen im Inneren der Fluiddomäne vergröbern, und wir benutzen einen speziellen Multigrid-Löser um eine möglichst schnelle Simulation zu gewährleisten. Trotz der unregelmäßigen Gitterstruktur baut die resultierende Methode auf regelmäßigen Berechnungsbausteinen auf und kann damit leicht parallelisiert werden, was die Simulation von großen Szenarien und komplexen Domänen mit einem Minimum an Ressourcen ermöglicht.

Im zweiten Teil dieser Arbeit führen wir das Konzept von Variability Plots für die Visualisierung von linienbasierten Merkmalen in Ensembles ein. Diese werden oft durch sogenannte Spaghetti Plots visualisiert. Spaghetti Plots entstehen durch eine einfache Überlagerung aller betrachteten Linien in einem Bild und führen daher leicht zu visueller Übersättigung. Für Variability Plots dagegen verwenden wir eine statistische Analyse und Clustering um abstrakte geometrische Darstellungen zu generieren, die die generellen Trends und Ausreißer in den Daten zeigen. Wir stellen zwei grundlegende Anwendungen dieses Konzepts für Ensembles von Stromlinien und Isokonturen vor, und wir untersuchen als dritte Anwendung die Visualisierung von Isokonturen in Ensemble-Wettervorhersagen. Für diesen speziellen Anwendungsfall erweitern wir Variability Plots auf den zeitabhängigen Fall, in-

dem wir ein besonderes zeit-hierarchisches Clustering vorschlagen. Wir demonstrieren das Potenzial unserer Methode anhand von zahlreichen Beispielen mit künstlichen und meteorologischen Datensätzen, sowie mit Strömungsdatensätzen, die mit der vorgeschlagenen Methode zur Fluidsimulation generiert wurden.

Acknowledgments

I gratefully acknowledge the support of all the people who made this thesis possible. First and foremost, I would like to express my sincere gratitude to my advisor Prof. Dr. Rüdiger Westermann. When i was still an undergraduate student, he provided me with early possibilities to assist with research in interactive flow visualization, which lead to my first publications. Later, he then offered me the possibility to pursue research in the highly interesting fields of fluid simulation and ensemble visualization, which ultimately lead to this thesis. I am very grateful for his guidance, his ideas and our numerous discussions during my time at his chair. His support and time commitment have made an essential contribution to the success of this work.

I also want to thank the co-authors of my papers which have contributed to this thesis, Kai Bürger, Christian Dick, Mathias Kanzler, and Marc Rautenhaus, for contributing their suggestions and ideas, and i want to thank all the other people i have closely collaborated with since my time as an undergraduate student, Ryoichi Ando, Joachim Georgii, Holger Theisel, Nils Thuerey, and Chris Wojtan. I have always enjoyed working with them. Furthermore, I would like to thank my current and former colleagues, Stefan Auer, Boris Bonev, Kai Bürger, Shunting Cao, Matthäus Chajdas, Rachel Chu, Ismail Demir, Christian Dick, Sebastian Eberhardt, Marie-Lena Eckert, Martin Ender, Roland Fraedrich, Mihaela Jarema, Mathias Kanzler, Johannes Kehrer, Alexander Kumpf, Tobias Pfaffelmoser, Marc Rautenhaus, Florian Reichl, Nils Thuerey, Marc Treib, Kiwon Um, and Jun Wu, who have always been available for discussions.

I specially thank Marc Rautenhaus and Andreas Dornbrack for providing access to the ECMWF prediction data in the context of the ECMWF special project “Support Tool for HALO Missions”, and i thank Mahsa Mirzargar, Donald House and their colleagues for providing the hurricane trajectories data and corresponding curve boxplot visualization.

I am enormously thankful to my parents and my friends for giving me all the support that I needed during this time.

This work was funded by the European Union under the ERC Advanced Grant 291372—SaferVis: Uncertainty Visualization for Reliable Data Discovery.

Contents

1 Introduction

Visualization is a powerful tool for presenting complex scientific data in a comprehensible way. Before drawing conclusions, however, we have to respect the fact that scientific data typically originates from sources like, e.g., measurements, human observations or numerical simulations, which are naturally afflicted with errors and uncertainties. Furthermore, the visualization process itself can introduce uncertainty as well. The need of including information about uncertainty in visualizations has been recognized almost twenty years ago [PWL97]. In order to avoid the possibility of drawing inaccurate or wrong conclusions, visualizations have to reveal where uncertainties in the data exist and how significant they are. While simple data types like uncertain scalar values have been investigated thoroughly in the literature, the visualization of uncertainty in more complex data still remains one of the top challenges in the field of scientific visualization [JS03, BHJ*14].

Depending on the type of uncertain data, visualization methods can be categorized into two groups. In classical *uncertainty visualization*, the uncertainty in the data is represented by a stochastic model. For instance, each data value may be given through a mean value and a corresponding standard deviation. In *ensemble visualization*, on the other hand, the uncertainty is represented by a set of possible states or data occurrences. Such a set—called *ensemble*—can be the result of, e.g., repeatedly conducting a real world experiment or performing multiple runs of a numerical simulation with varying input parameters. Ensembles are particularly useful in situations where common stochastic distribution models are not a good approximation of the actual distribution of data values.

In this thesis we are specifically concerned with ensembles that are created through numerical simulations. However, we are not only interested in the visualization of such ensembles, but also in their generation, i.e., we also investigate the area of numerical simulation. This is interesting to do because the visualization of simulations often requires knowledge about the data generating process, and, in particular, many advanced visualization pipelines rely on a tight coupling of visualization and simulation. An example for this is *visual steering*, in which a two-way coupling between a running

simulation and a live visualization is established. This can allow a user, e.g., to monitor and check the state of a simulation or to interactively tune simulation parameters. Another combination of simulation and visualization can be found in modern supercomputing: If the output of a simulation is too large to be permanently stored on disk, a workaround is to create the visualizations of interest immediately during the simulation and to store them in place of the raw simulation data, which is called *in situ* visualization. In any case, if a simulation and a visualization have to be combined, the performance as well as the memory requirements of the simulation are crucial. This is particularly true in the context of ensemble simulations, which require multiple simulation runs to be computed instead of only a single run.

In order to explore the possibilities of combining ensemble simulations and visualizations, we chose to focus on the field of computational fluid dynamics, since fluid simulation as well as flow visualization are very popular and well researched areas. This adds to our general effort of developing new, advanced visualization techniques for ensemble simulations and, accordingly, this thesis is split into two major parts describing our contributions to the fields of *fluid simulation* and *ensemble visualization*.

Fluid Simulation In the first part of this thesis, we propose a method for the efficient simulation of incompressible fluids in large and complex domains. For this purpose, we aim for a discretization scheme that can handle high effective resolutions at reasonable computational costs. While regular grids are generally attractive for numerical simulations because they give rise to efficient computational kernels, they are only of limited suitability in this case due to memory and time constraints. Unstructured grids, on the other hand, can flexibly adapt to the domain and only consume computational resources where needed, but they come at the cost of an irregular discretization and do not allow for a direct application of efficient hierarchical solvers like multigrid.

The main motivation of our method is to demonstrate that the efficiency of regular grids can be combined with the flexibility of unstructured grids. We use an adaptive octree grid with a hexahedral finite element discretization, which allows us to coarsen the grid in the interior of the fluid domain while keeping the computational kernels regular. To allow for arbitrary timesteps, we adapt concepts from fluid simulations in computer animation which make the simulation unconditionally stable. Further, we employ a geometric-algebraic multigrid solver to solve the pressure Poisson in a time-efficient manner, and we improve the convergence of this solver in complex scenarios by proposing a cell duplication scheme at coarser scales. In our examples, we focus on the simulation of liquids with free surfaces, and we show how the fluid/air interface can be resolved with second-order accuracy by enforcing the Dirichlet boundary conditions for the pressure in a variational sense using a particular class of Nitsche methods. On a single desktop computer, the resulting method can easily handle scenarios that would require hundreds of millions of simulation elements in a non-adaptive regime.

Ensemble Visualization In the second part of this thesis, we propose a novel visual metaphor for the visualization of line based features in scalar and vector field ensembles. In practice, such ensembles are often visualized using so-called *spaghetti plots*, which are created by overlaying the lines corresponding to individual ensemble members in a single view. Unfortunately, spaghetti plots often contain a lot of visual clutter, which can make it difficult to discern trends or outliers, and they do not scale well with increasing numbers of lines in the image. To solve these problems, we introduce the concept of *variability plots*. We use principal component analysis (PCA) to transform an input ensemble of lines into a low-dimensional feature space, and we use clustering to group the ensemble members into significantly differing trends. We then fit a continuous statistical model to the trends in the feature space, and exploit the invertibility of PCA to transform geometric representatives of this model back to the original domain space. This creates an abstract visualization of the input line ensemble, consisting of artificial representative lines and confidence regions, which illustrate the spatial standard deviation of individual trends. As a result, variability plots are able to effectively convey the major trends and outliers in an ensemble of line based features, which we demonstrate in three specific applications:

- We show how *streamline variability plots* can be created for ensembles of streamlines, which are seeded from the same location in each member of an ensemble of steady vector fields. In particular, this application demonstrates that PCA provides a powerful feature space representation for streamlines (and parameterized curves in general), which is well suited for line clustering. Moreover, because PCA is trivial to invert, we provide a mathematical model for creating artificial lines which are statistically similar to a given set of lines.
- We show how *contour variability plots* can be created for ensembles of iso-contours, which are extracted from each member of an ensemble of scalar fields using the same iso-value. Additionally, we extend the resulting plots with a special correlation visualization, for the first time enabling users to perform an interactive visual analysis of global correlations between the occurrences of iso-contours at different locations in the domain.
- We present advanced visualizations for iso-contours of time-dependent scalar fields in weather forecast ensembles, which are commonly analyzed by animating through sequences of spaghetti plots. Such animations are even harder to comprehend than single spaghetti plots, because individual timesteps have to be memorized in order to establish connections between different timesteps. To address this problem, we extend the clustering scheme of contour variability plots to a special *time-hierarchical clustering*, which is specifically tailored to the slowly diverging nature of weather forecast ensembles. It ensures hierarchical relationships between the clusterings of consecutive timesteps, and allows us to simplify the visualizations of single timesteps to a point where they can be combined with the visualizations of other timesteps in a meaningful

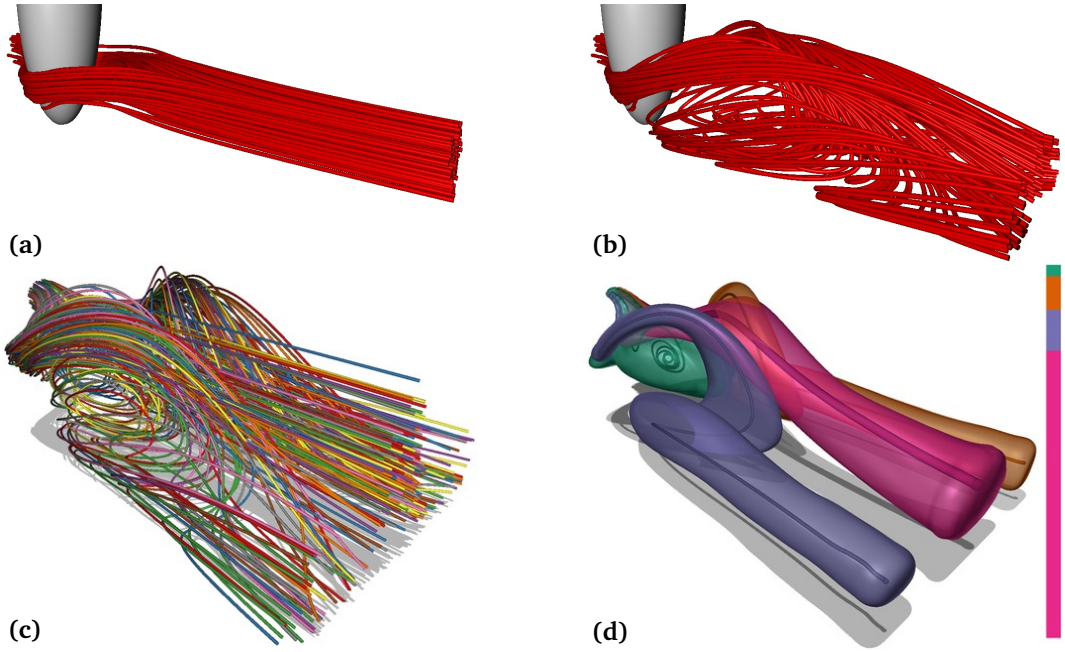


Figure 1.1.: Ensemble of 50 steady channel flows with different Reynolds numbers, simulated and visualized using the methods proposed in this thesis: **(a+b)** Standard visualization of two different ensemble members using streamlines. **(c)** Ensemble of 200 streamlines (four per ensemble member). **(d)** Three-dimensional streamline variability plot (see Chapter 5) with four clusters of the lines shown in (c).

way. For each timestep, we either draw single cluster representatives or *contour variability plots*, and create stacked plots following the principle of space-time cubes. As a result, we are able to effectively convey—in a single image—the major trends in the temporal evolution of the weather forecast ensemble towards a user-selected time window of interest.

Bringing everything together, we have also made efforts to combine the simulation and visualization methods proposed in this thesis. An example for this is depicted in Fig. 1.1, which shows several visualizations of an ensemble of 50 steady channel flows around an ellipsoidal obstacle. To generate the different ensemble members, the Reynolds number of the flow was varied. An ensemble of 200 streamlines is depicted in Fig. 1.1 (c), which was generated by seeding four streamlines from the same location (with a small, randomized offset) in each of the 50 members of the flow ensemble. A corresponding streamline variability plot is depicted in Fig. 1.1 (d). It reveals four major trends in the streamlines, each represented by an artifical median line and a surrounding confidence lobe illustrating its standard deviation. Furthermore, the bar plot to the right indicates the number of lines which belong to each individual trend.

1.1. Outline

The remainder of this thesis is structured as follows. In Part I, we present our proposed method for fluid simulation. For this, in Chapter 2, we first give a general introduction into fluid simulation based on our finite element discretization as well as multigrid, mostly under the simplifying assumption of a Cartesian grid. In Chapter 3, we then introduce the actual method, which extends the concepts introduced in Chapter 2 to an adaptive octree grid and adds support for free surfaces and second-order accurate boundary conditions.

In Part II, we introduce the concept of variability plots for ensemble visualization. In Chapter 4, we begin with a discussion of some common aspects and mathematical fundamentals. Subsequently, in Chapters 5 and 6, we present our two applications of variability plots to ensembles of vector and scalar fields by introducing streamline variability plots and contour variability plots, respectively. In Chapter 7, we then introduce our time-hierarchical clustering for iso-contours in weather forecast ensembles, which builds upon the concepts of contour variability plots. Finally, Chapter 8 summarizes the thesis and presents some directions for future work.

1.2. List of Publications

Some of the research results presented in this thesis have been originally published in the following peer-reviewed conference papers and journal articles:

- F. FERSTL, R. WESTERMANN, C. DICK: Large-Scale Liquid Simulation on Adaptive Hexahedral Grids. *IEEE Transactions on Visualization and Computer Graphics*, 20(10):1405-1417, 2014.
  doi:10.1109/TVCG.2014.2307873
- F. FERSTL, K. BÜRGER, R. WESTERMANN: Streamline Variability Plots for Characterizing the Uncertainty in Vector Field Ensembles. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2015)*, 22(1):767-776, 2016.
  doi:10.1109/TVCG.2015.2467204
- F. FERSTL, M. KANZLER, M. RAUTENHAUS, R. WESTERMANN: Visual Analysis of Spatial Variability and Global Correlations in Ensembles of Iso-Contours. *Computer Graphics Forum (Proceedings of EuroVis 2016)*, 35(3):221–230, 2016.
  doi:10.1111/cgf.12898
- F. FERSTL, M. KANZLER, M. RAUTENHAUS, R. WESTERMANN: Time-hierarchical Clustering and Visualization of Weather Forecast Ensembles. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2016)*, 2017, *accepted for publication*.
  doi:10.1109/TVCG.2016.2598868

Part I.

Fluid Simulation

2

Fundamentals of Finite Element Fluid Simulation

The numerical simulation of fluid phenomena has a long tradition in engineering and various other fields, and, since the fundamental works by Foster, Metaxas and Stam [FM96, Sta99], has also gained a lot of popularity in the area of computer graphics and animation. While traditional applications in computational fluid dynamics aim for numerical accuracy by, e.g., using boundary-fitted unstructured grids, in computer graphics, the need to realistically animate fluids in the shortest time possible has lead to a range of interesting simulation techniques which focus on simplicity and performance. In our method for incompressible fluid simulation, which we present in Chapter 3, we incorporate techniques from both "worlds". One the one hand, we employ a finite element method (FEM) on an octree grid in order to obtain an adaptive discretization which allows for coarsening in the fluids interior and which ensures consistency across the resulting level transitions in the grid. On the other hand, we make the simulation unconditionally stable by using implicit Euler integration for diffusion effects and semi-Lagragian advection for convective terms, which allows us to perform large timesteps. Furthermore, we present a special multigrid solver which is able to handle arbitrary octree grids in a generic fashion.

In this chapter, we cover the fundamentals of FEM, time integration and multigrid as used in our method. For this, we mostly use the assumption of a (non-adaptive) Cartesian grid. In Section 2.1, we briefly derive the Navier-Stokes equations. Then, in Section 2.2, we discuss the FEM discretization and our particular choice of elements, and, in Section 2.3, we setup a corresponding unconditionally stable timestep algorithm. In Section 2.4, we show how the analytical precomputation of element matrices can be used to accelerate the use of FEM at runtime, and, in Section 2.5, we explain how hanging vertices can be eliminated in order to extend the FEM discretization to octree grids. Finally, in Section 2.6, we discuss the fundamental concepts of multigrid by introducing a basic version of our solver.

2.1. Navier-Stokes Equations

The dynamic behavior of fluids modeled as a continuum is described by the Navier-Stokes equations (NSE) which were discovered by Claude-Louis Navier and George Gabriel Stokes in the 19th century. The NSE are a set of partial differential equations (PDEs) for which a variety of different formulations can be found in the literature. This is partly owed to the existence of several common special cases for different kinds of fluids (e.g., incompressible or inviscid fluids), and partly to the fact that the equations can be expressed in terms of different physical variables like velocity and vorticity or even mathematical constructs like stream functions. In the following section, we briefly show how to derive one of the most commonly used formulations: the NSE for incompressible, Newtonian fluids expressed in terms of the primitive variables velocity and pressure. For a more detailed derivation and discussion of the different formulations of the NSE we refer the reader to fundamental books on this topic, e.g., in the context of FEM [GS00, DH04, ESW14].

Let us consider a continuous fluid with velocity $\mathbf{u}$ and density ρ, and let Ω denote an infinitesimal control volume in space. Then, if no mass is lost or created, the total time rate of change of the density in the volume and the sum of the net flow across the volumes boundary $\partial\Omega$ have to cancel each other out:

$$\int_\Omega \frac{\partial \rho}{\partial t} + \oint_{\partial\Omega} \rho \mathbf{u} \cdot \mathbf{n} = 0, \tag{2.1}$$

where $\mathbf{n}$ denotes the normal of $\partial\Omega$. Using the divergence theorem on the surface integral, we get

$$\int_\Omega \frac{\partial \rho}{\partial t} + \int_\Omega \nabla \cdot (\rho \mathbf{u}) = \int_\Omega \left(\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) \right) = 0. \tag{2.2}$$

Since this identity has to hold for any control volume in space, the integrand of the last integral has to be zero at any point and, hence, we can leave out the integral. This leads us to what is generally known as the *continuum equation*, which ensures the conservation of mass and is the first part of the general NSE:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{2.3}$$

The second part is the so-called *momentum equation*, which ensures the conservation of momentum:

$$\frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u}) = \rho \mathbf{g} + \nabla \cdot \sigma \tag{2.4}$$

Roughly speaking, the left hand side of this equation can be derived similar to Eq. (2.3) by requiring the density momentum $\rho\mathbf{u}$ to be conserved instead of the density ρ, i.e., by replacing ρ with $\rho\mathbf{u}$ in

Eq. (2.3). The right-hand side adds additional source terms which allow for changes in momentum caused by external volume forces $\mathbf{g}$ and internal stresses in the fluid, which are described by the second-order stress tensor σ. Note that $\mathbf{uu}$ is a dyadic tensor (i.e., a second-order tensor).

From the general NSE (Eqs. (2.3 + 2.4)) we can derive a more specialized formulation for our purpose. Fluids are called *Newtonian* if their internal, viscous stresses linearly depend on the local strain rate, which is a property of many common liquids and gases like, e.g., water, oil and air. On the other hand, materials such as sand and toothpaste can be modeled as general fluids, but not as Newtonian fluids. Physically, the property of a fluid being Newtonian can be expressed by replacing the stress tensor term on the right hand side of Eq. (2.4) in the following way, which introduces the internal pressure p and the fluids viscosity μ:

$$\begin{aligned} \rho\mathbf{g} + \nabla\cdot\sigma &= \rho\mathbf{g} - \nabla p + \nabla\cdot\mu(\nabla\mathbf{u} + \nabla\mathbf{u}^T) \\ &= \rho\mathbf{g} - \nabla p + \mu\Delta\mathbf{u} + \nabla\mu\cdot(\nabla\mathbf{u} + \nabla\mathbf{u}^T) + \mu\nabla(\nabla\cdot\mathbf{u}) \end{aligned} \tag{2.5}$$

Here, in the second line, we have extended the viscosity term by applying the divergence operator. Similarly, we can extend the left hand side of Eq. (2.4) by applying the product rule and the divergence operator:

$$\frac{\partial(\rho\mathbf{u})}{\partial t} + \nabla\cdot(\rho\mathbf{uu}) = \frac{\partial\rho}{\partial t}\mathbf{u} + \rho\frac{\partial\mathbf{u}}{\partial t} + \rho\mathbf{u}\cdot\nabla\mathbf{u} + \mathbf{u}(\nabla\cdot(\rho\mathbf{u})) \tag{2.6}$$

Now, in addition to the fluid being Newtonian, we assume that the fluid is homogeneous and incompressible, which means that the density ρ and viscosity μ are constants. As a result, the general continuity equation (Eq. (2.3)) reduces to the well known incompressibility constraint

$$\nabla\cdot\mathbf{u} = 0. \tag{2.7}$$

This means that the velocity $\mathbf{u}$ has to be free of divergence at all times and, thus, that the corresponding vector field cannot contain any sources or sinks which would cause an expansion or compression of the fluid mass. Using Eq. (2.7) in addition to the assumption that ρ and μ are constant, the left and right hand side of the momentum equation (Eqs. (2.5+2.6)) can be reduced and recombined into the momentum equation for incompressible, Newtonian fluids:

$$\rho\dot{\mathbf{u}} + \rho\mathbf{u}\cdot\nabla\mathbf{u} = \rho\mathbf{g} - \nabla p + \mu\Delta\mathbf{u} \tag{2.8}$$

where we use $\dot{\mathbf{u}} := \frac{\partial\mathbf{u}}{\partial t}$. This equation expresses the fact that temporal changes of the velocity field are induced by a combination of four effects: convection ($\mathbf{u}\cdot\nabla\mathbf{u}$), diffusion due to viscous forces ($\mu\Delta\mathbf{u}$), external volume forces acting on the fluid ($\mathbf{g}$) and a force caused by the internal pressure of the fluid

(∇p). Note that the inviscid form of these equations, i.e., if $\mu = 0$, is also referred to as the *Euler equations*.

2.1.1. Pressure Poisson Equation

To integrate the NSE forward in discrete time, many methods rely on the pressure projection method, which goes back to work proposed by Chorin [Cho68]. Instead of calculating the velocity and pressure for the next timestep directly via a coupled system of equations, the basic idea of the projection method is to calculate the pressure before updating the velocity by solving a special Poisson equation. This yields a multi-step scheme which is both simpler and much more efficient to execute than the direct, coupled approach. To derive the pressure Poisson equation (PPE), we rearrange the momentum equation (Eq. (2.8)) and solve for $\dot{\mathbf{u}}$:

$$\dot{\mathbf{u}} = \underbrace{-\mathbf{u}\cdot\nabla\mathbf{u} + \frac{\mu}{\rho}\Delta\mathbf{u} + \mathbf{g}}_{:=\dot{\mathbf{u}}^*} - \frac{1}{\rho}\nabla p \tag{2.9}$$

In terms of the primitive variables, the rate of change of velocity depends on the current velocity field as well as the current pressure. The evolution of the pressure field, however, is not described by any equation. Instead, the pressure is always uniquely defined via the incompressibility constraint of the continuity equation. To illustrate this, let $\dot{\mathbf{u}}^*$ denote an intermediate rate of change of velocity as defined in Eq. (2.9), which can be evaluated at any given time without knowledge of the current pressure. We can plug the shortened version of Eq. (2.9) into the continuum equation (Eq. (2.7)) (or more precisely, its derivative with respect to time: $\nabla\cdot\dot{\mathbf{u}} = 0$), which gives:

$$\nabla\cdot\dot{\mathbf{u}} = \nabla\cdot\left(\dot{\mathbf{u}}^* - \frac{1}{\rho}\nabla p\right) = 0 \tag{2.10}$$

Rearranging this equation leads to the following Poisson equation, in which the pressure is the unknown variable and the right hand side depends on $\dot{\mathbf{u}}^*$:

$$\frac{1}{\rho}\Delta p = \nabla\cdot\dot{\mathbf{u}}^* \tag{2.11}$$

Solving this Poisson equation results in a pressure which can be used to calculate the final rate of change of velocity according to Eq. (2.9). This yields the following general multi-step scheme for advancing a simulation by a discrete timestep: First, we calculate $\dot{\mathbf{u}}^*$ from the current velocity, then we solve the PPE (Eq. (2.11)) for p and, lastly, we calculate the final $\dot{\mathbf{u}}$ according to Eq. (2.9) and use it for the numerical integration of the velocity. In practice, the whole scheme is typically formulated in terms of an intermediate velocity $\mathbf{u}^*$ (instead of $\dot{\mathbf{u}}^*$) in order to avoid the accumulation of numerical

errors. For instance, in this work, we use an explicit Euler scheme for time integration, which gives the following rule to advance a current velocity $\mathbf{u}_t$ by a timestep Δt:

$$\mathbf{u}_{t+\Delta t} = \mathbf{u}_t + \Delta t\ \dot{\mathbf{u}}_t = \underbrace{\mathbf{u}_t + \Delta t\ \left(-\mathbf{u}_t \cdot \nabla \mathbf{u}_t + \frac{\mu}{\rho}\Delta \mathbf{u}_t + \mathbf{g}\right)}_{:=\mathbf{u}^*} - \frac{\Delta t}{\rho}\nabla p_t \tag{2.12}$$

$$\frac{1}{\rho}\Delta p_t = \frac{1}{\Delta t}\nabla \cdot \mathbf{u}^* \tag{2.13}$$

This rule explains the term *pressure projection method*, because an intermediate velocity $\mathbf{u}^*$ is projected to the space of divergence free vector fields by solving for and applying the correct pressure (resulting in $\mathbf{u}_{t+\Delta t}$). Mathematically, this can also be seen as an application of the *Hodge decomposition*, which states that any vector field (in our case $\mathbf{u}^*$) can be decomposed into a sum of a divergence free vector field ($\mathbf{u}_{t+\Delta t}$) and the gradient of a scalar field ($\nabla \frac{\Delta t}{\rho} p_t$).

2.1.2. Boundary Conditions

For solving the NSE on a finite domain Ω we need to impose boundary conditions on the domains boundary $\Gamma := \partial\Omega$. As is common for the NSE, for each point on Γ, we either prescribe a velocity value and let the pressure vary freely (a Dirichlet boundary condition for the velocity), or we prescribe a pressure value and let the velocity vary freely (a Neumann boundary condition for the velocity). (Almost) any combination of these two options on different parts of the boundary will result in well-posed boundary conditions. Therefore, we partition the boundary into two distinctive parts $\Gamma_{D(irichlet)}$ and $\Gamma_{N(eumann)}$ (with $\Gamma_D \uplus \Gamma_N = \Gamma$) on which we apply the respective types of boundary condition. Note that, in Chapter 3, Γ_D and Γ_N are referred to as $\Gamma_{W(all)}$ and $\Gamma_{A(ir)}$, respectively, because they are associated specifically with a liquid-solid and a liquid-air interface. Depending on how the NSE and PPE are discretized, there exist slightly different approaches to enforce these boundary conditions, which is particularly true for FEM. In the following description, we limit ourselves to the formulations which apply to our approach.

On the Dirichlet boundary Γ_D we prescribe a fixed velocity $\mathbf{u}_D$, which can be used to model, e.g., *no-slip* walls (using $\mathbf{u}_D = 0$) or moving objects (using $\mathbf{u}_D =$ *"object velocity"*):

$$\mathbf{u} = \mathbf{u}_D \qquad \text{on } \Gamma_D \tag{2.14}$$

It is also possible to prescribe only the normal components of $\mathbf{u}_D$ and let the tangential components vary freely, which can be used to realize *slip* walls. Note that, as a consequence of the continuity equation (Eq. 2.7)) and in order to conserve mass, $\int_\Gamma \mathbf{u} \cdot \mathbf{n} = 0$ has to hold, which is important in cases in which the velocity is prescribed on the whole boundary (i.e., $\Gamma = \Gamma_D$). As indicated before, if

the velocity is fixed, we cannot fix the pressure at the same location because, according to Eq. (2.12), the pressure has to satisfy

$$\mathbf{u}_D = \mathbf{u}^* - \frac{\Delta t}{\rho}\nabla p \qquad \text{on } \Gamma_D, \tag{2.15}$$

where $\mathbf{u}^*$ is the intermediate velocity according to the pressure projection method. Taking the inner product of this equation with the boundary normal $\mathbf{n}$ and rearranging leads to the following Neumann boundary condition for the pressure, which we can enforce during the pressure solve:

$$\mathbf{n} \cdot \nabla p = \mathbf{n} \cdot \frac{\rho}{\Delta t}(\mathbf{u}^* - \mathbf{u}_D) \qquad \text{on } \Gamma_D \tag{2.16}$$

On the Neumann boundary Γ_N we prescribe a fixed pressure p_N, which can be used to model, e.g., free surfaces (using $p_N = 0$) or outflows. This is described by a Dirichlet boundary condition for the pressure:

$$p = p_N \qquad \text{on } \Gamma_N \tag{2.17}$$

In contrast to before, we can now allow the velocity to vary freely by using the following Neumann boundary condition for the velocity:

$$\mathbf{n} \cdot \nabla \mathbf{u} = 0 \qquad \text{on } \Gamma_N \tag{2.18}$$

2.2. Finite Element Discretization

In the numerous branches of computational fluid dynamics a variety of common discretization schemes has been used successfully for the NSE. While many people still rely on finite difference discretizations because of their ease of use, finite volume methods and FEM have proven to be powerful as well because they can naturally handle complex domains. In particular, FEM can handle unstructured grids in a generic way and can be directly extended to higher-order methods [GS00]. In computer graphics and animation, however, the vast majority of approaches is based on staggered, Cartesian grids and finite difference discretizations [Bri08], since this combination offers a sufficient amount of accuracy for the rapid development of efficient fluid solvers. Unfortunately, even though efforts have been made to extend finite difference discretizations to non-Cartesian grids [LGF04, ZLC*13], the resulting methods still suffer from numerical instability or limited flexibility. In this work we use an adaptive, hexahedral octree grid and explicitly chose to use FEM for the following two reasons: First, the discretization is consistent and stable across octree level transitions with hanging vertices. Second, due to our particular choice of finite elements (i.e., basis functions),

the pressure nodes are located at the vertices of the octree grid and the resulting numerical stencils for the Laplace operator in the PPE yield near-optimal convergence rates in our multigrid solver. In the following, we introduce the general principles of FEM, discuss our particular choice of elements and show how the discrete equations can be derived.

The core idea of FEM is fundamentally different to finite differences. When using finite differences, the continuous solution of a PDE is approximated through a finite number of samples which are distributed over a regular or semi-regular grid, and everything between the points of this grid is ignored. As the name implies, derivatives are approximated through differences between values on neighboring grid points. In contrast to this, the FEM seeks an exact, fully defined solution to a weakened form of the continuous equations. This solution is searched from a space of finite dimensional functions, which is systematically built from a set of nodal basis functions with a simple algebraic representation (e.g. polynomials). If a grid of self-similar cells is used (which is the case for our hexahedral octree grid), we can precompute the analytical terms from which the coefficients of the discretized, linear problem are assembled. The precomputed terms are typically stored in the form of so-called *element matrices*, which are a powerful, computational tool that is similar to finite difference stencils, yet with the distinctive advantage that element matrices can be used on octree grids as well.

2.2.1. Weak Form of the Navier-Stokes Equations

A valid solution $(\mathbf{u}(\mathbf{x},t), p(\mathbf{x},t))$ of the NSE has to satisfy the momentum and continuity equation at *every point* $\mathbf{x} \in \Omega$ and at *every time* $t \in [t_{start}, t_{end}]$ of a given time interval. Furthermore, the NSE contain first- and second-order derivatives, which implicates that the solution has to be continuous and has to have a certain degree of smoothness. In the first step of the FEM discretization, we relax these constraints by transforming the original PDEs into a so-called *weak form* which admits solutions from a corresponding *weak solution space*. A main ingredient of the weak form are so-called test functions, which are best imagined as functions with a small, compact support like, e.g., box or hat functions. Intuitively, they allow us to check the average validity of the solution in small regions (as opposed to checking for exact validity at single points).

We start by choosing weak solution spaces with minimal smoothness requirements. Let $H^1(\Omega)$ denote the Sobolev-Space of square-integrable and once-differentiable functions, which basically includes all continuous functions on Ω that are (piece-wise) differentiable at least once. Then, for the velocity, we define the following vector-valued function spaces $\mathbf{U}$ and $\mathbf{V}$ for weak solutions $\mathbf{u}$ and corresponding test functions $\mathbf{v}$, respectively:

$$\mathbf{u} \in \mathbf{U} := \left\{ \mathbf{u} \in \left((H^1(\Omega)) \right)^3 \;\middle|\; \mathbf{u}|_{\Gamma_D} = \mathbf{u}_D \right\} \tag{2.19}$$

$$\mathbf{v} \in \mathbf{V} := \left\{ \mathbf{v} \in \left(H^1(\Omega) \right)^3 \;\middle|\; \mathbf{v}|_{\Gamma_D} = 0 \right\} \tag{2.20}$$

The Dirichlet boundary conditions for the velocity (Eq. (2.14)) are directly embedded into $\mathbf{U}$ and, hence, we can require all test functions to be zero on Γ_D since potential solutions $\mathbf{u} \in \mathbf{U}$ do not have to be tested there. Similarly, for the pressure, we define the following scalar-valued function spaces P and Q for weak solutions p and corresponding test functions q, respectively:

$$p \in P := \left\{ p \in H^1(\Omega) \;\middle|\; p|_{\Gamma_N} = p_N \right\} \tag{2.21}$$

$$q \in Q := \left\{ q \in H^1(\Omega) \;\middle|\; q|_{\Gamma_N} = 0 \right\} \tag{2.22}$$

The Dirichlet boundary conditions for the pressure (Eq. (2.17)) are embedded into P and Q in the same way as for the velocity. For more details on the choice of function spaces, let us also refer to more theoretical FEM literature like, e.g., the book by Braess [Bra07].

With properly defined solution and test function spaces, we can derive the weak form of the momentum equation (Eq. (2.8)) and the PPE (Eq. (2.13)), which incorporates the incompressibility constraint (Eq. (2.7)) as shown in Section 2.1.1. For a better overview, let us first recall those two equations:

$$\dot{\mathbf{u}} = -\mathbf{u} \cdot \nabla \mathbf{u} + \frac{\mu}{\rho} \Delta \mathbf{u} + \mathbf{g} - \frac{1}{\rho} \nabla p \tag{2.23}$$

$$\frac{1}{\rho} \Delta p = \frac{1}{\Delta t} \nabla \cdot \mathbf{u}^* \tag{2.24}$$

Note that the time derivative $\dot{\mathbf{u}}$ and the intermediate velocity $\mathbf{u}^*$ are also assumed to be members of the weak solution space $\mathbf{U}$. To obtain a "raw" weak form, we multiply both equations with test functions $\mathbf{v} \in \mathbf{V}$ and $q \in Q$, respectively, and integrate over Ω (strictly speaking, the "multiplication" is an inner product in the first case):

$$\int_\Omega \dot{\mathbf{u}} \cdot \mathbf{v} = -\int_\Omega \mathbf{u} \cdot \nabla \mathbf{u} \cdot \mathbf{v} + \frac{\mu}{\rho} \int_\Omega \Delta \mathbf{u} \cdot \mathbf{v} + \int_\Omega \mathbf{g} \cdot \mathbf{v} - \frac{1}{\rho} \int_\Omega \nabla p \cdot \mathbf{v} \tag{2.25}$$

$$\frac{1}{\rho} \int_\Omega q \Delta p = \frac{1}{\Delta t} \int_\Omega q \nabla \cdot \mathbf{u}^* \tag{2.26}$$

Because our weak solution spaces only support first-order derivatives, we have to take care of the two terms containing second order derivatives. First, using the identity $\nabla \cdot (\nabla \mathbf{a} \cdot \mathbf{b}) = \Delta \mathbf{a} \cdot \mathbf{b} + \nabla \mathbf{a} : (\nabla \mathbf{b})^T$ for first-order tensors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$ (which is basically a multi-dimensional application of the product

rule), we transform the diffusion term as follows:

$$\begin{aligned}\int_\Omega \Delta\mathbf{u}\cdot\mathbf{v} &= \int_\Omega \nabla\cdot(\nabla\mathbf{u}\cdot\mathbf{v}) \quad - \int_\Omega \nabla\mathbf{u} : (\nabla\mathbf{v})^T \\ &= \underbrace{\int_\Gamma \mathbf{n}\cdot\nabla\mathbf{u}\cdot\mathbf{v}}_{=0} \quad - \int_\Omega \nabla\mathbf{u} : (\nabla\mathbf{v})^T\end{aligned} \tag{2.27}$$

In the second line, we have applied the divergence theorem to restrict the first integral term to the boundary Γ. Then, because $\mathbf{v}$ is zero on Γ_D (see Eq. (2.20)) and $\mathbf{n}\cdot\nabla\mathbf{u}$ is zero on Γ_N (see Eq. (2.18)), we can conclude that this term always evaluates to zero and can be dropped. Note that $\mathbf{A} : (\mathbf{B})^T = \sum_{i=1}^d \sum_{j=1}^d A_{ij}B_{ij}$ denotes the double-dot product of two second-order tensors $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{d\times d}$.

Second, using the identity $\nabla\cdot(a\nabla b) = a\Delta b + \nabla a\cdot\nabla b$ for scalar functions $a, b \in \mathbb{R}$ (another multi-dimensional application of the product rule), we rewrite the pressure Laplacian term as follows:

$$\begin{aligned}\int_\Omega q\Delta p &= \int_\Omega \nabla\cdot(q\nabla p) \quad - \int_\Omega \nabla p\cdot\nabla q \\ &= \int_\Gamma q\mathbf{n}\cdot\nabla p \quad - \int_\Omega \nabla p\cdot\nabla q \\ &= \frac{\rho}{\Delta t} \int_{\Gamma_D} q\mathbf{n}\cdot(\mathbf{u}^* - \mathbf{u}_D) \quad - \int_\Omega \nabla p\cdot\nabla q\end{aligned} \tag{2.28}$$

In the second line, again, we have applied the divergence theorem to restrict the first integral term to the boundary Γ. Since q is always zero on Γ_N (see Eq. (2.22)), we can further restrict this boundary integral to Γ_D. This allows us, in the third line, to replace $\mathbf{n}\cdot\nabla p$ according to the Neumann boundary condition for the pressure (Eq. (2.16)), which only applies to Γ_D. Bringing everything together, we can now plug the two transformations Eq. (2.27) and Eq. (2.28) into our initial weak form (Eq. (2.26)), which gives the following weak formulations of the momentum equation and PPE:

$$\int_\Omega \dot{\mathbf{u}}\cdot\mathbf{v} = -\int_\Omega \mathbf{u}\cdot\nabla\mathbf{u}\cdot\mathbf{v} - \frac{\mu}{\rho}\int_\Omega \nabla\mathbf{u} : (\nabla\mathbf{v})^T + \int_\Omega \mathbf{g}\cdot\mathbf{v} - \frac{1}{\rho}\int_\Omega \nabla p\cdot\mathbf{v} \tag{2.29}$$

$$\frac{1}{\rho}\int_\Omega \nabla p\cdot\nabla q = -\frac{1}{\Delta t}\int_\Omega q\nabla\cdot\mathbf{u}^* + \frac{1}{\Delta t}\int_{\Gamma_D} q\mathbf{n}\cdot(\mathbf{u}^* - \mathbf{u}_D) \tag{2.30}$$

In order to solve the NSE, we are now left with the weak problem of finding $(\mathbf{u}, p) \in \mathbf{U}\times P$, such that Eqs. (2.29+2.30) hold for all test functions $(\mathbf{v}, q) \in \mathbf{V}\times Q$. In particular, compared to the original, strict form of the NSE, we can now allow solutions from our weak solution spaces $\mathbf{U}$ and P, which means we have successfully loosened the smoothness requirements for possible solutions. Let us note that this is only one of many existing weak forms of the NSE. The choice of an *appropriate* weak form,

however, is not a matter of personal preference but is rather tightly coupled to the choice of basis functions and boundary conditions. For a good overview on different weak forms of the NSE we refer to, e.g., the book by Gresho [GS00].

2.2.2. Discrete Matrix Form

Based upon the weak forms Eqs. (2.29+2.30), we can perform the second step of the FEM discretization and transform the continuous problem into a discrete, linear problem using an approach that is sometimes referred to as the Galerkin finite element method. Before we actually start, we introduce the following shorthand operators which directly correspond to the terms in the weak forms:

$$\begin{aligned} \mathrm{m}(\mathbf{u},\mathbf{v}) &= \int_\Omega \mathbf{u}\cdot\mathbf{v} & \mathrm{l}(p,q) &= \frac{1}{\rho}\int_\Omega \nabla p\cdot\nabla q \\ \mathrm{c}(\mathbf{u},\mathbf{w},\mathbf{v}) &= \int_\Omega \mathbf{u}\cdot\nabla\mathbf{w}\cdot\mathbf{v} & \mathrm{d}(\mathbf{u},q) &= \int_\Omega q\nabla\cdot\mathbf{u} \\ \mathrm{k}(\mathbf{u},\mathbf{v}) &= \frac{\mu}{\rho}\int_\Omega \nabla\mathbf{u} : (\nabla\mathbf{v})^T & \mathrm{b}(\mathbf{u},q) &= \int_{\Gamma_D} q\mathbf{n}\cdot\mathbf{u} \\ \mathrm{g}(p,\mathbf{v}) &= \frac{1}{\rho}\int_\Omega \nabla p\cdot\mathbf{v} \end{aligned} \tag{2.31}$$

Since integration, differentiation and multiplication (including dot and double-dot products) are linear operations, all these operators are bilinear/trilinear forms, i.e., they are linear in their arguments. For instance, in the case of $\mathrm{m} : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}$, the following rules hold for all $\mathbf{u},\mathbf{v},\mathbf{w} \in \mathbb{R}^3$ and $\lambda \in \mathbb{R}$:

$$\begin{aligned} \mathrm{m}(\lambda\,\mathbf{u},\mathbf{v}) &= \lambda\,\mathrm{m}(\mathbf{u},\mathbf{v}) \\ \mathrm{m}(\mathbf{u}+\mathbf{w},\mathbf{v}) &= \mathrm{m}(\mathbf{u},\mathbf{v}) + \mathrm{m}(\mathbf{w},\mathbf{v}) \\ \mathrm{m}(\mathbf{u},\mathbf{v}+\mathbf{w}) &= \mathrm{m}(\mathbf{u},\mathbf{v}) + \mathrm{m}(\mathbf{u},\mathbf{w}) \end{aligned} \tag{2.32}$$

Using the shorthand operators, the weak forms Eqs. (2.29+2.30) can be written concisely as:

$$\mathrm{m}(\dot{\mathbf{u}},\mathbf{v}) = -\mathrm{c}(\mathbf{u},\mathbf{u},\mathbf{v}) - \mathrm{k}(\mathbf{u},\mathbf{v}) + \mathrm{m}(\mathbf{g},\mathbf{v}) - \mathrm{g}(p,\mathbf{v}) \tag{2.33}$$

$$\Delta t\cdot\mathrm{l}(p,q) = -\mathrm{d}(\mathbf{u}^*,q) + \mathrm{b}(\mathbf{u}^*-\mathbf{u}_D,q) \tag{2.34}$$

To get do the discrete problem, we express the weak solutions and test functions as linear combinations of a finite set of basis functions, and thus restrict all function spaces to discrete subspaces

$$\mathbf{U}_h \subset \mathbf{U}, \quad \mathbf{V}_h \subset \mathbf{V}, \quad P_h \subset P, \text{ and } Q_h \subset Q. \tag{2.35}$$

More precisely, for all pressure-related quantities we introduce M scalar basis functions $\{\psi_1, ..., \psi_M\}$ as basis for P_h and Q_h, and for all velocity-related quantities we introduce N vector valued basis functions $\{\boldsymbol{\varphi}_1, ..., \boldsymbol{\varphi}_N\}$ as basis for $\mathbf{U}_h$ and $\mathbf{V}_h$. Members of the discrete subspaces can then be written as:

$$p_h \in P_h \Rightarrow p_h(\mathbf{x}, t) = \sum_{m=1}^{M} p_m(t)\, \psi_m(\mathbf{x}) \tag{2.36}$$

$$q_h \in Q_h \Rightarrow q_h(\mathbf{x}, t) = \sum_{j=1}^{M} q_j(t)\, \psi_j(\mathbf{x}) \tag{2.37}$$

$$\mathbf{u}_h \in \mathbf{U}_h \Rightarrow \mathbf{u}_h(\mathbf{x}, t) = \sum_{n=1}^{N} u_n(t)\, \boldsymbol{\varphi}_n(\mathbf{x}) \tag{2.38}$$

$$\mathbf{v}_h \in \mathbf{V}_h \Rightarrow \mathbf{v}_h(\mathbf{x}, t) = \sum_{i=1}^{N} v_i(t)\, \boldsymbol{\varphi}_i(\mathbf{x}) \tag{2.39}$$

As a consequence, assuming that the basis functions are fixed, all functions of interest are now uniquely defined by a discrete set of coefficients (e.g., $u_1(t), ..., u_N(t)$ for $\mathbf{u}_h$). These coefficients only depend on time and thus control all temporal dynamics, and the basis functions themselves only vary in space.

While our particular choice of basis functions is discussed in the upcoming section, here, we focus on the remaining, discrete problem in general. It reads as follows: Find $(\mathbf{u}_h, p_h) \in \mathbf{U}_h \times P_h$ such that the weak forms Eqs. (2.33+2.34) are satisfied for all $(\mathbf{v}_h, q_h) \in \mathbf{V}_h \times Q_h$:

$$\mathrm{m}(\dot{\mathbf{u}}_h, \mathbf{v}_h) = -\mathrm{c}(\mathbf{u}_h, \mathbf{u}_h, \mathbf{v}_h) - \mathrm{k}(\mathbf{u}_h, \mathbf{v}_h) + \mathrm{m}(\mathbf{g}, \mathbf{v}_h) - \mathrm{g}(p_h, \mathbf{v}_h) \qquad \forall \mathbf{v}_h \in \mathbf{V}_h \tag{2.40}$$

$$\Delta t \cdot \mathrm{l}(p_h, q_h) = -\mathrm{d}(\mathbf{u}_h^*, q_h) + \mathrm{b}(\mathbf{u}_h^* - \mathbf{u}_h^D, q_h) \qquad \forall q_h \in Q_h \tag{2.41}$$

Even though our solutions have a discrete representation now, we still have to test against an infinite set of test functions in order check whether a solution $(\mathbf{u}_h, p_h)$ is valid. Fortunately, we can avoid this by exploiting the linearity of the shorthand operators in the above weak forms. If we perform a test for each individual basis function, we can guarantee that any test against a linear combination of these basis functions will succeed as well. This means we can replace the tests against all $\mathbf{v}_h \in \mathbf{V}_h$ and $q_h \in Q_h$ with tests against all basis functions $\boldsymbol{\varphi}_i$ and ψ_j, respectively, and thus the following formulation is equivalent:

$$\mathrm{m}(\dot{\mathbf{u}}_h, \boldsymbol{\varphi}_i) = -\mathrm{c}(\mathbf{u}_h, \mathbf{u}_h, \boldsymbol{\varphi}_i) - \mathrm{k}(\mathbf{u}_h, \boldsymbol{\varphi}_i) + \mathrm{m}(\mathbf{g}, \boldsymbol{\varphi}_i) - \mathrm{g}(p_h, \boldsymbol{\varphi}_i) \qquad \forall \boldsymbol{\varphi}_i \ (i = 1, ..., N) \tag{2.42}$$

$$\Delta t \cdot \mathrm{l}(p_h, \psi_j) = -\mathrm{d}(\mathbf{u}_h^*, \psi_j) + \mathrm{b}(\mathbf{u}_h^* - \mathbf{u}_h^D, \psi_j) \qquad \forall \psi_j \ (j = 1, ..., M) \tag{2.43}$$

This leaves a discrete set of $N+M$ constraints, into which we plug the sum representations for the remaining functions which are based on P_h and $\mathbf{U_h}$ (see Eqs. (2.36+2.38)):

$$\begin{aligned} \mathrm{m}\left(\sum_{n=1}^{N}\dot{u}_n\boldsymbol{\varphi}_n,\boldsymbol{\varphi}_i\right) = &-\mathrm{c}\left(\sum_{n=1}^{N}u_n\boldsymbol{\varphi}_n,\sum_{n=1}^{N}u_n\boldsymbol{\varphi}_n,\boldsymbol{\varphi}_i\right)-\mathrm{k}\left(\sum_{n=1}^{N}u_n\boldsymbol{\varphi}_n,\boldsymbol{\varphi}_i\right) \\ &+\mathrm{m}(\mathbf{g},\boldsymbol{\varphi}_i)-\mathrm{g}\left(\sum_{m=1}^{M}p_m\psi_m,\boldsymbol{\varphi}_i\right) \end{aligned} \qquad \forall\boldsymbol{\varphi}_i \qquad (2.44)$$

$$\Delta t\cdot\mathrm{l}\left(\sum_{m=1}^{M}p_m\psi_m,\psi_j\right) = -\mathrm{d}\left(\sum_{n=1}^{N}u_n^*\boldsymbol{\varphi}_n,\psi_j\right)+\mathrm{b}\left(\sum_{n=1}^{N}\left(u_n^*-u_n^D\right)\boldsymbol{\varphi}_n,\psi_j\right) \qquad \forall\psi_j \qquad (2.45)$$

Note that we do no represent $\mathbf{g}$ with basis functions in order to support arbitrary external forces. Using the linearity of the shorthand operators for a second time, we can pull out the sums and the coefficients of the basis functions:

$$\begin{aligned} \sum_{n=1}^{N}\dot{u}_n\,\mathrm{m}(\boldsymbol{\varphi}_n,\boldsymbol{\varphi}_i) = &-\sum_{n_1=1}^{N}\sum_{n_2=1}^{N}u_{n_1}u_{n_2}\,\mathrm{c}\left(\boldsymbol{\varphi}_{n_1},\boldsymbol{\varphi}_{n_2},\boldsymbol{\varphi}_i\right)-\sum_{n=1}^{N}u_n\,\mathrm{k}(\boldsymbol{\varphi}_n,\boldsymbol{\varphi}_i) \\ &+\mathrm{m}(\mathbf{g},\boldsymbol{\varphi}_i)-\sum_{m=1}^{M}p_m\,\mathrm{g}(\psi_m,\boldsymbol{\varphi}_i) \end{aligned} \qquad \forall\boldsymbol{\varphi}_i \qquad (2.46)$$

$$\Delta t\cdot\sum_{m=1}^{M}p_m\,\mathrm{l}\left(\psi_m,\psi_j\right) = -\sum_{n=1}^{N}u_n^*\,\mathrm{d}\left(\boldsymbol{\varphi}_n,\psi_j\right)+\sum_{n=1}^{N}\left(u_n^*-u_n^D\right)\mathrm{b}\left(\boldsymbol{\varphi}_n,\psi_j\right) \qquad \forall\psi_j \qquad (2.47)$$

This separates the coefficients of the basis functions from the basis functions themselves, and eventually lets us recognize the final, discrete problem. Our degrees of freedom are the solution coefficients u_n, p_m etc., and they are related through fixed coefficients which only depend on the basis functions. The discrete problem is almost completely linear except for the c()-term, which is hardly surprising: This term goes back to the non-linear convection term $\mathbf{u}\cdot\nabla\mathbf{u}$ in the NSE, whose non-linearity in $\mathbf{u}$ is now reflected in the appearance of the quadratic expression $u_{n_1}u_{n_2}$. We can also write this term in a way that is similar to the other, linear terms:

$$\sum_{n_1=1}^{N}\sum_{n_2=1}^{N}u_{n_1}u_{n_2}\,\mathrm{c}\left(\boldsymbol{\varphi}_{n_1},\boldsymbol{\varphi}_{n_2},\boldsymbol{\varphi}_i\right)=\sum_{n=1}^{N}u_n\left(\sum_{n'=1}^{N}u_{n'}\,\mathrm{c}(\boldsymbol{\varphi}_{n'},\boldsymbol{\varphi}_n,\boldsymbol{\varphi}_i)\right) \qquad (2.48)$$

This makes it possible to express the discrete problem given by Eqs. (2.46+2.47) in a matrix-vector form, for which we first put the solution coefficients into vectors:

$$\begin{aligned} &\vec{\mathbf{u}}_h := (u_1,\ldots,u_N)^T, \quad \vec{p}_h := (p_1,\ldots,p_M)^T, \\ &\dot{\vec{\mathbf{u}}}_h := (\dot{u}_1,\ldots,\dot{u}_N)^T, \quad \vec{\mathbf{u}}_h^* := (u_1^*,\ldots,u_N^*)^T, \quad \vec{\mathbf{u}}_h^D := (u_1^D,\ldots,u_N^D)^T \end{aligned} \qquad (2.49)$$

Then, we put the coefficients which depend on the basis functions into matrices, and end up with the following linear relationships:

$$M\vec{\mathbf{u}}_h = -C(\vec{\mathbf{u}}_h)\vec{\mathbf{u}}_h - K\vec{\mathbf{u}}_h + \vec{\mathbf{g}}_h - G\vec{p}_h \tag{2.50}$$

$$\Delta t \cdot L\vec{p}_h = -D\vec{\mathbf{u}}_h^* + B(\vec{\mathbf{u}}_h^* - \vec{\mathbf{u}}_h^D), \tag{2.51}$$

where the entries of C depend on $\vec{\mathbf{u}}_h$ as indicated in Eq. (2.48). For the sake of simplicity, we drop the underscript "$\cdot_h$" and superscript "$\vec{\cdot}$" in the remainder, since it can be deferred from the context which variables refer to continuous functions and which refer to discrete vectors:

$$M\dot{\mathbf{u}} = -C(\mathbf{u})\mathbf{u} - K\mathbf{u} + \mathbf{g} - Gp \tag{2.52}$$

$$Lp = -\tfrac{1}{\Delta t}D\mathbf{u}^* + \tfrac{1}{\Delta t}B(\mathbf{u}^* - \mathbf{u}_D) \tag{2.53}$$

In detail, according to Eqs. (2.46+2.47), the dimensions and entries of the above matrices and the vector $\mathbf{g}$ are defined as follows:

$$N \times N: \quad (M)_{in} = \mathrm{m}(\boldsymbol{\varphi}_n, \boldsymbol{\varphi}_i) = \int_\Omega \boldsymbol{\varphi}_n \cdot \boldsymbol{\varphi}_i \tag{2.54}$$

$$N \times N: \quad (C)_{in} = \sum_{n'=1}^{N} u_{n'}\, \mathrm{c}(\boldsymbol{\varphi}_{n'}, \boldsymbol{\varphi}_n, \boldsymbol{\varphi}_i) = \sum_{n'=1}^{N} \int_\Omega \boldsymbol{\varphi}_{n'} \cdot \nabla\boldsymbol{\varphi}_n \cdot \boldsymbol{\varphi}_i \tag{2.55}$$

$$N \times N: \quad (K)_{in} = \mathrm{k}(\boldsymbol{\varphi}_n, \boldsymbol{\varphi}_i) = \frac{\mu}{\rho}\int_\Omega \nabla\boldsymbol{\varphi}_n : (\nabla\boldsymbol{\varphi}_i)^T \tag{2.56}$$

$$N \times 1: \quad (\mathbf{g})_i = \mathrm{m}(\mathbf{g}, \boldsymbol{\varphi}_i) = \int_\Omega \mathbf{g} \cdot \boldsymbol{\varphi}_i \tag{2.57}$$

$$N \times M: \quad (G)_{im} = \mathrm{g}(\psi_m, \boldsymbol{\varphi}_i) = \frac{1}{\rho}\int_\Omega \nabla\psi_m \cdot \boldsymbol{\varphi}_i \tag{2.58}$$

$$M \times M: \quad (L)_{jm} = \mathrm{l}\big(\psi_m, \psi_j\big) = \frac{1}{\rho}\int_\Omega \nabla\psi_m \cdot \nabla\psi_j \tag{2.59}$$

$$M \times N: \quad (D)_{jn} = \mathrm{d}\big(\boldsymbol{\varphi}_n, \psi_j\big) = \int_\Omega \psi_j \nabla \cdot \boldsymbol{\varphi}_n \tag{2.60}$$

$$M \times N: \quad (B)_{jn} = \mathrm{b}\big(\boldsymbol{\varphi}_n, \psi_j\big) = \int_{\Gamma_D} \psi_j \mathbf{n} \cdot \boldsymbol{\varphi}_n \tag{2.61}$$

Each matrix has a specific purpose. $C(\mathbf{u})$ corresponds to the convection term in the NSE and thus is the discrete convection operator. Likewise, K is the diffusion operator, G the pressure gradient operator, L the Laplace operator, D the divergence operator, and B a boundary operator corresponding to the boundary term in Eq. (2.30). Note that B has non-zero entries only in places which correspond to basis

functions that are non-zero at the boundary. M is the so-called mass matrix, which is a characteristic element of FEM discretizations. Intuitively, it can be seen as a scaling matrix which accounts for differences between the individual basis functions in, e.g., the size of their support or their "volume" $\left(\text{i.e., } \int \psi_j \text{ and } \int \boldsymbol{\varphi_i}\right)$. To be able to invert this matrix efficiently, we use a common technique called *mass-lumping*: We simplify M by "lumping" all entries of each row onto the diagonal. More precisely, we replace M with an approximate diagonal matrix M_{lumped}, whose diagonal entries are defined as the row-wise sums of M:

$$N \times N: \quad (M_{lumped})_{ii} = \sum_{n=1}^{N} (M)_{in} = \sum_{n=1}^{N} \mathrm{m}(\boldsymbol{\varphi}_n, \boldsymbol{\varphi}_i) = \sum_{n=1}^{N} \int_{\Omega} \boldsymbol{\varphi}_n \cdot \boldsymbol{\varphi}_i \overset{(*)}{=} \int_{\Omega} \boldsymbol{\varphi}_i \tag{2.62}$$

If the basis functions $\boldsymbol{\varphi}_i$ form a nodal basis and satisfy $\sum_{n=1}^{N} \boldsymbol{\varphi}_n = 1$ (as is the typical case), the diagonal entries of M_{lumped} are equal to the "volume" (or "mass") of the corresponding basis functions (*). Furthermore, the basis functions of nodal bases have a compact, local support, and thus most integrals in Eqs. (2.55-2.61) will evaluate to zero, which means that the remaining matrices in Eqs. (2.52+2.53) are very sparse.

Discrete Boundary Conditions

The boundary conditions for the continuous formulation (see Section 2.1.2) have been used to simplify the weak forms Eqs. (2.29+2.30), and thus are already integrated to a certain degree into the discrete matrix forms Eqs. (2.52+2.53), which we have just derived. For practical applications, however, it is necessary to discuss the treatment of boundary conditions in this discrete matrix form in more detail.

As indicated previously, it is common to use nodal basis functions in FEM. This entails that the values of $\mathbf{u}$ and p on the boundaries Γ_D and Γ_N are controlled by the coefficients of a small subset of basis functions, namely, the basis functions whose nodes are located on the respective boundaries. Therefore, in the presence of Dirichlet boundary conditions for the velocity or pressure, the coefficient vectors $\mathbf{u}$ and p in Eqs. (2.52+2.53) contain two types of entries: degrees of freedom and fixed (prescribed) entries. In addition to that, only the basis functions which are associated with a degree of freedom are used as test functions and yield a corresponding discrete constraint, i.e., a row of matrix entries (we do not have to test on Dirichlet boundaries, see Eqs. (2.20+2.22)). As a consequence, we loose one constraint for every fixed entry in $\mathbf{u}$ and p, and thus the number of degrees of freedom remains equal to the number of constraints.

The boundary conditions for the velocity are straightforward to realize in practice. In order to ensure that the velocity $\mathbf{u}$ stays fixed to the prescribed velocity $\mathbf{u}_D$ on Γ_D, we simply treat the corresponding basis function coefficients as read-only when performing velocity updates according to

Eq. (2.52). In the PPE (Eq. (2.53)), the velocity only acts as "input" to the right-hand side, and therefore we cannot violate any velocity boundary conditions when solving for the pressure.

In contrast to this, the boundary conditions for the pressure require more care, because the pressure is determined through the PPE (Eq. (2.53)), i.e., by solving a linear equation system. Let us assume without loss of generality that the first $M' < M$ coefficients in p are degrees of freedom and, consequently, that the last $\bar{M} := M - M'$ coefficients are fixed to the corresponding values of p_N on Γ_N. Then, by testing the weak PPE (Eq. (2.47)) against the M' basis functions which correspond to degrees of freedom, we get an intermediate PPE with a $M' \times M$ system matrix:

$$\begin{pmatrix} L' & \bar{L} \end{pmatrix} \begin{pmatrix} p' \\ \bar{p} \end{pmatrix} = b', \tag{2.63}$$

where $L' \in \mathbb{R}^{M' \times M'}$, $\bar{L} \in \mathbb{R}^{M' \times \bar{M}}$, $p' \in \mathbb{R}^{M'}$, $\bar{p} \in \mathbb{R}^{\bar{M}}$ and $b' \in \mathbb{R}^{M'}$. In this intermediate equation, we can eliminate the fixed pressure coefficients $\bar{p}$ from the left-hand side and bring them to the right-hand side:

$$L'p' = b' - \bar{L}\bar{p} \tag{2.64}$$

This leaves us with a proper, linear equation system in M' unknowns and with a square, $M' \times M'$ system matrix, which can be used to correctly solve for the pressure's degrees of freedom given by p'. In practice, the elimination of the fixed entries in p can be performed directly during the assembly of the equation system.

2.2.3. Choice of Elements

The choice of finite elements plays a central role in every FEM discretization and includes two separate aspects. Firstly, we have to decide how to decompose the domain into a finite number of elements and, secondly, we have to decide which basis functions to use. In this thesis, we use a structured grid of hexahedral elements and we represent both velocity and pressure with continuous functions which are trilinear on each element. For now, we assume the grid to be a Cartesian grid, but we relax this restriction in Section 2.5, where we consider hexahedral octree grids. The use of the same basis functions for velocity and pressure is referred to as *equal-order interpolation* and is not a very common choice for FEM based fluid simulations [GS00]. This is due to the fact that instabilities in the PPE can arise when trying to enforce the incompressibility constraint $\nabla \cdot \mathbf{u} = 0$ in a consistent discrete way. To avoid these problems, we use a so-called *approximate projection* [ABS96, GCCH95, SCG02, GF05], which conserves mass in a less strict manner but, on the other hand, is suited extremely well for iterative solvers including, in particular, multigrid. In the following, we first introduce our specific

choice of finite elements and basis functions and, second, discuss our decision.

The Q_1Q_1 Element

In common FEM terminology, the space of continuous functions on Ω which are trilinear on each finite element is referred to as Q_1. Other common function spaces on hexahedral grids include, e.g., P_0, the space of element-wise constant functions, and Q_2, the space of continuous, element-wise triquadratic functions (i.e., functions that can be written as polynomials $\sum_{i=0}^{2}\sum_{j=0}^{2}\sum_{k=0}^{2}\alpha_{ijk}x^i y^j z^k$ in each element). Because it is possible (and also common) to use different spaces for velocity and pressure, we need to specify a pair of spaces to describe a finite element for an incompressible fluid simulation. Since we use Q_1 for velocity as well as pressure, our choice of function spaces is thus called the Q_1Q_1 element.

In accordance with this choice, we have to properly define the bases $\{\psi_1, ..., \psi_M\}$ and $\{\varphi_1, ..., \varphi_N\}$, which we introduced in Section 2.2.2 for pressure and velocity related quantities, respectively. This means that we have to choose the φ_i and ψ_j such that

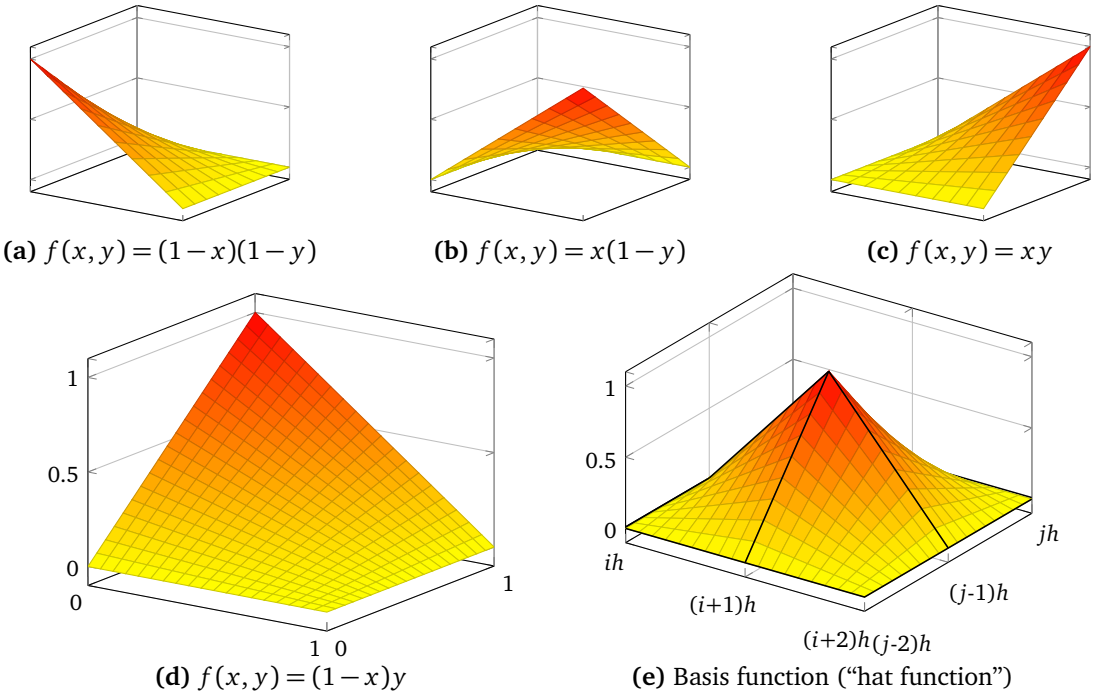
$$Q_1 = \operatorname*{span}_{j=1,...,M}\left\{\psi_j\right\} \quad \text{and} \quad (Q_1)^3 = \operatorname*{span}_{i=1,...,N}\left\{\varphi_i\right\}, \tag{2.65}$$

The vector valued velocity function space is obtained as a tensor product of the given scalar valued space Q_1. Hence, assuming that we will define a proper, scalar valued basis for Q_1, we can create the corresponding vector valued version by repeating the scalar basis functions in every dimension. This means that the velocity basis functions φ_i can be derived from appropriate ψ_j as

$$\{\varphi_1, ..., \varphi_N\} := \{\psi_1\mathbf{e}_1, \psi_1\mathbf{e}_2, \psi_1\mathbf{e}_3,\ \ \psi_2\mathbf{e}_1, \psi_2\mathbf{e}_2, \psi_2\mathbf{e}_3,\ \ \ldots, \psi_M\mathbf{e}_3\}, \tag{2.66}$$

where $\mathbf{e}_i$ denotes the i-th unit vector and $N = 3M$. Thus, we can limit our following considerations to determining the scalar valued basis functions ψ_j.

These are constructed as a nodal basis, using a given decomposition of the domain into—in our case—a finite set of hexahedra. In a nodal basis, each basis function ψ_j is associated with a point $\mathbf{x}_j \in \Omega$ (called node or vertex) and the basis functions are chosen such that they evaluate to one at their respective node (i.e., $\psi_j(\mathbf{x}_j) = 1$) and to zero at all other nodes (i.e., $\psi_j(\mathbf{x}_k) = 0$ if $j \neq k$). Furthermore, at least in the case of piece-wise polynomial bases, all basis functions have a compact support of only a few elements and sum up to a constant function of one (i.e., $\sum_j \psi_j = 1$). The use of a nodal basis implicitly ensures that our solution will be continuous across element boundaries (though not necessarily differentiable) and that the coefficients of the basis functions are equal to the value of the solution at the position of the corresponding node. In the case of Q_1, the nodes are located at the vertices of the hexahedral grid and, in practice, the coefficients can thus be stored and



(a) $f(x,y) = (1-x)(1-y)$ **(b)** $f(x,y) = x(1-y)$ **(c)** $f(x,y) = xy$

(d) $f(x,y) = (1-x)y$ **(e)** Basis function ("hat function")

Figure 2.1.: Q_1 in 2D: **(a-d)** The four bilinear basis function components on a reference cell $\left(=[0;1]^2\right)$. **(e)** A complete, nodal basis function in a Cartesian grid with spacing h, centered at the grid vertex $((i+1)h,(j-1)h)^T$ and composed from four basis function components. It has a support of four cells.

managed as vertices of a Cartesian grid. In contrast to Q_1, the nodes of P_0 are typically positioned at the center of each element, and the nodes of Q_2 are distributed on a regular, 3×3 grid over each element and comprise the elements' centroid, corner vertices, edge centroids and face centroids.

Since a visualization of the nodal basis functions for Q_1 is hard to realize in three dimensions, it is illustrated in Fig. 2.1 for two dimensions instead. Figs. 2.1 (a-d) show the four different components from which the individual basis functions can be composed. They are obtained on a square reference cell $\left(=[0;1]^2\right)$ by fitting 2D polynomials $f(x,y) = (\alpha_1 + \alpha_2 x)(\alpha_3 + \alpha_4 y)$ such that they evaluate to one at one corner and to zero at all other corners. Each nodal basis function ψ_i can be composed by scaling and translating these four components to the node's position $\mathbf{x}_j$, which is shown in Fig. 2.1(e) (less than four components are required at boundaries). Because of their appearance in 2D, the basis functions of Q_1 are often called "hat functions".

Discussion of Q_1Q_1

In contrast to many other FEM approaches for fluid simulation, we use an equal-order interpolation scheme with Q_1 basis functions for velocity as well as pressure. If strict, global mass conservation is desired (in a discrete sense), the typical choice are mixed-order interpolation schemes in which the

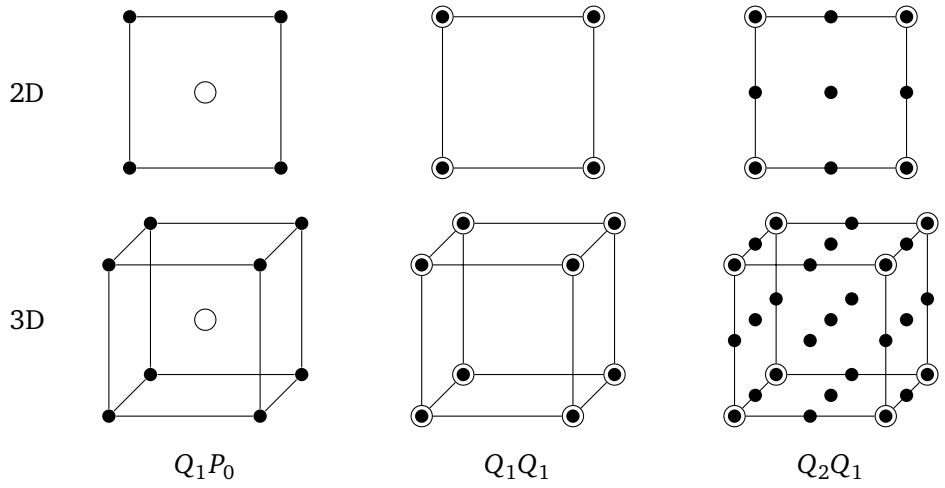


Figure 2.2.: The Q_1Q_1 element compared to other elements. For each element the location of its velocity nodes (●) and pressure nodes (○) is shown. In a full grid, on average, Q_1P_0 and Q_1Q_1 have an equal number of velocity and pressure nodes, whereas Q_2Q_1 has eight velocity nodes for every pressure node.

order of the weak velocity solution is higher than the order of the weak pressure solution. The most popular element [GS00] in this regard is Q_1P_0 (element-wise trilinear velocity, element-wise constant pressure), which is also the mixed-order element that has the lowest possible order. Higher-order elements like the so-called Taylor-Hood element Q_2Q_1 [TH73] (element-wise tri-quadratic velocity, element-wise trilinear pressure) have even more desirable numerical properties. A graphical comparison of these elements with our element Q_1Q_1 is depicted in Fig. 2.2.

As indicated before, the pros and cons of equal-order interpolation are strongly related to the PPE and instabilities that can arise in its discrete form. If we look back at our derivation of the pressure Poisson matrix L, we can see that we have first derived the continuous form of the PPE (Eq. (2.13)), from which we then proceeded to its weak form (Eq. (2.30)) and corresponding discrete matrix form (Eq. (2.53)). This derivation path is illustrated in the top path of Fig. 2.3. We have chosen a slightly *inconsistent* way of deriving the discrete PPE, because we only take into account the continuous form of the continuity equation (Eq. (2.7)), which is responsible for correct mass conservation. A more *consistent* way of deriving the discrete PPE is, roughly speaking, to discretize the continuity equation itself and to incorporate the result into the discrete PPE, thereby ensuring that the discrete continuity equation is satisfied exactly (bottom path in Fig. 2.3). A characteristic of this consistent way is that the pressure Poisson matrix is computed as a triple matrix product $DM^{-1}G$, where M is the FEM mass matrix and D and G are FEM divergence and gradient operators, respectively. Note that often $D = G^T$ (for more details we refer to, e.g., the book by Gresho [GS00], p. 640). This leads to an *exact projection* scheme (in a discrete sense), which ensures that $\int_\Omega \psi_j(\nabla \cdot \mathbf{u}) = 0$ for every test function ψ_j of the pressure space. For instance, in the case of Q_1P_0, the ψ_j are element-wise constant functions,

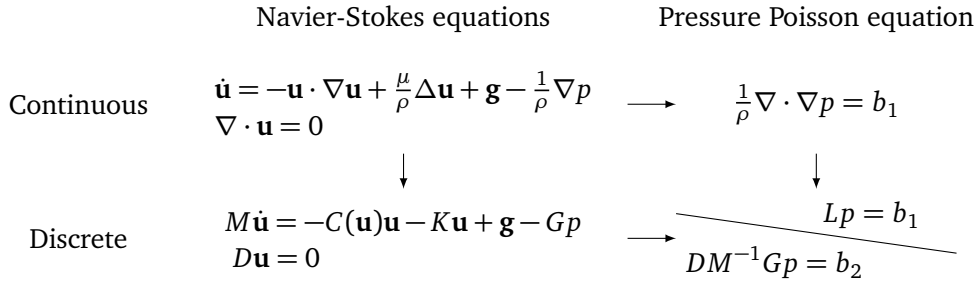


Figure 2.3.: Derivation of the discrete PPE in FEM. *Consistent* (bottom path): Discretizing the NSE and deriving the PPE from the discrete problem yields a Poisson matrix $DM^{-1}G$, which allows for *exact* mass conservation (in the discrete sense $D\mathbf{u} = 0$) but is only stable if LBB-stable elements are used. *Inconsistent* (top path): Deriving a continuous Poisson equation and discretizing it yields our Poisson matrix L, which only allows for an *approximate projection* but is extremely well suited for multigrid. Note that finite difference discretizations typically follow the inconsistent path as well [GF05, Bri08].

which lets us guarantee that $\int_e \nabla\cdot\mathbf{u} = 0$ in every hexahedral element e—this is one of the strongest possible forms of mass conservation.

Using a consistent derivation of the discrete PPE, on the other hand, can lead to issues of instability. This is the case for our equal-order interpolation scheme of Q_1Q_1. Here, the matrix product $DM^{-1}G$ results in a pressure Poisson matrix which does not have full rank (by far) and which has many so-called *spurious pressure modes* (i.e., vectors lying in the null-space of $DM^{-1}G$). Furthermore, as our experiments have shown, $DM^{-1}G$ has a very bad convergence behavior when plugged into iterative solvers like our multigrid solver.

The appearance of instabilities in the consistent pressure Poisson matrix $DM^{-1}G$ is mathematically well understood and completely depends on the choice of function spaces for velocity and pressure. We can make a statement about the presence of spurious pressure modes based on whether a pair (Q_h, V_h) of test-function spaces does fulfill the *LBB-condition* (named after Ladyshenskaya, Babuška and Brezzi), which is also referred to as the *inf-sup condition*. It is fulfilled if there exists a constant $\beta > 0$ independent of the grid spacing h, such that the following holds:

$$\inf_{q_h\in Q_h}\sup_{\mathbf{v}_h\in\mathbf{V}_h}\frac{(\nabla\cdot\mathbf{v}_h, q_h)}{\|\mathbf{v}_h\|\|q_h\|} \geq \beta \tag{2.67}$$

Here $\|\cdot\|$ denotes the norms in $H^1(\Omega)$ and $(H^1(\Omega))^3$, and $(v,q) := \int_\Omega v\,q$. For a detailed, mathematical explanation of the LBB-condition we refer to, e.g., the books by Gresho and Sani [GS00] and Brezzi and Fortin [BF91]. Interestingly, of all the aforementioned elements, only Q_2Q_1 fulfills this condition, whereas Q_1Q_1 and the popular element Q_1P_0 do not. In fact, Q_1P_0 has many spurious pressure modes in $DM^{-1}G$ which cause undesirable side effects (such as checker-style patterns in the resulting pressure).

There are two common remedies to the problem of LBB instability [GS00]. One the one hand, we can stabilize consistently derived but instable Poisson matrices $DM^{-1}G$ by adding artificial terms which affect the mathematical properties of the matrix but do not significantly affect the exactness of resulting numerical solutions. This stabilization idea has been successfully applied to, e.g., Q_1P_0 [DW89, HFB86] and Q_1Q_1 [Wal99]. Unfortunately, as our experiments with Q_1Q_1 have confirmed, the stabilization of $DM^{-1}G$ does not resolve the problem of bad convergence behavior in iterative solvers like multigrid. On the other hand, we can use what is often referred to as *approximate projection*. This is the path that we have decided to follow and that corresponds to using the pressure Poisson matrix L, which we have derived it in the previous sections (cf. Fig. 2.3). Even though methods based on Q_1Q_1 and an approximate projections are not widespread, several different authors have used this combination successfully [GCCH95, ABS96, SCG02]. For our application, this choice can be justified with two reasons: First, Q_1Q_1 uses a collocated grid in which all nodes reside on the vertices of the grid and which eases the realization of an octree grid. Second, the resulting discrete PPE can be assembled efficiently and has a small numerical stencil (27 points on a Cartesian grid in 3D) which is extremely multigrid friendly. Lastly, let us note that finite difference methods typically use approximate projections as well. Particularly in the area of computer graphics and animation this is common practice [GF05, Bri08]. It can easily be recognized by the fact that the continuous PPE is discretized directly and that the (often second-order accurate) boundary conditions for the pressure are imposed directly during this discretization.

2.3. Time Integration

To integrate the NSE forward in discrete time we use an explicit Euler scheme, and we further use operator splitting to be able to handle each term of the momentum equation separately. Following the fundamental ideas of Stam [Sta99] for computer animations, we integrate semi-Lagrangian advection and implicit diffusion into our FEM based simulation to make it unconditionally stable. This allows us to perform large timesteps and thus contributes to our goal of creating a highly efficient simulation.

Let us first recall the discrete NSE as derived in Section 2.2.2:

$$M\dot{\mathbf{u}} = -C(\mathbf{u})\mathbf{u} - K\mathbf{u} + \mathbf{g} - Gp \tag{2.68}$$

$$Lp = -\tfrac{1}{\Delta t}D\mathbf{u}^* + \tfrac{1}{\Delta t}B(\mathbf{u}^* - \mathbf{u}_D), \tag{2.69}$$

where we note that the PPE (Eq. (2.69)) already incorporates the assumption of an explicit Euler step.

By solving Eq. (2.68) for $\dot{\mathbf{u}}$, we can setup a general timestep from time t to $t + \Delta t$:

$$\begin{aligned} \mathbf{u}_{t+\Delta t} &= \mathbf{u}_t + \Delta t\, \dot{\mathbf{u}}_t \\ &= \underbrace{\mathbf{u}_t + \Delta t\, M^{-1}(-C(\mathbf{u}_t)\mathbf{u}_t - K\mathbf{u}_t + \mathbf{g})}_{=\mathbf{u}^*} - \Delta t\, M^{-1} G p_t \end{aligned} \tag{2.70}$$

At this point, let us remind of the fact that we use a lumped mass matrix and, thus, M is diagonal and can be inverted trivially. By means of operator splitting [Sta99], we can divide the Euler step (Eq. (2.70)) into smaller steps, where the "input" of each step only depends on the "output" of the previous step. This gives the following preliminary timestep algorithm:

$$1:\ \mathbf{u}' := \mathbf{u}_t - \Delta t\, M^{-1} C(\mathbf{u}_t)\mathbf{u}_t \tag{2.71}$$

$$2:\ \mathbf{u}'' := \mathbf{u}' + \Delta t\, M^{-1}\mathbf{g} \tag{2.72}$$

$$3:\ \mathbf{u}^* := \mathbf{u}'' - \Delta t\, M^{-1} K \mathbf{u}'' \tag{2.73}$$

$$4:\ \text{Solve for } p_t\text{: } \Delta t\, L p_t = -D\mathbf{u}^* + B(\mathbf{u}^* - \mathbf{u}_D) \tag{2.74}$$

$$5:\ \mathbf{u}_{t+\Delta t} := \mathbf{u}^* - \Delta t\, M^{-1} G p_t \tag{2.75}$$

Note that the intermediate velocity $\mathbf{u}^*$ and pressure projection are used in the same way as before. Despite the fact that we have added incremental dependencies between the individual steps and, e.g., that the diffusion term (Eq. (2.73)) now depends on $\mathbf{u}''$ instead of $\mathbf{u}_t$, this algorithm is still a valid explicit Euler scheme and converges to the correct solution, independent of the ordering of the substeps. In practice, however, it is a good idea to perform the convection (Eq. (2.71)) at the beginning of every timestep since, at this point, the velocity is still divergence free as a result of the projection (Eq. (2.75)) in the preceding timestep.

The major advantage of the split algorithm in Eqs. (2.71-2.75) is that it enables us to modularly replace individual explicit steps and to use alternative, more stable methods instead. In particular, the explicit Euler steps for convection and diffusion each impose severe timestep limitations on the numerical simulation. Thus, in the following two sections, we describe how to replace the explicit diffusion step (Eq. (2.73)) with an implicit one, and how to use semi-Lagrangian advection instead of the explicit convection step (Eq. (2.71)).

2.3.1. Implicit Diffusion

The explicit Euler step for the diffusion $\mathbf{u}^* := \mathbf{u}'' - \Delta t\, M^{-1} K \mathbf{u}''$ is based on the matrix K, which corresponds to a continuous Laplacian operator. Thus, K describes a second-order derivative and the explicit Euler step is only guaranteed to give stable results if $\Delta t = O(h^2)$ (where h is the grid spacing). If we factor in that the entries of K are proportional to μ / ρ, we get the following timestep limit for

the explicit diffusion step:

$$\Delta t \leq c_D \frac{\rho}{\mu} h^2, \tag{2.76}$$

where $c_D > 0$ is an appropriately chosen constant. With increasing μ / ρ or decreasing h this quickly becomes a performance bottleneck because the number of required timesteps can increase dramatically. A remedy for this problem is to use an implicit Euler step instead of an explicit one:

$$\mathbf{u}^* = \mathbf{u}'' - \Delta t \, M^{-1} K \mathbf{u}^* \tag{2.77}$$

This approach is unconditionally stable and does not cause the simulation to diverge, even for arbitrary large timesteps. However, it does not allow us to compute $\mathbf{u}^*$ directly from $\mathbf{u}''$. Rather, we rearrange Eq. (2.77) to reveal a linear equation system with unknown $\mathbf{u}^*$ and right-hand side $\mathbf{u}''$:

$$(ID + \Delta t \, M^{-1} K) \, \mathbf{u}^* = \mathbf{u}'', \tag{2.78}$$

where ID is the identity matrix. Thus, we can replace the explicit Euler step (Eq. (2.73)) with a linear solve of Eq. (2.78) for $\mathbf{u}^*$. This solve can be performed using any iterative solver and, since a few Jacobi iterations are typically sufficient, we do not use a more advanced method like conjugate gradients. This is supported by the fact that the simplicity of the Jacobi method allows our custom implementation to assemble the entries of $ID + \Delta t \, M^{-1} K$ in place and, thus, to avoid an explicit assembly of the entire matrix.

2.3.2. Semi-Lagrangian Advection

The timestep size for the explicit convection (or advection) step $\mathbf{u}' := \mathbf{u}_t - \Delta t \, M^{-1} C(\mathbf{u}_t) \mathbf{u}_\mathbf{t}$ is bounded by the well-known CFL-condition (named after Courant, Friedrichs and Lewy [CFL28]). Based on the maximum velocity magnitude u_{max} in the domain it limits the timestep according to

$$\Delta t \leq \frac{c_{CFL} \, h}{u_{max}}, \tag{2.79}$$

where $c_{CFL} > 0$ is a constant called the *Courant number* (or *CFL number*). Intuitively, the CFL-condition limits the distance that the currently fastest fluid particle can travel in the upcoming timestep to a multiple of h—the distance between two grid points. Accordingly, the CFL number prescribes how many grid cells the fastest fluid particle is allowed to travel in any one timestep. Explicit Euler integration can become instable even for $c_{CFL} < 1$. An unconditionally stable alternative is *semi-Lagrangian* advection, which can handle CFL numbers far greater than one, albeit at the cost of increased numerical damping.

To explain the central idea behind semi-Lagrangian advection, let us consider the general advection equation $\dot{s}+\mathbf{u}\cdot\nabla s = 0$, which describes the advection of a scalar quantity s through a velocity field $\mathbf{u}$. A Lagrangian interpretation of this equation is that s does not change along particle trajectories, which can also be expressed in terms of the material derivative, i.e., $\frac{Ds}{Dt} = 0$. Therefore, if we know the positions $\mathbf{p}_t$ and $\mathbf{p}_{t+\Delta t}$ of a particle in two consecutive timesteps, the following must hold:

$$s_{t+\Delta t}(\mathbf{p}_{t+\Delta t}) = s_t(\mathbf{p}_t) \tag{2.80}$$

We can estimate $\mathbf{p}_t$ from $\mathbf{p}_{t+\Delta t}$ using numerical integration over the velocity field $\mathbf{u}_t$ (backwards in time):

$$\mathbf{p}_t = \text{trace}(\mathbf{u}_t, \mathbf{p}_{t+\Delta t}, -\Delta t) := \mathbf{p}_{t+\Delta t} - \Delta t\, \mathbf{u}_t(\mathbf{p}_{t+\Delta t}) \tag{2.81}$$

The numerical integration is represented by the "trace" operator, which we have defined here using a simple explicit Euler scheme as an example. Furthermore, we have used the assumption that $\mathbf{u}_t \approx \mathbf{u}_{t+\Delta t}$. If we combine the ideas in Eqs. (2.80+2.81), transfer them to a fully discrete setting and switch back to the full convection equation $\dot{\mathbf{u}} + \mathbf{u}\cdot\nabla\mathbf{u} = 0$, we get the following *semi-Lagrangian* rule for computing an updated velocity at a vertex location $\mathbf{x}_j$ from the current, discrete velocity $\mathbf{u}_t$:

$$\mathbf{u}_{t+\Delta t}(\mathbf{x}_j) := \text{interpolate}(\mathbf{u}_t, \text{trace}(\mathbf{u}_t, \mathbf{x}_j, -\Delta t)) \tag{2.82}$$

Because $\mathbf{u}_t$ is discretely sampled on a grid, we introduce an additional "interpolate" operator to emphasize that we have to evaluate $\mathbf{u}_t$ at an arbitrary location. The two operators "trace" and "interpolate" represent the main ingredients of semi-Lagrangian advection which can be used to adjust the accuracy (and efficiency) of the method. In our case, we use a fourth-order Runge-Kutta scheme for the computation of particle trajectories in combination with a trilinear interpolation of the velocity field. The interpolation thus is in accordance with our choice of trilinear basis functions in the FEM discretization and can be extended consistently to octree grids.

Putting everything together, we incorporate implicit diffusion and semi-Lagrangian advection into our preliminary timestep algorithm (Eqs. (2.71-2.75)), which yields the final timestep algorithm:

Algorithm 1 Basic, unconditionally stable timestep algorithm.

1: $\mathbf{u}'$ $:= \text{Advect}(\mathbf{u}_t)$
2: $\mathbf{u}''$ $:= \mathbf{u}' + \Delta t\, M^{-1}\mathbf{g}$
3: Solve for $\mathbf{u}^*$: $(ID + \Delta t\, M^{-1}K)\,\mathbf{u}^* = \mathbf{u}''$
4: Solve for p_t: $\Delta t\, L p_t = -D\mathbf{u}^* + B(\mathbf{u}^* - \mathbf{u}_D)$
5: $\mathbf{u}_{t+\Delta t}$ $:= \mathbf{u}^* - \Delta t\, M^{-1} G p_t$

The semi-Lagrangian update (Eq. (2.82)) is applied to every vertex of the underlying grid, which is denoted with the "Advect" operation in Alg. 1.1, and the diffusion step Alg. 1.3 is now a linear equation solve. If diffusion is not desired (i.e., $\mu = 0$), this step can simply be omitted. Note that, as a side-effect of semi-Lagrangian advection, we have dropped the FEM convection operator $C(\mathbf{u})$, which is not required anymore. This is convenient because $C(\mathbf{u})$ is much more complex to use than the other FEM operators.

2.4. Element Matrices

After establishing the basic timestep algorithm formally, we now turn towards more practical aspects of FEM and exploit some properties of the structured, hexahedral grid. Since the nodal basis functions in this grid are all composed out of the same, self-similar components, we can identify reappearing integral terms in the weak forms (Eqs. (2.54-2.61)) and precompute them analytically. The result is typically stored in the form of so-called *element matrices*, which enable us to assemble the discrete FEM operators in a very efficient manner at runtime or, alternatively, to apply FEM operators in place if they are only required once (without an explicit assembly of the matrix). The concept of element matrices can even be extended to octree grids with hanging vertices, which we describe in Chapter 3. In the following, for demonstration purposes, we show how to derive and use the element matrix for the pressure Poisson operator L, and we present an explicit 2D example for this case. Element matrices for the other FEM operators can be derived analogously, but slightly more care has to be taken if the vector valued basis functions φ_i for the velocity are involved.

Let us assume an existing decomposition of our domain Ω into a finite set of hexahedra $\{e_1, \dots, e_E\}$, such that $\Omega = \biguplus_{i=1}^{E} e_i$. According to the definition of L in Eq. (2.59), we have to perform an integration over Ω to determine its entries, which we now replace with an integration in parts:

$$(L)_{jm} = \frac{1}{\rho} \int_{\Omega} \nabla\psi_m \cdot \nabla\psi_j = \sum_{i=1}^{E} \frac{1}{\rho} \int_{e_i} \nabla\psi_m \cdot \nabla\psi_j \tag{2.83}$$

Correspondingly, we can write L as a sum of "element-wise" matrices L_i^e:

$$L = \sum_{i=1}^{E} L_i^e \quad \text{with} \quad (L_i^e)_{jm} = \frac{1}{\rho} \int_{e_i} \nabla\psi_m \cdot \nabla\psi_j \tag{2.84}$$

These L_i^e are almost the matrices we are looking for, but they currently have the same dimensions as $L \in \mathbb{R}^{M \times M}$ and are only useful as a formal construct. However, they are built very systematically: Since each L_i^e is obtained through integration over a single element, and since each element is overlapped by exactly eight scalar basis functions (four in 2D), each L_i^e has to contain exactly 8×8 non-zero entries

(4×4 in 2D). Furthermore, since in all elements the basis functions' restrictions to the respective element are equal, the submatrices containing these non-zero entries have to be equal as well (up to row and column permutations). If we drop all zero rows and columns in each L_i^e and agree upon a standard ordering of the remaining rows and columns, we thus end up with identical matrices. Hence, it is sufficient to compute a representative element matrix $L^e \in \mathbb{R}^{8\times 8}$ ($\mathbb{R}^{4\times 4}$ in 2D) on a reference element, which contains the non-zero submatrix of each L_i^e in a standardized order.

The entries of this element matrix L^e can be precomputed analytically because we use a polynomial basis. As a concrete example, we consider the reference cell $[0,h]^2$ in two dimensions. The nodal basis functions corresponding to its four nodes (and restricted to the cell) are the following four bilinear polynomials (cf. Fig. 2.1):

$$\begin{aligned}
\Psi_1(x,y) &= \left(1-\frac{x}{h}\right)\cdot\left(1-\frac{y}{h}\right)\\
\Psi_2(x,y) &= \left(\frac{x}{h}\right)\cdot\left(1-\frac{y}{h}\right)\\
\Psi_3(x,y) &= \left(1-\frac{x}{h}\right)\cdot\left(\frac{y}{h}\right)\\
\Psi_4(x,y) &= \left(\frac{x}{h}\right)\cdot\left(\frac{y}{h}\right)
\end{aligned} \tag{2.85}$$

The 2D element matrix L^e is then computed as

$$L^e \in \mathbb{R}^{4\times 4} \quad \text{with} \quad (L^e)_{jm} = \frac{1}{\rho}\int_{[0,h]^2} \nabla\Psi_m \cdot \nabla\Psi_j, \tag{2.86}$$

which gives the following result:

$$L^e = \frac{1}{6\rho}\begin{pmatrix} 4 & -1 & -1 & -2\\ -1 & 4 & -2 & -1\\ -1 & -2 & 4 & -1\\ -2 & -1 & -1 & 4 \end{pmatrix} \tag{2.87}$$

Note that, in contrast to the general case, L^e in 2D does not depend on the grid spacing h (in 3D, it does). If we interpret the nodes of the reference cell as nodes of a graph and the element matrix as a weighted adjacency matrix, we can create an intuitive "visualization" of L^e, which is shown in Fig. 2.4 (a). The numbering of the Ψ_i and, consequently, the rows and columns in Eq. (2.87) corresponds to the ordering of the nodes in this illustration.

In order to assemble L from the element matrix L^e according to Eq. (2.84), we need to match the local node numbering, which was used to compute the element matrix, to the global numbering of grid nodes. For a given grid of elements, this matching can be represented by an index operator

$\mathcal{I} : \{e_1, \ldots, e_E\} \times \{1, \ldots, 4\} \to \{N_1, \ldots, N_M\}$, where $\mathcal{I}(e_i, k)$ gives the global node index N_j for the k-th node of element e_i as seen locally from this element. I.e., intuitively, $\mathcal{I}$ defines a computational grid by telling us which nodes belong to each cell. An example 2D grid with a corresponding index operator is shown in Figs. 2.4 (b+c). With the index operator $\mathcal{I}$ it is now possible to assemble L using the following systematic procedure:

Algorithm 2 Assembly of L from its element matrix L^e (in 2D).

1: $L := 0$
2: **for** $i = 1..E$ **do** ▷ Loop over elements e_i
3: **for** $k_1 = 1..4$ **do** ▷ Loop over nodes of e_i
4: **for** $k_2 = 1..4$ **do** ▷ Loop over nodes of e_i
5: $(L)_{\mathcal{I}(e_i,k_1),\mathcal{I}(e_i,k_2)} \mathrel{+}= (L^e)_{k_1,k_2}$
6: **end for**
7: **end for**
8: **end for**

This algorithm naturally extends to 3D and to other element matrices. It works for any hexahedral domain and, in particular, Ω is not required to be rectangular. In practice, the index operator $\mathcal{I}$ can be implemented in a tabular fashion (cf. Fig. 2.4 (c)) or, if Ω is rectangular, it can be specified implicitly. Furthermore, L corresponds to a numerical stencil with $3^3 = 27$ entries ($3^2 = 9$ in 2D), and can thus be stored in a matrix-free fashion, not requiring a generic sparse matrix data structure.

In many cases, we only need to apply an FEM operator once. This applies to, e.g., the projection step in Alg. 1.5 in which the gradient operator G is used to compute Gp. In such cases, we can use the element matrix of the FEM operator to directly perform the corresponding matrix-vector product in-place. Assume that we want to compute Lp in 2D. Then, according to Eq. (2.84), we can split up L into its sum of element matrices, $Lp = \sum_{i=1}^{E} L_i^e p$, which yields the following procedure:

Algorithm 3 In-place computation of $a := Lp$ using the element matrix L^e (in 2D).

1: $a := 0$
2: **for** $i = 1..E$ **do** ▷ Loop over elements e_i
3: **for** $k_1 = 1..4$ **do** ▷ Loop over nodes of e_i
4: **for** $k_2 = 1..4$ **do** ▷ Loop over nodes of e_i
5: $(a)_{\mathcal{I}(e_i,k_1)} \mathrel{+}= (L^e)_{k_1,k_2} \cdot (p)_{\mathcal{I}(e_i,k_2)}$
6: **end for**
7: **end for**
8: **end for**

This algorithm is very similar to Alg. 2 in its structure but does not require the entries of the global matrix L. Also, it is interesting to note that this algorithm is basically the FEM equivalent to evaluating a finite difference stencil at every node of a grid.

2.5. Octree Grids and Hanging Vertices

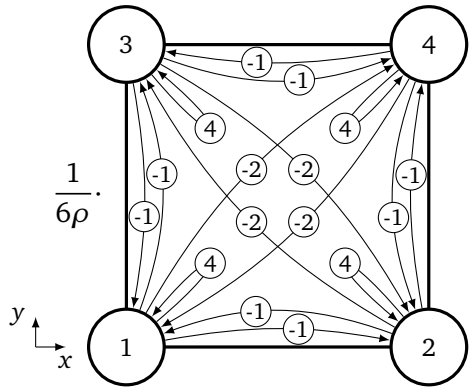
The FEM discretization, for which we have assumed a Cartesian grid up to this point, can be extended to support octree grids in a systematic manner. Octree grids contain level transitions where hexahedral elements of different size—in the following called *cells*—are located next to each other, which results in *hanging vertices*. We only use *restricted octrees*, in which the level difference of adjacent cells is limited to a maximum of one, and thus hanging vertices can either exist in the middle of cell edges or at the center of cell faces. To guarantee a continuous solution across level transitions, we restrict the weak solution spaces such that discontinuities are no longer possible. This is achieved by constraining the basis function coefficients at hanging vertices to trilinearly interpolated values of surrounding, non-hanging vertices, which implicates that coefficients at hanging vertices are no longer degrees of freedom. An example octree grid in two dimensions (and with square cells) is depicted in Fig. 2.5 (a). It has a single hanging vertex (N_{11}^+). To guarantee continuity along the corresponding edge from N_3 to N_8, we constrain its value using linear interpolation, e.g., a scalar basis function coefficient p_{11}^+ has to be constrained to $0.5p_3 + 0.5p_8$.

Formally, the restriction of values at hanging vertices can be expressed through an interpolation operator. Let M^+ denote the total number of vertices in the grid (including hanging vertices), and let M be the number of non-hanging vertices in the grid (i.e., the actual number of degrees of freedom). Then, the restriction constraints for the coefficients of the scalar valued basis functions ψ_j can be summarized in a linear interpolation operator $\mathrm{I} \in \mathbb{R}^{M^+ \times M}$ such that I maps a vector $p \in \mathbb{R}^M$, which contains the coefficients of all non-hanging grid vertices, to a vector $p^+ \in \mathbb{R}^{M^+}$, which contains the coefficients of all grid vertices:

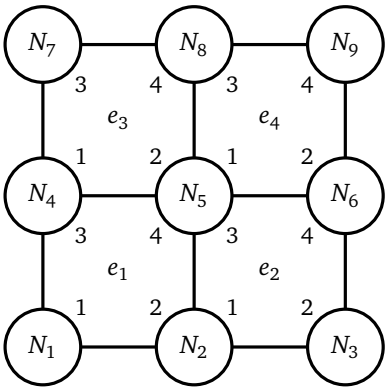
$$p^+ = \mathrm{I}p \tag{2.88}$$

I does not change the coefficients at non-hanging vertices and, thus, p^+ can be ordered such that the upper $M \times M$ part of I is an identity matrix. For the coefficients of the scalar valued basis functions φ_i, analogously, we can define a second interpolation operator $\mathbb{I} \in \mathbb{R}^{3M^+ \times 3M}$, which is basically a repetition of I in all three dimensions. Consequently, $\mathbb{I}$ constrains $\mathbf{u}^+ \in \mathbb{R}^{3M^+}$ to $\mathbf{u} \in \mathbb{R}^{3M}$, where

$$\mathbf{u}^+ = \mathbb{I}\mathbf{u}. \tag{2.89}$$



(a) L^e

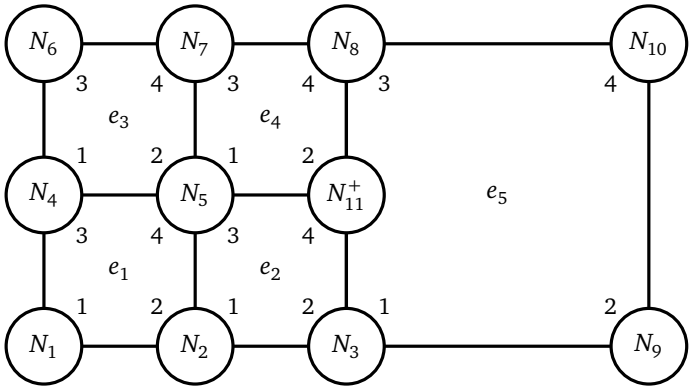


(b) Example Grid

e_i	k			
	1	2	3	4
e_1	N_1	N_2	N_4	N_5
e_2	N_2	N_3	N_5	N_6
e_3	N_4	N_5	N_7	N_8
e_4	N_5	N_6	N_8	N_9

(c) $\mathcal{I}(e_i, k) = N_j$

Figure 2.4.: Element matrix and global matrix assembly for L in 2D. **(a)** "Graph visualization" of the element matrix $L^e \in \mathbb{R}^{4\times4}$, interpreted as a weighted adjacency matrix, where each entry $(L^e)_{jm}$ is used to annotate an arrow from local node m to local node j. Note that L^e depends on h in 3D, but not in 2D. **(b)** Example hexahedral grid with four elements e_i. Nodes N_j are numbered with a global index j, local node indices $\{1,\dots,4\}$ with respect to each cell are indicated in the elements' interior. **(c)** The index operator $\mathcal{I}$ for the grid shown in (b), specifying four global node indices N_j for each element e_i. The global matrix $L \in \mathbb{R}^{9\times9}$ can be assembled using Alg. 2 and the information from (a-c). E.g., $(L)_{N4,N5} = \frac{1}{6\rho}\left((L^e)_{3,4} + (L^e)_{1,2}\right) = \frac{1}{6\rho}(-1-1)$, where the first contribution is due to element e_1, and the second due to element e_3.



(a) Example Octree Grid

e_i	k				h
	1	2	3	4	
e_1	N_1	N_2	N_4	N_5	h_0
e_2	N_2	N_3	N_5	N_{11}^+	h_0
e_3	N_4	N_5	N_6	N_7	h_0
e_4	N_5	N_{11}^+	N_7	N_8	h_0
e_5	N_3	N_9	N_8	N_{10}	$2h_0$

(b) $\mathcal{I}(e_i, k) = N_j$

Figure 2.5.: Octree grids and hanging vertex treatment in 2D (continuation of Fig. 2.4). **(a)** Example hexahedral octree grid with 5 elements and 11 nodes. Node N_{11}^+ is a hanging vertex and, hence, the basis function coefficients at this node are restricted via linear interpolation to ensure a continuous solution, i.e., $p_{11}^+ = 0.5p_3 + 0.5p_8$. **(b)** The index operator $\mathcal{I}$ for the octree grid shown in (b), additionally including the size h of each element. A global matrix $L^+ \in \mathbb{R}^{11\times11}$, which does not incorporate the hanging-vertex restriction, can be assembled using Alg. 2 in the same way as for non-octree grids and, in particular, the same element matrix L^e can be used (see Fig. 2.4(a)).

Having defined I and $\mathbb{I}$, we start from the initially unrestricted, discrete problem

$$M^+\dot{\mathbf{u}}^+ = -C^+(\mathbf{u}^+)\mathbf{u}^+ - K^+\mathbf{u}^+ + \mathbf{g}^+ - G^+p^+ \tag{2.90}$$

$$L^+p^+ = -\tfrac{1}{\Delta t}D^+\mathbf{u}^{*+} + \tfrac{1}{\Delta t}B^+(\mathbf{u}^{*+} - \mathbf{u}_D^+), \tag{2.91}$$

in which all operators and coefficient vectors have dimensions M^+ and $3M^+$, and which does allow for discontinuous solutions. We then obtain the desired restriction of the discrete solution spaces ($\mathbb{R}^{M^+}$ to $\mathbb{R}^M$, and $\mathbb{R}^{3M^+}$ to $\mathbb{R}^{3M}$) in two steps: First we, insert the hanging vertex constraints according to Eqs. (2.88+2.89):

$$M^+\mathbb{I}\,\dot{\mathbf{u}} = -C^+(\mathbb{I}\mathbf{u})\,\mathbb{I}\,\mathbf{u} - K^+\mathbb{I}\,\mathbf{u} + \mathbb{I}\,\mathbf{g} - G^+\mathrm{I}\,p \tag{2.92}$$

$$L^+\mathrm{I}\,p = -\tfrac{1}{\Delta t}D^+\mathbb{I}\,\mathbf{u}^* + \tfrac{1}{\Delta t}B^+\mathbb{I}(\mathbf{u}^* - \mathbf{u}_D) \tag{2.93}$$

Second, we recombine the linear equations according to the new unknowns, i.e., we multiply Eq. (2.92) with $\mathbb{I}^T$ and Eq. (2.93) with I^T:

$$\underbrace{(\mathbb{I}^T M^+\mathbb{I})}_{M}\,\dot{\mathbf{u}} = -(\mathbb{I}^T C^+(\mathbb{I}\mathbf{u})\,\mathbb{I})\,\mathbf{u} - \underbrace{(\mathbb{I}^T K^+\mathbb{I})}_{K}\,\mathbf{u} + (\mathbb{I}^T\mathbb{I})\,\mathbf{g} - \underbrace{(\mathbb{I}^T G^+\mathrm{I})}_{G}\,p \tag{2.94}$$

$$\underbrace{(\mathrm{I}^T L^+\mathrm{I})}_{L}\,p = -\tfrac{1}{\Delta t}\underbrace{(\mathrm{I}^T D^+\mathbb{I})}_{D}\,\mathbf{u}^* + \tfrac{1}{\Delta t}\underbrace{(\mathrm{I}^T B^+\mathbb{I})}_{B}(\mathbf{u}^* - \mathbf{u}_D) \tag{2.95}$$

This means that, effectively, we obtain a consistent formulation by determining the final FEM operators as $L := \mathrm{I}^T L^+\mathrm{I}$, $K := \mathbb{I}^T K^+\mathbb{I}$, $G := \mathbb{I}^T G^+\mathrm{I}$ etc. In practice, the initial FEM operators L^+, K^+, G^+ etc. can be assembled exactly as described previously in Alg. 2 and, similarly, Alg. 3 applies if these operators have to be applied to a vector and thus are not required in an explicit form. Fig. 2.5 (b) shows the grid index operator corresponding to the example octree in Fig. 2.5 (a), which is required in these two algorithms. Note that, in the case of octrees, the index operator also has to specify the size of each cell because element matrices typically depend on h. Both algorithms can be extended to include the interpolation operators I and $\mathbb{I}$, e.g., to assemble $L = \mathrm{I}^T L^+\mathrm{I}$ or to compute $Gp = (\mathbb{I}^T G^+\mathrm{I})\,p$. For this, the three matrices corresponding to interpolation, FEM operator and restriction (I^T and $\mathbb{I}^T$) either can be multiplied together explicitly or can be applied in sequence. In both cases, the sparseness and simplicity of I and $\mathbb{I}$ allows us to determine entries on the fly and thus to apply these two operators in-place.

The whole restriction process can also be interpreted as a change of basis functions, where the basis function at each hanging vertex is "attached" to the surrounding, non-hanging nodes, scaled with the corresponding constraint weights. For instance, in Fig. 2.5 (a), the three basis functions ψ_3, ψ_8 and ψ_{11} are replaced with only two basis functions $(\psi_3 + 0.5\psi_{11})$ and $(\psi_8 + 0.5\psi_{11})$, which are

associated with nodes N_3 and N_8, respectively. This removes the degree of freedom at the hanging vertex N_{11}^+ and extends the support of node N_3 to also cover e_4 and the support of N_8 to also cover e_2. If we change the basis functions in this way, both the discrete weak solution spaces P_h and $\mathbf{U}_h$ and the discrete test function spaces Q_h and $\mathbf{V}_h$ are affected (see Eqs. (2.36-2.39)). Accordingly, the replacement of p^+ with $\mathrm{I}p$ and $\mathbf{u}^+$ with $\mathbb{I}\mathbf{u}$ in Eqs. (2.90 + 2.91) is the consequence of changing P_h and $\mathbf{U}_h$, and, on the other hand, the multiplication of the resulting equations with $\mathbb{I}^T$ and I^T is the consequence of changing Q_h and $\mathbf{V}_h$.

2.6. Basic Multigrid Solver

We use a multigrid solver in order to efficiently solve the PPE in each timestep. While standard iterative methods for solving linear equation systems like, e.g., the *Gauss-Seidel method* or *conjugate gradients* quickly smooth out high-frequency errors in the solution, they have difficulties with removing low-frequency errors. Therefore, the idea of *multigrid* is to introduce a hierarchy of ever coarser grids, on which low-frequency errors from the original grid can be represented as high-frequency errors and thus can be handled efficiently using standard iterative methods. Particularly when applied to elliptic problems like the PPE, multigrid solvers are very well-known for their exceptional performance. In our method, we use a *geometric-algebraic* multigrid solver, which builds upon concepts originally developed by Dick et al. [DGW11, Dic12] for finite-element based elasticity simulations. We use geometric rules to generate coarse versions of the original grid and to exchange information between the different levels of the resulting multigrid hierarchy, and we use an algebraic scheme to generate the coarse grid operators in a generic way.

In this section, we introduce these core principles by introducing a basic version of our solver which is restricted to a non-adaptive Cartesian grid. In Chapter 3, we then describe an extended version of this solver which can be applied to octree grids and which can handle complex, fragmented domains by using the concept of *cell duplication*. For a more detailed introduction into multigrid theory we refer to, e.g., the book by Hackbusch [Hac85].

2.6.1. Coarse Grid Hierarchy

According to Alg. 1.5, the discrete PPE is given by

$$L_0 p_0 = \underbrace{-\tfrac{1}{\Delta t} D\mathbf{u}^* + \tfrac{1}{\Delta t} B(\mathbf{u}^* - \mathbf{u}_D)}_{=:\, b_0}, \tag{2.96}$$

where the index 0 is used to indicate the first (= finest) level of a multigrid hierarchy. In order to create coarse versions of the linear operator L_0, our first step is to construct a hierarchy of coarse

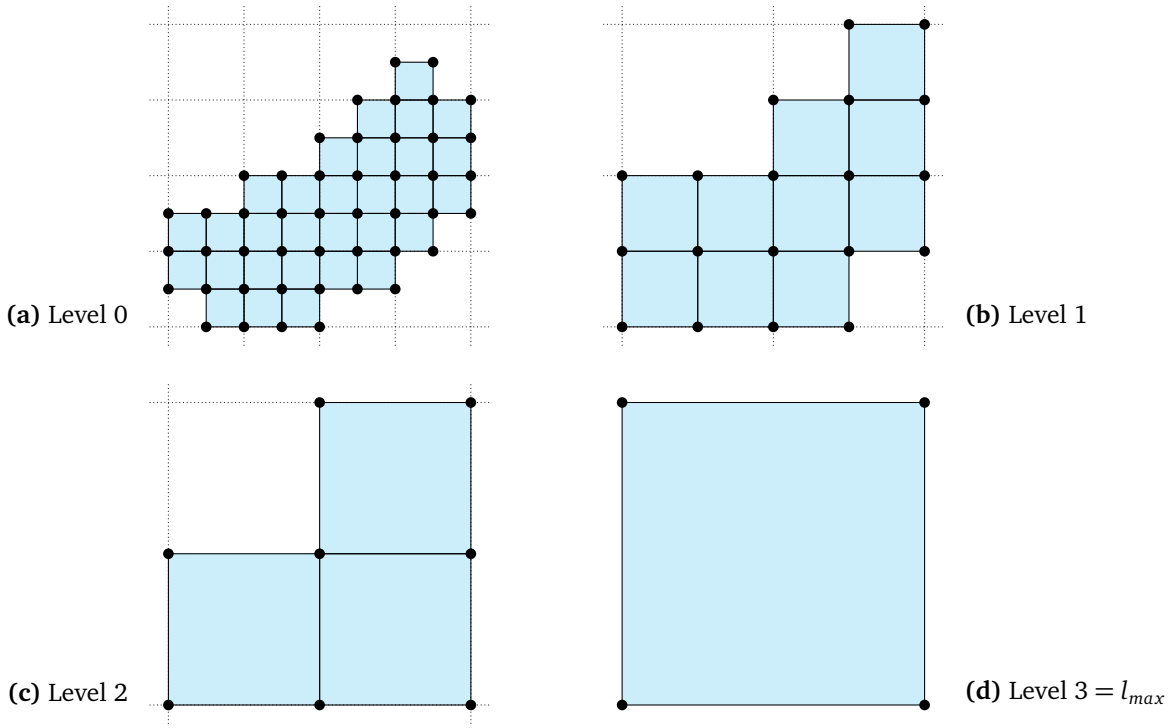


Figure 2.6.: Multigrid hierarchy in 2D. **(a)** Original Cartesian grid of square elements. **(b-d)** Each coarse level $l+1$ is created from the minimum set of cells which covers all cells of level l. The dotted lines indicate the imaginary lattice which is used in each coarsening step.

grids. For this, we start with the decomposition of our domain into a set of equally sized hexahedra of size h, which are cells of a Cartesian grid. Because we use trilinear basis functions for the pressure, the unknowns represented by p_0 are located at the grid's vertices, which constitute the finest level $l = 0$ of the grid hierarchy. It is important to note that the dimensions of the original grid are not required to be powers of two, and that the domain does not have to be rectangular. The shape of the original grid can be arbitrary like, e.g., the two-dimensional grid depicted in Fig. 2.6 (a).

Starting at the finest level 0, the coarse grids are created successively in an incremental fashion, up to a coarsest level l_{max}. In each step, to create the coarser grid on level $l+1$ from the finer grid on level l, we use a cell based coarsening approach in which cells of size $2^l h$ are merged into cells of size $2^{l+1}h$. We first imagine a coarsening lattice with vertices $(2^{l+1}h)\mathbb{Z}^3$, and determine all cells in this lattice which contain at least one cell of the finer level l. This yields the coarse cells on level $l+1$, to which we add all adjacent vertices of the imaginary lattice. The coarsening procedure can be repeated up to a level in which only one cell remains (or which cannot be reduced any further). In practice, however, we stop creating coarser grids as soon as the number of cells is smaller than 1000, because the remaining coarse problem can easily be solved directly at this point. The grid coarsening process is illustrated in Figs. 2.6 (b-d), which depict all possible coarse levels for the original grid

shown in Fig. 2.6 (a).

After the coarse grids have been generated, we algebraically compute coarse grid operators L_l (with $l > 0$) using *Galerkin based coarsening*. This requires us to define linear interpolation operators I_{l+1}^l, which describe how values are interpolated from the vertices of level $l+1$ to the vertices of level l, such that

$$p_l = I_{l+1}^l p_{l+1}. \tag{2.97}$$

We use trilinear interpolation weights for these operators, which is also referred to as *full-weighting* in the context of multigrid. Due to our cell based generation of the coarse grids, we can guarantee that, for each vertex of a finer level l, all vertices required for trilinear interpolation from the corresponding coarser level $l+1$ are always available. For the opposite direction, we are required to define restriction operators R_l^{l+1}, which describe how values are restricted from the vertices of level l to the vertices of level $l+1$. For Galerkin based coarsening, these have to be transposes of the interpolation operators, such that

$$p_{l+1} = R_l^{l+1} p_l \quad \text{with} \quad R_l^{l+1} := \left(I_{l+1}^l\right)^T. \tag{2.98}$$

Note that, while I_{l+1}^l is a proper interpolation operator and thus its rows sum up to one, the rows of the restriction operators R_l^{l+1} do not sum up to one in general. A re-normalization of these rows, however, is not required.

Based on the interpolation and restriction operators, the coarse grid operators are computed according to the following recursive rule:

$$L_{l+1} = R_l^{l+1} L_l I_{l+1}^l \tag{2.99}$$

This algebraic construction is the central principle of Galerkin based coarsening. Intuitively, it can be read as follows: An application of the coarse grid operator L_{l+1} on level $l+1$ should have the same effect as interpolating to the finer level l first, then applying L_l and then restricting back to level $l+1$. This construction ensures that complicated domain shapes and the boundary conditions for the pressure, which are embedded into the original system matrix L_0, are automatically inherited by the coarse grid operators. From a computational perspective, we can further state that the matrix product in Eq. (2.99) can be computed very efficiently, since it only involves sparse operators which correspond to small numerical stencils. In our case, L_0 corresponds to a stencil with $3^3 = 27$ entries, and I_{l+1}^l and R_l^{l+1} can also be represented by stencils of this size (with respect to the spacing of the finer grid level l). As a consequence, the stencils corresponding to the coarse operators L_l (with $l > 0$) are also limited to $3^3 = 27$ entries.

2.6.2. V-Cycle Algorithm

We incorporate the interpolation, restriction and coarse grid operators into a standard multigrid V-cycle scheme. The central update principle of this scheme can be derived in the following way: Suppose that we want to improve an intermediate solution or initial guess p_0^i for our linear problem $L_0 p_0 = b_0$ by using one or more coarse levels of the multigrid hierarchy. Then, we are searching for a correction term c_0^i to improve p_0^i:

$$L_0(p_0^i + c_0^i) = b_0 \tag{2.100}$$

This means that the correction term c_0^i can be found by solving the related problem

$$L_0\, c_0^i = \underbrace{b_0 - L_0\, p_0^i}_{=:\, r_0^i}, \tag{2.101}$$

where the unknown is c_0^i and the right-hand side is the residual r_0^i corresponding to our current solution p_0^i. Instead of solving Eq. (2.101) directly, we can solve it approximately using the first coarse level of the multigrid hierarchy. For this, we restrict the right-hand side to level 1 and solve the corresponding coarse problem:

$$L_1\, c_1^i = (R_0^1\, r_0^i) \tag{2.102}$$

Then, we interpolate the resulting coarse correction term c_1^i back to level 0. This yields an approximate correction term which we can use to improve the current solution p_0^i according to Eq. (2.100):

$$p_0^{i+1} := p_0^i + I_1^0 c_1^i \tag{2.103}$$

This basic two-level scheme can be extended to a full multigrid hierarchy in a recursive manner: The level 1 problem (Eq. (2.102)) can be solved approximately using multigrid level 2, yielding a level 2 problem which can be solved approximately using multigrid level 3, and so on. Since we do not have intermediate solutions on the coarse levels, like, e.g., an initial guess for c_1^i in Eq. (2.102), we start with an initial guess of 0 in these cases.

Altogether, this yields the standard V-Cycle algorithm, which is listed in Alg. 4. There, instead of using a separate variable, the correction terms on coarse levels $l > 0$ are directly stored in the variables p_l for the current solution. In addition to the recursive update scheme, which we have discussed above, a so-called *smoother* has to be applied on each level of the hierarchy to smooth out high-frequency errors. For this, on each level l, we perform n_{pre} Gauss-Seidel iterations before descending to the next coarser level, which is called *pre-smoothing*, and we perform n_{post} Gauss-Seidel

Algorithm 4 Multigrid V-Cycle Iteration

1: **for** l **from** 0 **to** $l_{max}-1$ **do** ▷ First "leg" of V-cycle: Descend from level 0 to l_{max}
2: **repeat** n_{pre} **times** $p_l := \text{GSITER}(L_l, p_l, b_l)$ ▷ Pre-smoothing using Gauss-Seidel iterations
3: $r_l := b_l - P_l\, p_l$
4: $b_{l+1} := R_l^{l+1}\, r_l$
5: $p_{l+1} := 0$
6: **end for**
7: $p_{l_{max}} := \text{CGSOLVE}(P_{l_{max}}, b_{l_{max}})$ ▷ Full solve on coarsest level using conjugate gradients
8: **for** l **from** l_{max} **to** 0 **do** ▷ Second "leg" of V-cycle: Ascend from level l_{max} to 0
9: $p_l \mathrel{+}= I_{l+1}^{l}\, p_{l+1}$
10: **repeat** n_{post} **times** $p_l := \text{GSITER}(P_l, p_l, b_l)$ ▷ Post-smoothing using Gauss-Seidel iterations
11: **end for**

iterations after returning from the coarser level $l+1$, which is called *post-smoothing*. This is denoted with the procedure "GSIter", which represents a single Gauss-Seidel iteration. Typically, useful values for n_{pre} and n_{post} are in the range from zero to five. After descending to the coarsest level l_{max}, the remaining problem can be solved using any direct solver. Here, we use a conjugate gradients solver, which is run until the solution converges to a sufficiently exact value. This is denoted with the procedure "CGSolve". A full V-cycle then corresponds to a single iteration of the multigrid solver. Before we start with the first V-cycle, we can use any existing initial guess to initialize p_0, or, if no initial guess is available, we can initialize p_0 with zeros. Like with any other iterative solver, a fixed number of iterations (i.e., V-cycles) can be performed or the solver can be run until the residual norm $||b_0 - L_0\, p_0||_2$ falls below a desired threshold.

3

Large-Scale Liquid Simulation on Adaptive Hexahedral Grids

In this chapter, we present our method for incompressible fluid simulation on adaptive octree grids, which is based on the FEM discretization and fundamental concepts introduced in the previous chapter. While this method is applicable to incompressible fluids in general, here, we specifically focus on the simulation of large-scale liquid phenomena for applications in computer animation. This means that the fluid domain is not only bounded by fixed Dirichlet and Neumann boundaries (the interface between the fluid and walls), but we additionally have to track the evolving fluid/air interface, which is a Neumann boundary. The equal-order interpolation scheme Q_1Q_1 is now applied on a restricted octree grid, where the hanging vertices are treated as indicated in Section 2.5, and we also extend the basic multigrid solver introduced in Section 2.6 to the octree setting. We further extend this solver by introducing the concept of cell duplication, which allows us to robustly handle fragmented domains on coarser levels of the multigrid hierarchy. Lastly, we also show how the treatment of boundary conditions can be realized with second-order accuracy by using a particular class of Nitsche methods, which requires adding appropriate penalty terms to the FEM weak form of the Navier-Stokes equations as derived in Section 2.2.1. We demonstrate the efficiency of the proposed method with large-scale liquid simulations in which the fluid would occupy up to 400 million grid cells if a non-adaptive grid was used. This corresponds to effective resolutions of up to $1024^2 \times 3072$.

This chapter is based on material that has been originally published in F. FERSTL, R. WESTERMANN, C. DICK: Large-Scale Liquid Simulation on Adaptive Hexahedral Grids. *IEEE Transactions on Visualization and Computer Graphics*, 20(10):1405-1417, 2014.

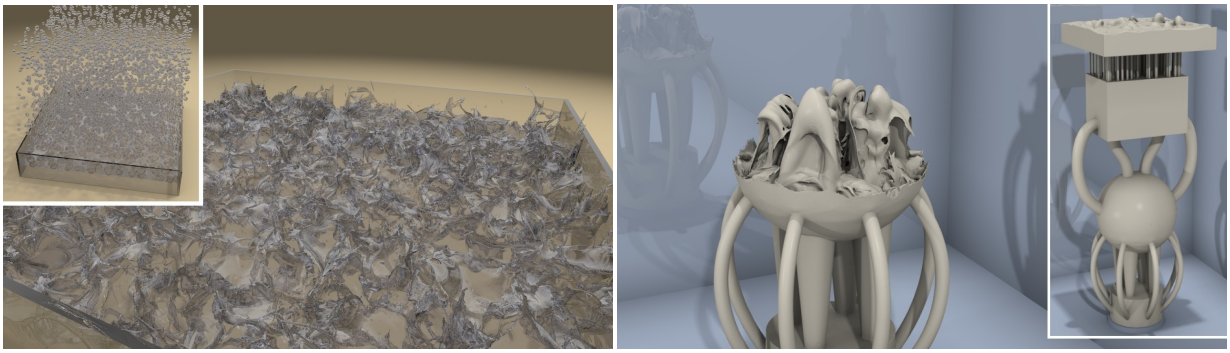


Figure 3.1.: Liquid flows using 34 **(left)** and 13 **(right)** million multi-resolution finite elements are simulated on an 8-core single node system with 64 GB main memory. A uniform hexahedral grid at the same effective resolution and restricted to the liquid domain would consist of about 130 million cells in both examples.

3.1. Introduction

Liquid simulation on regular grids has a long tradition in computer animation. Such techniques are attractive because of the implicit encoding of topology by the grid, giving rise to efficient computational kernels for simulating the liquid's motion. Especially when paired with efficient solvers for the pressure Poisson equation, such as linear time-complexity geometric multigrid solvers [MST10], the projection step for enforcing the incompressibility constraint on the velocity field is possible at high rates. By using data structures like OpenVDB [MLJ*13], regular grids can be dynamically restricted to the fluid domain, which reduces memory requirements in comparison to static grids that cover the whole simulation domain. This makes regular grids appealing in scenarios where violent flows with fragmentation and large deformations cover only a small area of the simulation domain. Since the number of computational elements is proportional to the liquid volume, in such scenarios a good ratio between resolution and memory is achieved.

On the other hand, keeping the number of computational elements proportional to the liquid body is not sufficient when large bodies of liquid are simulated at high resolution. This is demonstrated in Fig. 3.1, where even a regular grid restricted to the liquid domain requires about 130 millions of cells, imposing severe memory requirements. Losasso and co-workers [LGF04] addressed this problem via a first order accurate finite difference (FD) scheme on an adaptive octree grid. It supports an adaptive coarsening of the computational elements, yet it was shown recently that it introduces velocity oscillations due to the special treatment of hanging nodes at the coarse-to-fine grid interface [ATW13]. Zhu et al. [ZLC*13] introduced a FD scheme on a rectilinear grid that enlarges the cells in the far field surrounding a region of interest. This approach avoids hanging nodes, but it can less flexibly adapt the cell size locally due to the regular structure imposed by a rectilinear grid.

Unstructured grids, on the other hand, can adapt flexibly to local flow features and the changing

liquid domain [CFL*07, MEB*12, CWSO13, ATW13]. However, they cannot take full advantage of efficient hierarchical solvers because no canonical coarse versions of such a grid exist in general. Furthermore, additional workload is required to generate an unstructured grid from a given set of vertices or regular cells, and to build the algebraic equations from the unstructured discretization.

In this chapter we shed light on the question whether techniques based on regular grids can be realized in much the same adaptive, and thus memory efficient way as techniques based on unstructured grids, yet keeping the computational advantages of a regular grid. We focus on the simulation of liquids at extreme effective resolutions, and we therefore strive for a solution that a) uses computational elements only where the liquid is present, b) uses ever coarser simulation resolutions in the liquid body to achieve an element number proportional to the surface area, c) inherits the efficient computational kernels of regular grids, and d) is amenable to geometric multigrid solvers to achieve high convergence rates. To the best of our knowledge, grid based simulation approaches combining all these properties have not been proposed so far.

Contribution

To achieve our goals, we make the following specific contributions:

- We present a multi-resolution hexahedral finite element method (FEM) including a special treatment of hanging vertices, combined with a second order accurate representation of the free liquid surface.
- We make the number of finite elements proportional to the liquid boundary by using an octree grid that adapts spatially to the surface.
- We embed the finite element hierarchy into a geometric multigrid solver to achieve optimal convergence rates, and we use an adaptive variational formulation as well as element duplication to represent the domain at coarser scales.
- We propose a cell based formulation for both FEM and the multigrid solver by exploiting that most elements share the same generic element matrix. This formulation enables us to alleviate memory bandwidth limitations, leading to speedup factors between 2 and 3 compared to a stencil based formulation.

Some achievements of our method are demonstrated in Fig. 3.1. To demonstrate the different element resolutions that were used in the simulations, Fig. 3.2 illustrates the octree grid structure for a part of the multi-stage water fountain on a 2D slice. The liquid surface steers the evolution of the hierarchical grid over time (see Fig. 3.3). The surface is represented by means of a surface tracking method such as the particle level-set method by Enright et al. [ELF05].

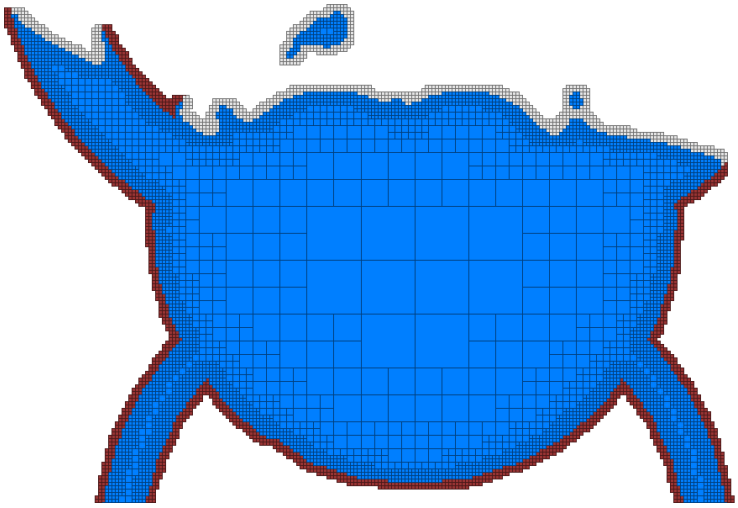


Figure 3.2.: Slice through the 3D octree grid of the multi-stage water fountain in Fig. 3.1 (right). A coarser version is shown for better readability. Cells are classified into fluid (blue), empty (gray), and solid (red). Note that the grid refinement is also affected by cells in front of and behind the shown layer of cells.

3.2. Related Work

Eulerian liquid simulation methods discretize the simulation domain using a fixed grid and compute the liquid's movement through this grid. In particular the use of regular hexahedral grid structures in combination with finite difference schemes to transform the governing equations into systems of difference equations has a long tradition in computer animation, see, for instance, [KM90, FM96, Sta99, FF01]. Let us also refer to the book by Bridson [Bri08], which provides a thorough overview of grid based approaches for liquid simulation. To efficiently generate detailed liquid surfaces from Eulerian simulations, surface tracking mechanisms making use of implicit level-sets and additional tracker particles [FF01, EFFM02, ELF05] as well as explicit approaches using semi-Lagrangian advection of distance functions [BGOS06] and surface meshes [WTGT09, BBB10, TWGT10] have been proposed. A good overview of the different tracking mechanisms used can be found in the course notes by Wojtan et al. [WMFB11].

Regular grids are appealing because they give rise to efficient numerical stencils and fast computational solvers. In particular multigrid methods [Hac85] have become popular in computer animation due their linear time-complexity in the number of computational elements. The potential of geometric multigrid schemes for projecting the velocity field to a divergence free state has been employed by McAdams et al. [MST10], and Chentanez and Müller [CM11a, CM11b]. Only few approaches have addressed adaptivity in liquid simulations using regular grids. Varying resolutions have been accommodated via an octree grid [LGF04], by enforcing incompressibility locally from a divergence-free field on a coarse resolution grid [LZF10], and by using simultaneously a coarse grid for represent-

ing the liquid body and a fine grid near the surface [CM11b]. The most recent approach by Zhu et al. [ZLC*13] makes use of a rectilinear grid to fade out the grid resolution from a prescribed focus region.

Instead of using regular grids, in a number of previous approaches unstructured grids have been used to enforce a sharp representation of the liquid interface and thus to effectively restrict the computational workload to the liquid domain [FOKG05, KFCO06, CFL*07, MEB*12, CWSO13]. The grids are either re-meshed to the liquid boundary in every simulation step, or they are used in arbitrary Lagrangian-Eulerian (ALE) methods where the grid vertices are moved according to the fluid motion [HAC97], eventually requiring re-meshing when deformations become large. Recently, Ando and coworkers [ATW13] proposed the integration of a tetrahedral background grid into FLIP. Here, a quasi regular, yet adaptive tetrahedralization was constructed from the dual of a multi-level hexahedral grid. FLIP [BR86, ZB05], is a hybrid method employing an auxiliary background grid for velocity projection and using particle based transport. We mention FLIP here because the use of an adaptive regular grid as proposed in this thesis is promising for reducing the number of required FLIP particles in the liquid interior.

3.3. Creeping Octree Grid

We start with a description of the spatially adaptive octree grid used in our method. The grid is constructed in a way that does not require any initial domain specification. It is sufficient to define the size h (i.e., the side-length) of the smallest hexahedral cells to be used in the simulation. All grid cells of size $2^k h$ are aligned on an imaginary lattice with lattice points $(2^k h)\mathbb{Z}^3$, where $k = 0, 1, \ldots$ is referred to as the octree level of the respective cells.

Based on these imaginary lattices, we build the grid such that only the liquid body is covered by grid cells, except for an additional thin refinement band around the liquid boundary (both the fluid/air and fluid/solid interface). The enclosing space is *not* represented. At the beginning of the simulation, the grid is constructed from a given surface describing the initial liquid body. During the simulation, in each time step, the octree grid is adapted to the current surface of the liquid, as described below. As a result, the simulation grid seems to 'creep' along with the liquid boundary, as illustrated in Fig. 3.3.

The adaptivity of the octree grid is controlled as follows. In a narrow band around the liquid boundary, which we refer to as the refinement band, we require all simulation cells to be at the finest resolution. The radius $r := \omega h$ ($\omega \in \mathbb{N}$) of the refinement band has to be specified. We enforce that the liquid boundary never leaves the refinement band, and thus is always represented on the finest resolution level. As a consequence, the maximum time step that can be simulated is coupled to r. While it is desirable in principle to make ω large in order to reduce time step constraints, this can

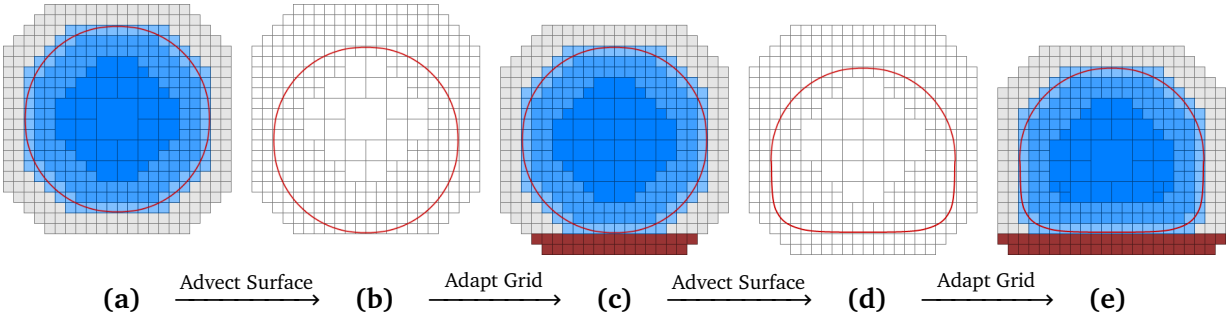


Figure 3.3.: Illustration of the evolution of the adaptive octree grid for the simulation of a falling water drop: **(a+c+e)** Grid and liquid boundary (red curve) in three consecutive time steps. Solid cells only appear in the grid when the liquid's surface comes close to an obstacle. Three different shades of blue indicate fluid cells intersected by the surface, fluid cells within the refinement band around the surface, and interior fluid cells. **(b+d)** Surface and grid (with unclassified cells) after surface advection, but before grid adaptation.

significantly increase the number of grid cells. We have found that $\omega = 2$ results in a good trade-off between time step constraints and memory requirements, and have used this value in all of our experiments. The interior of the liquid is filled by coarse cells, with the restriction that the level difference between neighboring cells sharing at least one common vertex is at most one. This results in a restricted octree grid as proposed by Popinet [Pop03]. In this way, the fluid interior is covered by a minimum number of cells, while the grid resolution decreases smoothly with increasing distance to the surface.

In the first simulation step, a restricted octree grid is built from an analytical or triangle mesh representation of the given surface of the initial liquid body, with the requirement that all cells intersected by the surface are on the finest octree level.

In each simulation step (including the first, in order to build the refinement band), the following steps are performed to adapt the grid to the current liquid boundary:

1. Creation of the refinement band: The set of cells intersected by the fluid surface is determined. The set is then successively extended in ω iterations, such that in each iteration the finest level cells in the 26-neighborhood of each cell of the set are added to the set. If in this process we encounter finest level cells that do not exist yet, we create them. If we encounter cells that are not on the finest octree level, we split them and apply further splits recursively to keep the octree grid restricted. In this way, we create one layer of the refinement band (on both sides of the liquid boundary) in each iteration. The resulting refinement band has a minimum width of $(2\omega + 1)h$ at any given point. Cells outside of the liquid body that do not belong to the refinement band are deleted.
2. Grid coarsening: As many cells as possible are merged in the fluid interior, while keeping the octree grid restricted.

3. Cell classification: The cells of the new grid are re-classified: Cells in the interior of solids are classified as *solid*, cells of which at least one vertex is lying inside the liquid boundary as *fluid*. All remaining cells are labeled *empty*. Note that per construction, all non-finest-level cells are fluid cells and lie completely within the liquid boundary. As an optimization, we utilize that all solid cells remain solid when the obstacles do not move.

The union of all fluid cells now defines the actual computational domain for the FEM discretization. The other cells are required only in the advection steps of the simulation. Consequently, all fluid cells and all vertices that are adjacent to at least one fluid cell (referred to as fluid vertices) carry a full set of simulation quantities. All other cells and vertices are light-weight and carry only a small subset of these quantities, in particular including extrapolated velocities and level-set values required for advection.

To represent the octree grid, an unbalanced octree data structure is used. Since we do not use an initially specified domain, the height of this octree varies over the course of the simulation according to the dynamically changing extent of the liquid.

Obstacles

Solid obstacles are handled in a similar way as proposed by Houston et al. [HNB*06] for generating a compact level-set representation. Obstacles are initially voxelized with respect to the imaginary finest level lattice such that a run-length encoded binary voxelization is created. This representation introduces a negligible memory footprint (compared to the final simulation grid and the additional data structures used by the solver), yet it allows us to efficiently test whether a finest level cell is inside a solid obstacle or not. It is important to note that the solids' voxelization is decoupled from the simulation grid, and that only those solid parts are represented in the simulation grid which are located in the refinement band around the liquid boundary.

3.4. Finite Element Fluid Simulation

To simulate the liquid, we solve for its motion on the fluid domain Ω as dictated by the incompressible Navier-Stokes equations. Here, we start with a brief summary of the fundamental concepts introduced in Chapter 2, and then present the required extensions for the boundary treatment and our multigrid solver. The continuous form of the Navier-Stokes equations is:

$$\dot{\mathbf{u}} = -\mathbf{u}\cdot\nabla\mathbf{u} + \frac{\mu}{\rho}\Delta\mathbf{u} + \mathbf{g} - \frac{1}{\rho}\nabla p \tag{3.1}$$

$$\nabla\cdot\mathbf{u} = 0, \tag{3.2}$$

where $\mathbf{u}$ is the fluid velocity, ρ the density, μ the viscosity, p the internal pressure, and $\mathbf{g}$ the gravity force. Note that ρ and μ are constant. We solve the pressure Poisson equation

$$\frac{1}{\rho}\Delta p = \frac{1}{\Delta t}\nabla \cdot \mathbf{u}^* \qquad \text{on } \Omega \tag{3.3}$$

$$p = p_A \qquad \text{on } \Gamma_A \tag{3.4}$$

$$\mathbf{n} \cdot \nabla p = \mathbf{n} \cdot \frac{\rho}{\Delta t}\left(\mathbf{u}^* - \mathbf{u}_W^{t+\Delta t}\right) \qquad \text{on } \Gamma_W \tag{3.5}$$

in each time step to project the intermediate velocity $\mathbf{u}^*$ to the space of divergence free functions according to $\mathbf{u}^{t+\Delta t} = \mathbf{u}^* - \frac{\Delta t}{\rho}\nabla p$ in order to enforce Eq. (3.2). $\Gamma_{A(ir)}$ and $\Gamma_{W(all)}$ denote the fluid/air and fluid/solid interface parts of the liquid boundary $\partial\Omega$, and correspond to the Dirichlet and Neumann boundaries introduced in Chapter 2, respectively ($\partial\Omega = \Gamma_A \uplus \Gamma_W$, $\Gamma_A = \Gamma_N$, $\Gamma_W = \Gamma_D$). On Γ_A the pressure p_A is prescribed as a Dirichlet boundary condition (Eq. (3.4)). On Γ_W, the velocity $\mathbf{n} \cdot \mathbf{u}_W^{t+\Delta t}$ in normal direction $\mathbf{n}$ to the boundary, and optionally the velocity $\tau \cdot \mathbf{u}_W^{t+\Delta t}$ in tangential direction τ to the boundary for the next time step $t + \Delta t$ is prescribed. The normal component is enforced via Eq. (3.5) as a Neumann boundary condition for the pressure. The optional tangential component can be enforced by setting $\tau \cdot \mathbf{u}^{t+\Delta t}$ to the respective value after each projection step [BCM01].

The liquid's surface is captured using the level-set method [OF03], i.e., the interface location is represented by the zero-contour of a signed distance function ϕ. Additionally, following [ELF05], we augment the level set with marker particles to improve mass conservation.

To obtain a consistent discretization on the spatially adaptive octree grid, we employ FEM as introduced in Chapter 2. Each *fluid* cell is a finite element. We use the equal-order interpolation scheme Q_1Q_1, where both $\mathbf{u}$ and p are discretized via element-wise trilinear basis functions. Accordingly, all degrees of freedom are located at the grid vertices. At the hanging vertices at octree level transitions, however, a special treatment procedure is required: Their values are constrained by linear/bilinear interpolation along the edge/within the face with respect to which the vertex is hanging. See Fig. 3.4 for an illustration, and see Section 2.5 for more details. The constraints ensures that the resulting weak solution is C^0 continuous across the entire domain. Furthermore, as will be demonstrated below, the specific choice of basis functions yields a discretization of the pressure Poisson equation that is exceptionally well suited for a multigrid scheme, as it results in very good convergence rates.

Let $\mathbf{v}$ and q denote the vector and scalar valued test functions corresponding to $\mathbf{u}$ and p, respectively. By testing Eq. (3.1) against $\mathbf{v}$ and Eq. (3.3) against q (i.e., by multiplying with the test function and

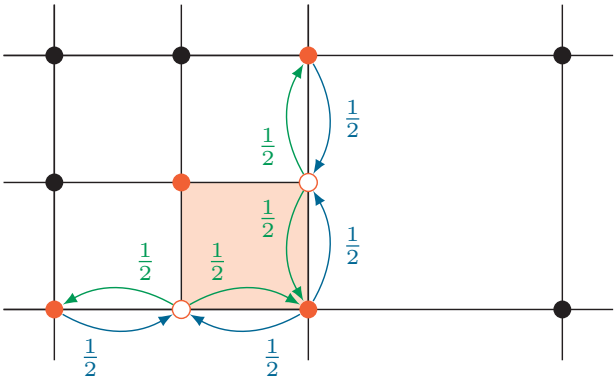


Figure 3.4.: Handling of hanging vertices (2D case), illustrated for the shaded element: The unknowns at hanging vertices (empty orange dots) are substituted (blue arrows) by linear interpolation from the unknowns at non-hanging vertices (filled orange dots). The equations at hanging vertices are then distributed (green arrows) to non-hanging vertices using the same weights, corresponding to the interpolation of values at hanging vertices in the test functions.

integrating over Ω), we obtain the following weak form of the continuous problem (see Section 2.2.1):

$$\int_\Omega \dot{\mathbf{u}} \cdot \mathbf{v} = -\int_\Omega \mathbf{u} \cdot \nabla \mathbf{u} \cdot \mathbf{v} - \int_\Omega \frac{\mu}{\rho} \left(\nabla \mathbf{u} : \nabla \mathbf{v}^T \right) + \int_\Omega \mathbf{g} \cdot \mathbf{v} - \int_\Omega \frac{1}{\rho} \nabla p \cdot \mathbf{v} \tag{3.6}$$

$$\int_\Omega \frac{1}{\rho} \nabla q \cdot \nabla p = -\frac{1}{\Delta t} \int_\Omega q \nabla \cdot \mathbf{u}^* + \frac{1}{\Delta t} \int_{\Gamma_W} q \mathbf{n} \cdot \left(\mathbf{u}^* - \mathbf{u}_W^{t+\Delta t} \right) \tag{3.7}$$

For a thorough overview of different FEM weak forms for the Navier-Stokes equations let us refer to the book by Gresho [GS00].

Applying the FEM discretization as described in Section 2.2.2 leads to the following equations ($\mathbf{u}$, p, etc. now are vectors containing the respective values at the grid vertices):

$$M\dot{\mathbf{u}} = -C(\mathbf{u})\mathbf{u} - K\mathbf{u} + \mathbf{g} - Gp \tag{3.8}$$

$$Lp = -\tfrac{1}{\Delta t} D\mathbf{u}^* + \tfrac{1}{\Delta t} B(\mathbf{u}^* - \mathbf{u}_W^{t+\Delta t}) \tag{3.9}$$

M is the (lumped) mass matrix, $C(\mathbf{u})$ the (FEM) convection operator, K the diffusion operator, G pressure gradient operator, L the Laplace operator, D the divergence operator, and B a boundary operator corresponding to the boundary term in Eq. (3.7).

To perform the time integration of the derived equations we use an explicit Euler scheme with operator splitting as proposed by Stam [Sta99]. The resulting procedure, which is performed in each time step, is listed in Alg. 5. This is the basic timestep algorithm derived in Section 2.3, extended to include the handling of the level-set ϕ and the adaptation of the octree grid.

The listing shows that the octree grid is adapted after the level-set has been advected, but before it is

Algorithm 5 Algorithm for time integration according to equations (3.8) and (3.9). *ID* denotes the identity matrix. Note that the mass matrix M is lumped, thus M^{-1} is a diagonal matrix.

1: $\phi^{t+\Delta t} := \text{Advect}(\phi^t)$
2: Adapt octree grid
3: Reinitialize $\phi^{t+\Delta t}$
4: $\mathbf{u}' := \text{Advect}(\mathbf{u}_t)$
5: $\mathbf{u}'' := \mathbf{u}' + \Delta t M^{-1}\mathbf{g}$
6: Solve for $\mathbf{u}^*$: $(ID + \Delta t M^{-1}K)\mathbf{u}^* = \mathbf{u}''$
7: Solve for p: $\Delta t\ Lp_t = -D\mathbf{u}^* + B(\mathbf{u}^* - \mathbf{u}_W^{t+\Delta t})$
8: $\mathbf{u}_{t+\Delta t} := \mathbf{u}^* - \Delta t M^{-1} G p_t$
9: Extrapolate $\mathbf{u}_{t+\Delta t}$

re-initialized. In this way, every new vertex that is introduced during octree adaptation is immediately assigned a correct value for ϕ, instead of an interpolated value. Level-set re-initialization is performed only in every tenth time step [OF03].

We use semi-Lagrangian advection to handle the advection of the level-set as well as the convective term ($C(\mathbf{u})$ is not required). During advection, trilinear interpolation is used to retrieve samples within the adaptive hexahedral grid. Furthermore, an implicit Euler scheme is used to handle the diffusive term with unconditional stability (see Section 2.3).

3.4.1. Second Order Accurate Pressure Boundary Conditions

The boundary conditions in Eqs. (3.4 + 3.5) for the pressure Poisson equation are handled via a particular second order accurate FEM discretization, while at the same time the symmetry of the resulting system matrix as required by the multigrid solver is preserved. For this method, the integration domain Ω in the weak form (Eqs. (3.6 + 3.7)) is restricted with subgrid accuracy to the part of the domain that is actually covered by the fluid (rather than being aligned to the hexahedral grid). Considering each single finite element, this means that the support of the basis functions is restricted to the portion of the element that is covered by fluid. Note that the finite element grid is not modified, and that the degrees of freedom of the finite elements remain at the grid vertices.

This restriction of the integration domain is sufficient to handle the pressure Neumann boundary condition on the fluid/solid interface Γ_W with subgrid accuracy. Thus, no special treatment as in finite difference schemes has to be performed, e.g., by using the variational approach by Batty et al. [BBB07].

In contrast, for the pressure Dirichlet boundary condition on the fluid/air interface Γ_A, further considerations are necessary. In finite difference schemes, this condition can be imposed with subgrid accuracy by using so-called ghost fluid methods [GFCK02]. These methods introduce additional ghost pressure values in the air, and linear dependencies among the pressure values around the boundary to

enforce the Dirichlet boundary condition at the actual surface position. However, when this procedure is directly applied to the FEM formulation (Eq. (3.7)), the boundary term cannot be restricted from Γ to Γ_W (as has been done in Eq. (3.7); cf. Section 2.2.1) since $q = 0|_{\Gamma_A}$ no longer holds, resulting in an asymmetric contribution to the system matrix.

A way to handle the pressure Dirichlet boundary condition on the fluid/air interface with subgrid accuracy that we have found to work very well in our formulation is the group of Nitsche methods [Nit71]. In these methods, all vertices of a finite element are full degrees of freedom, and a carefully chosen penalty term is added to the weak form such that the Dirichlet boundary condition is enforced at the actual surface position in a variational sense.

In particular, we have adapted the method developed by Dolbow et al. [DH09] for general Poisson problems, which achieves second order accuracy [HD10]. It maintains symmetry of the system matrix by constructing the penalty term such that the aforementioned asymmetric boundary term is counterbalanced. Leaving the detailed derivation and convergence proof to [DH09], we extend Eq. (3.7) according to:

$$\begin{aligned}&\int_\Omega \frac{1}{\rho}\nabla q\cdot\nabla p - \int_{\Gamma_A}\left(\frac{1}{\rho}q\mathbf{n}\cdot\nabla p + \frac{1}{\rho}p\mathbf{n}\cdot\nabla q - \alpha q p\right)\\ =&-\frac{1}{\Delta t}\int_\Omega q\nabla\cdot\mathbf{u}^* + \frac{1}{\Delta t}\int_{\Gamma_W} q\mathbf{n}\cdot(\mathbf{u}^* - \mathbf{u}_W^{t+1}) - \underbrace{\int_{\Gamma_A}\left(\frac{1}{\rho}p_A\mathbf{n}\cdot\nabla q + \alpha q p_A\right)}_{=0 \text{ if } p_A=0}\end{aligned} \tag{3.10}$$

In principle, Eq. (3.10) is constructed by testing Eq. (3.3) against q, Eq. (3.4) against αq, Eq. (3.4) against $-\frac{1}{\rho}(\mathbf{n}\cdot\nabla q)$ and then by summing up the resulting three equations. The method requires a constant stabilization parameter $\alpha = \frac{s}{\rho}\frac{\text{Area}(\Gamma_A)}{\text{Volume}(\Omega)}$ per cell, where $\text{Area}(\Gamma_A)$ is the area of the fluid/air interface and $\text{Volume}(\Omega)$ the volume of the fluid in the cell. s depends on the finite element type used and has been set to $s = 5$ in all of our experiments. If $p_A = 0$, the last term in Eq. (3.10) vanishes. Note that for cells which do not contain any part of Γ_A (i.e., which are fully covered by fluid), the weak form (Eq. (3.10)) is equivalent to Eq. (3.7).

In order to apply the extended weak form (Eq. (3.10)), in grid cells that are intersected by the fluid/air interface, the calculation of volume and surface integrals must be restricted to the fluid domain. For this purpose, we triangulate/tetrahedralize the boundary and the fluid domain in these cells, as illustrated in Fig. 3.5 (right). The integration problem has been similarly addressed in [MG07, HJWST12]. We then evaluate the surface and volume integrals using numerical quadrature on each of the resulting simplexes to obtain the respective element matrices.

For the triangulation/tetrahedralization, we have derived an extended, marching-cubes-style lookup table [LC87]. It provides precalculated triangulations/tetrahedralizations for any given configuration of level-set values (positive/non-positive) at the eight vertices of a fluid cell. Note that

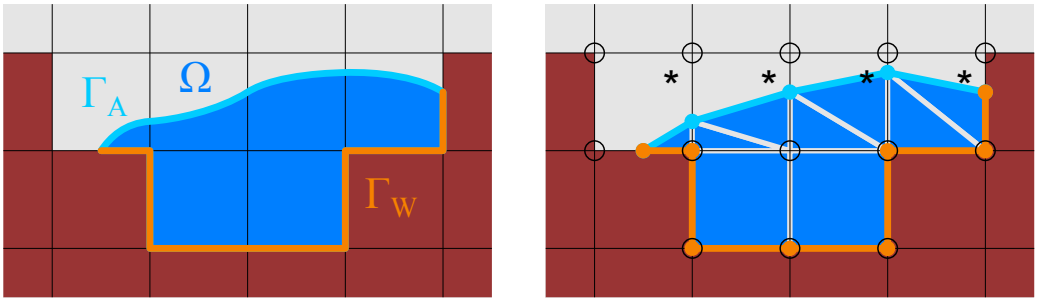


Figure 3.5.: Subgrid accurate treatment of boundary conditions (for illustration purposes shown in 2D). **(left)** The computational domain Ω is restricted to the portion of the grid cells that is actually covered by the fluid. **(right)** In cells intersected by the fluid/air interface (cells marked with a star), Ω, Γ_A, and Γ_W are tetrahedralized/triangulated by employing a precomputed lookup table. The degrees of freedom of the finite elements (black circles) remain located at the grid vertices.

in our implementation, solids are represented on a per-cell basis. Let $\Omega_c \subset \Omega$ denote the part of a cell's domain that is covered by the fluid. Then, for each configuration, the lookup table contains the following three components:

- A set of triangles that triangulates the fluid/air interface $\partial\Omega_c \cap \Gamma_A$ within the cell (native marching-cubes).
- For each of the six cell faces a set of triangles that triangulates the potential fluid/solid interface $\partial\Omega_c \setminus \Gamma_A$ (used only if a respective neighbor cell is solid).
- A set of tetrahedra that tetrahedralizes the fluid domain Ω_c within the cell (up to ten tetrahedra are required per cell).

The lookup table is constructed such that the triangulation and tetrahedralization are consistent, i.e., each triangle always corresponds to a face of a tetrahedron. In order to avoid near-zero values in the system matrix (due to cells containing almost zero fluid), we enforce the generated vertices on the edges of a cell to have a minimum distance of $0.01h$ to the cell's vertices inside the liquid body. Note that any remaining degenerated triangles/tetrahedra do not cause problems, because their contribution to the integral in Eq. (3.10) vanishes. For the numerical quadrature on the triangles and tetrahedra, we use a three-point Gaussian and an eight-point Newton-Cotes quadrature rule [Str71], respectively. For the evaluation of the surface integrals, the face normals of the triangles are used as the normal of the boundary.

The simple example in Fig. 3.6 demonstrates the accuracy achieved by our approach. Considering the performance, the subgrid accurate treatment of boundary conditions does not have a significant impact on the total computing time of our method. The time required for the computation of the element matrices for the cells that are intersected by the fluid/air interface is typically less than 5%

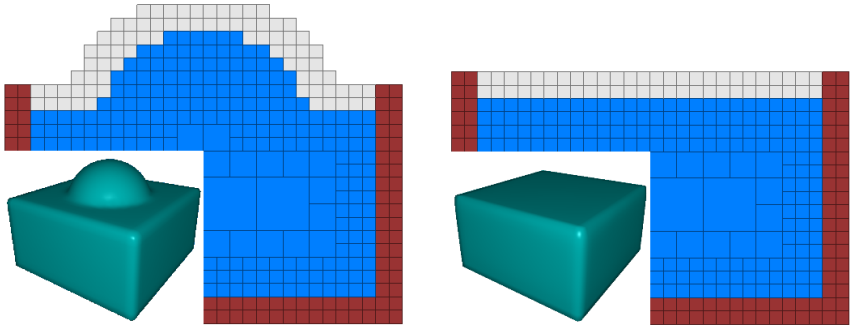


Figure 3.6.: Demonstration of second order accurate boundary conditions for a perturbed liquid in a box: A bump in the surface **(left)** is smoothed out after several seconds of simulation **(right)**. The resulting surface is perfectly planar. Note that the surface is tracked by a level-set without particles in this experiment, and that the blue color indicates fluid cells, rather than depicting the actual fluid domain.

of the time required for one complete time step. From an implementation point of view, it is worth noting that the computation of the element matrices, including triangulation/tetrahedralization and quadrature can be performed independently for each cell, and thus can be easily parallelized.

3.5. Multigrid Solver

The pressure Poisson equation is solved in each time step with an efficient geometric-algebraic multigrid solver, which exhibits linear time-complexity in the number of unknowns [Hac85]. Its efficiency is crucial, since with increasing grid size, the pressure solve quickly becomes the major performance bottleneck when less efficient solvers are used.

A major challenge in the design of geometric multigrid solvers is the treatment of complex shaped domain boundaries (here, the *fluid* domain boundary) on the successively coarser scales. In contrast to common multigrid schemes on simple shaped (e.g., rectangular) domains, we have to use modified processes for the construction of the coarse grid hierarchy and the algebraic operators on these grids. Our scheme relies on a specific strategy to generate the coarse grids incrementally from the fine grid, and uses a variational principle to generate the transfer and coarse grid operators. In this way our solver can handle complex fluid boundaries in a highly efficient manner.

For the octree setting, we extend the basic, non-adaptive solver introduced in Section 2.6. In particular, we address a problem that specifically arises when the simulation domain is composed of multiple (locally) separated components (for instance, separated fluid branches, drops or splashes). In such situations, separated domain components might get represented by the same coarse grid cell, when a canonical coarse grid hierarchy is used. As a consequence, the current error cannot be represented very well on the coarse grids, which in turn causes the multigrid convergence to break down. This is demonstrated in Section 3.7. We address this problem by duplicating coarse grid cells

at the same geometric position, with each instance corresponding to a separated fluid component, similar to [ABA00, TSB*05, NKJF09, CCG11]. Conceptually, we build a distinct coarse grid for each separated fluid component, with the coarse grids being allowed to overlap. This strategy alleviates the convergence problem and introduces only little extra cost when building the multigrid hierarchy.

In each simulation step, the multigrid hierarchy is rebuilt and the corresponding coarse grid operators are recomputed. Note that the multigrid hierarchy is completely separate from the octree data structure that is used for the representation of the adaptive octree grid. Cell duplication is controlled by means of a connectivity graph G_l per multigrid level, which is constructed on the finest multigrid level $l = 0$ from the set of fluid cells, and on the coarser levels $l = 1, \ldots, l_{max}$ by propagating the previous level graph to the current level along with the grid. The nodes of each graph G_l correspond to the cells of the grid on the respective level, and edges in the graph represent physical connections among these cells. Edges only exist between nodes corresponding to adjacent cells that share at least one common vertex. Since the graphs are only required during the construction of the hierarchy, they do not need to be stored permanently.

Multigrid hierarchy construction starts by creating the finest level connectivity graph G_0. For each *fluid* cell a node is created, and two nodes are connected if the respective fluid cells share at least one common vertex. After initialization of the finest level graph, we build the multigrid hierarchy successively in a fine-to-coarse process. This process stops at the coarsest level l_{max}, as soon as the remaining number of cells is less than 1000. In the coarsening procedure, we have to consider that the finest level grid is an adaptive octree grid consisting of cells of different size. On the transition from multigrid level l to level $l + 1$, we therefore combine only cells of size $2^l h$, but copy all larger cells to the next level.

The construction of multigrid level $l + 1$ is divided into the following three steps:

1. Creation of coarse grid cells: Considering an imaginary lattice with lattice points $(2^{l+1}h)\mathbb{Z}^3$, for each cell of this lattice, we determine the connectivity components of the subgraph of G_l induced by the grid cells of size $2^l h$ within this lattice cell. For each connectivity component, a coarse grid cell of size $2^{l+1}h$ is created, which subsumes the grid cells corresponding to the respective subgraph. In this way, multiple coarse grid cells might be created at the same location. Then, all grid cells with size $\geq 2^{l+1}h$ are copied from the level l to level $l + 1$.
2. Propagation of the connectivity graph: G_{l+1} is obtained by collapsing each subgraph of G_l that corresponds to a set of merged cells into a single node.
3. Creation of shared vertices on the coarse grid: At each vertex position, the subgraph of G_{l+1} corresponding to all cells that are incident to this position is investigated. A shared vertex is generated for each connectivity component of this subgraph.

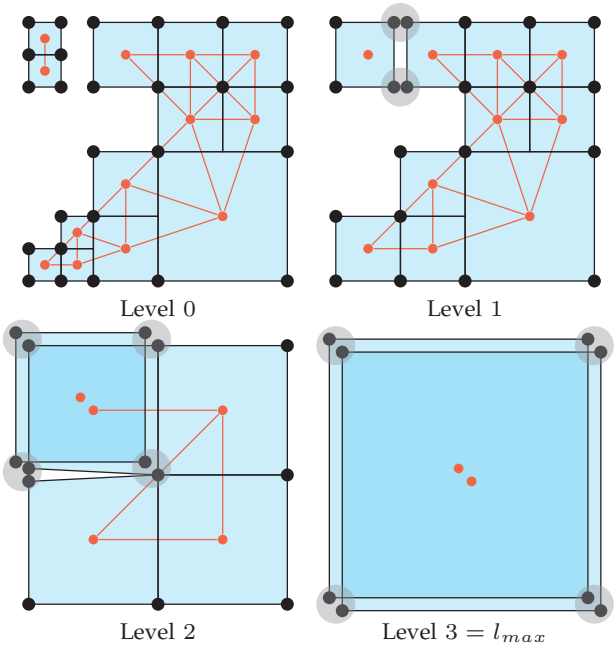


Figure 3.7.: A 2D example for a multigrid hierarchy. Circles shaded gray indicate duplicate vertices sharing the same geometric location, overlapping cells indicate duplicate cells. Cell connectivity graphs G_l are depicted in red. Note that for demonstration purposes, the level 0 grid does not strictly adhere to the rules specified in Section 3.3, i.e., not all boundary cells are on the finest octree level.

An example illustrating the coarse grid hierarchy construction according to the specified rules is demonstrated in Fig. 3.7. Hanging vertices are not shown, because they are eliminated before the multigrid hierarchy is constructed, as described in Section 3.6. It is worth noting that due to the particular coarsening strategy, no additional hanging vertices are created in the coarse grid. Furthermore, it can be observed that duplicate vertices only occur on levels $l \geq 1$ and duplicate cells only on levels $l \geq 2$.

The transfer and coarse grid operators are constructed as follows: We use full-weighting for the interpolation operators I_{l+1}^{l}, i.e., values on level l are trilinearly interpolated from values on level $l+1$. More precisely, considering the occurrence of duplicate coarse grid cells in our advanced multigrid hierarchy, each cell is merged into exactly one coarse cell on the next coarser level, and the cell's vertices exactly interpolate from and restrict to this coarse cell's vertices (see Fig. 3.8). It is important to note that this scheme also works at the fluid boundary where coarse cells are only partially 'filled' by cells on the finer level, in that the constructed interpolation operator always has full rank.

Starting from the linear equation system $Lp = b$ on the finest level (Eq. (3.9)), the restriction

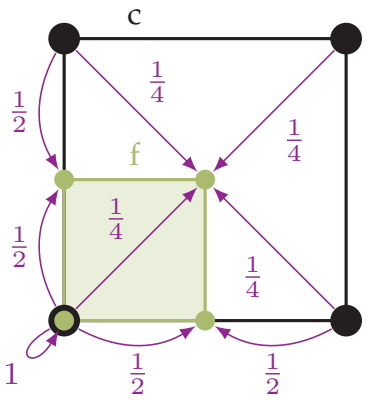


Figure 3.8.: Galerkin based coarsening with our cell based formulation (2D case): The vertices of a fine grid cell f (green) interpolate from the vertices of its associated coarse grid cell c (black) using linear/bilinear weights (purple). The matrix of a coarse grid cell is computed as a sum of matrices derived from its associated fine grid cells. The contribution of each fine grid cell is obtained by substituting the fine grid unknowns by interpolation from the coarse grid unknowns, and distributing the equations from the fine grid vertices to the coarse grid vertices using the same weights.

operators R_l^{l+1} and coarse grid operators L_l ($L_0 \equiv L$) are built from fine to coarse according to

$$R_l^{l+1} = (I_{l+1}^l)^T, \tag{3.11}$$

$$L_{l+1} = R_l^{l+1} L_l I_{l+1}^l, \tag{3.12}$$

with Eq. (3.12) being known as Galerkin based coarsening. Note that the construction of the operators is purely algebraic, i.e., boundary conditions are automatically incorporated on the coarse levels. The construction of the operators follows a variational principle [BHM00], in that by applying the computed coarse grid correction, the error is minimized. In particular, in combination with a converging smoother such as Gauss-Seidel relaxation, it can be shown that the norm of the error is monotonically decreasing with each further iteration of the solver, i.e., the convergence of the solver is guaranteed [TOS01]. As a side note, let us mention that from a mathematical point of view, the resulting coarse grid operators can be interpreted as composite FEM discretizations [LPR*09] of the continuous problem on the coarse grids.

Our solver traverses the multigrid hierarchy according to the standard V-cycle scheme. On each level of the hierarchy, we perform two pre-smoothing and two post-smoothing (multi-color) Gauss-Seidel relaxation steps. On the coarsest level l_{max}, the linear system is solved using a Jacobi-preconditioned conjugate gradient solver. The multigrid solver can be used as a direct solver as well as a preconditioner for a conjugate gradient solver, which improved its performance even further in our experiments. When used as a preconditioner, we perform one V-cycle per CG iteration.

In our experiments, we stop the multigrid solver when the norm of the residual $r = b - Lp$ has

been reduced by a factor of 10^{-6} (i.e., when $\|r_i\|_2 \,/\, \|r_0\|_2 \leq 10^{-6}$, where r_i denotes the residual after the ith V-cycle or CG iteration).

3.6. Implementation Details

As a consequence of the increasing gap between CPU and memory speed, main memory access has become the major bottleneck in numerical simulation on today's PC architectures, both in terms of limited memory throughput and high memory access latencies.

In grid based fluid simulation based on the projection method, computing time is known to be dominated by the run-time of the linear solver for the pressure Poisson equation. In particular, for a geometric-algebraic multigrid solver, the most frequent and most time-consuming operations are the pre- and post-smoothing relaxation steps as well as the residual computation step on each level of the multigrid hierarchy. In each of these steps, the numerical stencil at each vertex, consisting of (approximately) $3^3 = 27$ floating point values, has to be accessed, leading to a significant amount of memory traffic when this stencil is fetched from main memory.

Cell based FEM and Multigrid Formulation

To alleviate the aforementioned memory bottleneck, our implementation is built upon a cell based formulation of FEM and the geometric multigrid solver. This cell based formulation is leveraged by our particular design choices of using a hexahedral discretization, trilinear finite elements and Galerkin-based coarsening. More specifically, these choices allow us to compute the FEM operators, including in particular the Poisson operator and its corresponding multigrid coarse grid operators on a per-cell basis (rather than a per-vertex basis). During the computation, the numerical stencil at each vertex is then assembled on-the-fly from the matrices of the incident cells. Due to the regular hexahedral shape of the elements in our adaptive grid, for each FEM operator, (almost) all elements share the same generic element matrix (except for scaling with the cell size). These generic element matrices are precomputed analytically (as described in Section 2.4). On the finest level, the only exception are elements that are intersected by the fluid/air interface. Only for those we numerically compute and store the non-generic element matrices resulting from the second-order accurate boundary conditions.

For the coarse grid versions of the Poisson operator, the matrix of each cell is assembled from the matrices of its associated fine grid cells by means of Galerkin based coarsening (see Fig. 3.8). This matrix is equal to the generic Poisson operator element matrix (except for scaling with the cell size), if (a) the cell is completely 'filled' by fine grid cells and (b) all of these cells' matrices are generic. As a consequence, on the coarse levels, we only have to compute and store the matrices of those cells for which one of these two conditions is not met.

Since the generic element matrices can be assumed to reside in cache memory, the amount of main memory traffic is reduced considerably. Compared to an implementation where the numerical stencil at each vertex is fetched from main memory, we observe a speedup factor for the multigrid solver between 2 and 3—despite of the increased number of floating point operations due to the on-the-fly assembly of the stencil. As an additional benefit, the maximum memory footprint during the course of a time step is reduced by 20% to 30%.

It is worth noting that the size of the 27-point (in average) FEM Poisson stencil should not be considered a major performance drawback per se (compared to a size of only 7 for the commonly used uniform grid FD Poisson stencil). Because the performance bottleneck is the memory throughput resulting from fetching pressure values rather than arithmetic throughput, the large vertex overlap among neighboring stencils alleviates the impact on performance to a certain extent. This is especially true in a parallel implementation. Consider a uniform grid on a rectangular domain: Although the FEM stencil is larger than the FD stencil by a factor of $27/7 \approx 4$, the memory throughput increases only by a factor of $9/5 \approx 2$ (the size ratio of the stencils' 2D footprints) when iterating over the vertices of the grid due to cache coherence. In a simple experiment, we could actually verify this factor of roughly 2 between FEM and FD on our test system.

To efficiently handle hanging vertices occurring in the adaptive hexahedral discretization, we eliminate these vertices directly on the finest level (see Section 2.5), similar to the approach proposed in [Wan00]. This is achieved by observing that a hanging vertex is lying inside an edge or face of a neighboring cell, with the unknown at the hanging vertex being determined by linear or bilinear interpolation from the edge or face's vertices. Note that in a restricted octree, these vertices are guaranteed to be non-hanging. By replacing all hanging-vertices with the corresponding non-hanging vertices, each cell can always be associated with 8 non-hanging vertices (which may be different from the cell's geometric vertices). The vertex/cell incidence relationship is then determined accordingly. To obtain a cell's element matrix from the generic element matrix, we first substitute the unknowns at the cell's hanging vertices with the respective unknowns at its associated non-hanging vertices, and then distribute the equations at the hanging vertices to the non-hanging vertices, using the same weights as are used for linear/bilinear interpolation (see Fig. 3.4). Since there are 8 possible positions of a cell with respect to its (imaginary) parent cell in the octree, and since for each position at most 6 of the 8 vertices can be hanging, there are at most $8 \cdot 2^6 = 512$ hanging vertex configurations, corresponding to at most 512 adapted generic element matrices. These matrices are precomputed and stored in a lookup table, which is then used to assemble the numerical stencils on-the-fly. The elimination of hanging vertices leads to a speedup factor of about 2 compared to continuously handling the hanging vertices on-the-fly during the computation.

Parallelization

To exploit the performance available on today's multi-core CPU architectures, we have parallelized all parts of our algorithm where cells or vertices can be processed independently. Only the adaptation of the octree grid, the construction of the multigrid coarse grid hierarchy (not including the computations of the coarse grid operators), and the re-initialization of the level set function using a fast marching technique [OF03] are running sequentially. In terms of run-time on a single core, more than 90% of the algorithm are parallelized.

For the parallelization of the multi-color Gauss-Seidel relaxation step, a vertex coloring defining subsets of independent vertices is required. For our adaptive octree grid, we construct this vertex coloring independently for each multigrid level l as follows: For each vertex, we define its level as the maximum octree level of its incident cells. Considering each subset of vertices on the same level l_v ($l \leq l_v \leq l_{\max}$), these vertices are lying on a uniform grid with spacing $2^{l_v}h$, and none of the vertices is incident to a cell larger than $2^{l_v}h$. Therefore, this subset can be further partitioned into subsets of independent vertices using 8 colors. As a consequence, on multigrid level l, at most $8(l_{\max} - l + 1)$ colors are required to partition the vertices into subsets of independent vertices.

All per-cell and per-vertex quantities are stored in linear arrays, which are accessed by means of cell and vertex indices. In order to allow for an efficient, parallel 'traversal' of the cells and vertices of the adaptive octree grid and the coarse grid hierarchy, the cells and vertices of each level are consecutively enumerated in a particular order, which is determined as follows. First, all fluid cells and fluid vertices are located prior to non-fluid cells and vertices, respectively. Considering that the non-fluid cells and vertices carry a reduced set of simulation quantities, this allows us to store all quantities contiguously in memory. Second, considering the parallelization of the Gauss-Seidel relaxation, the fluid vertices on each multigrid level are ordered according to their vertex color as defined above. In this way, the simulation quantities of each vertex subset are lying consecutively in memory. To maintain this particular order after adaptation of the octree grid, we re-enumerate the cells and the vertices in each simulation step, and re-order the linear arrays of per-cell and per-vertex quantities accordingly.

3.7. Results

To demonstrate the effectiveness of our approach, we have run two large-scale simulations with up to 34 million elements, which are shown in Fig. 3.9. In addition, we have analyzed the convergence behavior of our solver in several scenarios and compared it to a conjugate gradient solver. All simulations and tests in this chapter were carried out at double floating point precision on a single workstation equipped with two Intel Xeon E5-2643 processors with a total of eight cores running at 3.3 GHz and 64 GB of RAM.

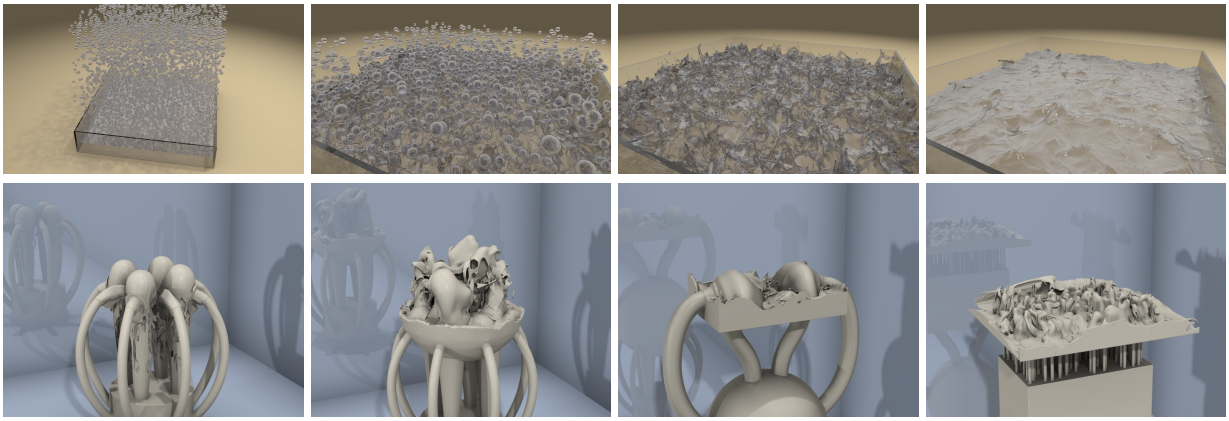


Figure 3.9.: Two large-scale liquid simulations: **(top)** 4000 drops of water fall into a rectangular basin. Each drop has an extent of up to 30 finest octree level cells. **(bottom)** A liquid is pumped into a multi-stage water fountain.

Convergence Study

In Fig. 3.10 we showcase the performance of our multigrid solver for two scenarios exhibiting different complexities of the simulation domain. In particular, we compare our multigrid solver, used as a direct solver (MG) and as a preconditioner for the conjugate gradient method (MGCG), to conjugate gradient solvers with Jacobi (JCG) and incomplete Cholesky IC(0) preconditioners (ICCG). To demonstrate the effectiveness of our cell duplication strategy in the construction of the coarse grid hierarchy, we have also included the convergence behavior of our multigrid solver using a standard coarse grid hierarchy, which is obtained by merging cells purely based on their geometric position without considering the connectivity between cells (cf. coarse grid construction in Section 2.6). Multigrid variants with cell duplication are marked with a $^+$, whereas variants without cell duplication are marked with a $^-$. In order to guarantee a fair comparison, the CG solvers are optimized for low memory traffic in exactly the same way (matrix-free) as our MG solver based on the on-the-fly assembly of the numerical stencils, and also run in parallel. For the IC(0) preconditioner, forward and backward substitution are parallelized using the existing vertex coloring scheme. Furthermore, we have included the initialization times of the solvers (e.g., the construction of the coarse grid hierarchy and the computation of the coarse grid operators for the multigrid solvers). Note that the MG solvers without cell duplication require slightly less time for initialization, because no connectivity graphs have to be built. When analyzing the IC(0) preconditioned CG solver, it is also important to note that in our scenario each CG iteration with IC(0) takes almost three times longer than with its Jacobi counterpart. As a consequence, although IC(0) requires less than half of the iterations, the achieved solver times are roughly identical.

In the first scenario ('Simple Domain') we solve for the pressure in a solid cube that is almost

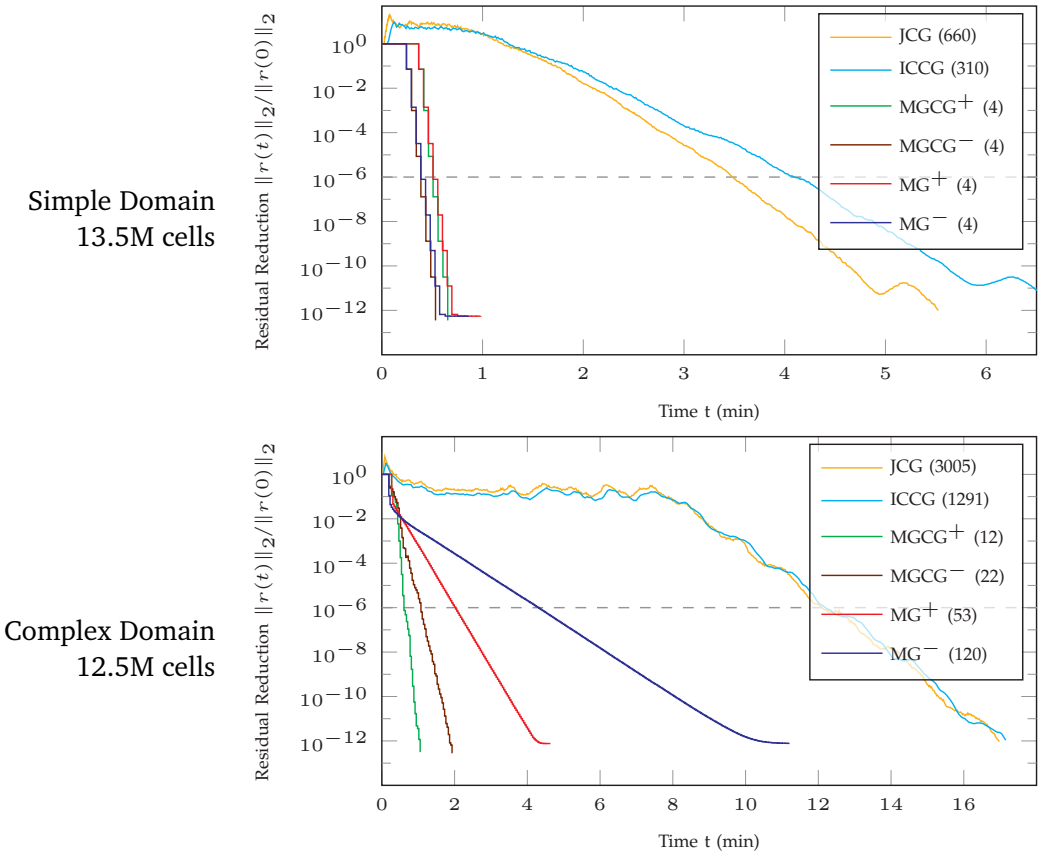


Figure 3.10.: Solver convergence over time. In parentheses are given the number of iterations (V-cycles or CG iterations) that were required to achieve a residual reduction by 10^{-6} (dashed line). As can be seen, the MGCG^{+} solver outperforms the JCG/ICCG solvers roughly by a factor of 8 to 12. Furthermore, the second example demonstrates the necessity of the cell duplication strategy in complex scenarios, where we can observe a speedup of almost a factor of 2 when comparing MGCG^{+} to MGCG^{-} and MG^{+} to MG^{-}.

completely filled with liquid. Since the liquid covers a convex domain, no cell-duplicates are created. Therefore, the coarse grid hierarchy generated by our approach is identical to a standard hierarchy, thus the direct MG solvers exhibit the same stable convergence rate of $\|r_{i+1}\|_2 \,/\, \|r_i\|_2 = 0.03$. The very high convergence rate is not further increased by using the MG solver as a preconditioner for the CG solver. As a consequence of the slightly higher initialization time required for creating a coarse grid hierarchy with connectivity analysis, in this scenario the MG solvers using the standard hierarchy are slightly faster. Compared to the commonly used Jacobi and IC(0) preconditioned CG solvers, all MG variants are about 6 times faster.

The second scenario ('Complex Domain') corresponds to the pressure solve in the fully filled fountain scenario shown in Fig. 3.1. In contrast to the first scenario, now our advanced coarse grid hierarchy exhibits a significant number of cell duplicates due to the complex shape of the fluid domain,

formed by a multitude of fine branches and separated domain fragments. For this scenario, our direct MG solver achieves a stable convergence rate of 0.81 and 0.92 for the advanced and the standard coarse grid hierarchy, which clearly demonstrates the effectiveness of our cell duplication strategy. Note that the decrease of the convergence rates compared to the first scenario is fully in-line with multigrid theory, considering the very complex shape of the simulation domain. Here, when using MG as a preconditioner, the convergence behavior is significantly improved, leading to an average residual reduction of 0.33 and 0.53 per CG iteration, respectively. Compared to the standard Jacobi and IC(0) preconditioned CG solvers, the MG preconditioned CG solver is 12 times faster.

Since the performance advantages of MG over CG are generally weaker at smaller resolutions, we have further tested the second scenario at several smaller resolutions. Here, MG as a preconditioner still proved to be significantly faster, e.g., by a factor of 6 at a resolution of 1M cells, and a factor of 3 at a resolution of 250k cells.

Performance Analysis

Table 3.1 lists timings and memory footprints for distinct time steps for the scenes depicted in Figs. 3.1+3.9. The first column '#Fluid Cells' shows the number of fluid cells in the respective state of the adaptive octree grid. To demonstrate the efficacy of the adaptiveness of the grid, we have also included in parentheses the number of fluid cells that would be required without coarsening in the liquid interior. Our experiments demonstrate (last row of the table) that by using the octree grid, the number of fluid cells is reduced by up to 90% compared to using an uncoarsened octree grid. 'Max. Domain Res.' indicates the resolution of a uniform hexahedral grid that would have to be used instead of our creeping grid in order to run the simulation using the same cell size as in the refinement band around the liquid boundary. In the next columns, we specify the total computation time T_{Total}, which is composed of the time spent for level set advection and velocity extrapolation (T_ϕ), adaptation of the octree grid (T_{Octr}), the projection step including construction of the multigrid hierarchy, the construction of the coarse grid operators, and the pressure solve (T_{Proj}), and the time spent in the remaining parts of the simulation (T_{Other}). In the remaining columns, we specify the maximum memory footprint of our simulation during the course of the time step ('Total'), which is composed out of three major parts. First, the memory required for all simulation quantities stored on a per-cell or per-vertex basis ('Sim'), such as u, u^*, p, ϕ, the cell classification flags, the non-generic matrices of the FEM and coarse grid operators, as well as the required temporary CG variables of the MG preconditioned CG solver. Second, the memory required for the topological encoding of the coarse grid hierarchy of the multigrid solver ('MG'), not including the finest level (i.e., the adaptive octree grid). Third, the memory required for the topological encoding of the adaptive octree grid ('Octree').

Table 3.1.: Number of elements, timings and memory footprints for distinct time steps of the two simulations depicted in Fig. 3.9. From top to bottom, these time steps correspond to the images in Fig. 3.9 (top row, 1st), Fig. 3.1 (left), Fig. 3.9 (top row, 4th), Fig. 3.1 (right) and Fig. 3.9 (bottom row, 4th).

Scene (Time Step)	#Fluid Cells (Uncoarsened)	Max. Domain Resolution	Timings (min)				
			T_{Total}	T_{ϕ}	T_{Octr}	T_{Proj}	T_{Other}
Drops (first step)	30.5M (131M)	1024^3	4.25	1.28	1.16	1.41	0.40
Drops (longest step)	33.3M (129M)	1024^3	5.10	1.77	1.38	1.46	0.49
Drops (last step)	12.0M (124M)	1024^3	1.34	0.28	0.26	0.69	0.11
Fountain (beginning)	13.4M (129M)	$1024^2 \times 3072$	2.89	0.33	0.30	2.13	0.13
Fountain (last step)	34.1M (391M)	$1024^2 \times 3072$	4.53	0.69	0.95	2.55	0.34
				Memory Usage (GB)			
				Total	Sim	MG	Octree
Drops (first step)	30.5M (131M)	1024^3		25.4	18.0	1.1	6.3
Drops (longest step)	33.3M (129M)	1024^3		24.8	17.2	1.2	6.4
Drops (last step)	12.0M (124M)	1024^3		6.7	4.6	0.5	1.6
Fountain (beginning)	13.4M (129M)	$1024^2 \times 3072$		8.6	5.9	0.5	2.2
Fountain (last step)	34.1M (391M)	$1024^2 \times 3072$		22.4	14.8	1.2	6.4

The timings given in the table refer to our parallel implementation. Compared to the algorithm running on a single core, we achieve a speedup factor between 5 and 6 on our eight core workstation.

Octree vs. Uniform Grid

Coarsening the grid in the liquid interior does inevitably influence the simulation result compared to running the simulation on an uncoarsened grid (corresponding to a uniform grid). To demonstrate that our coarsening strategy does result in very little differences, we have performed a comparison for a dambreak scenario, which is depicted in Fig. 3.11. The same setup was run twice: First, using our octree grid as proposed and second, using the same grid with the difference that no cells are merged in the liquid interior. Hence the only difference between the two simulations is the size of the cells in the liquid interior (in the octree case cells are coarsened up to size $16h$). As can be seen, the major characteristics of the flow are maintained almost exactly. There are only minor differences in surface details and small splashes, which naturally increase over the course of the simulation due to the accumulation of small errors.

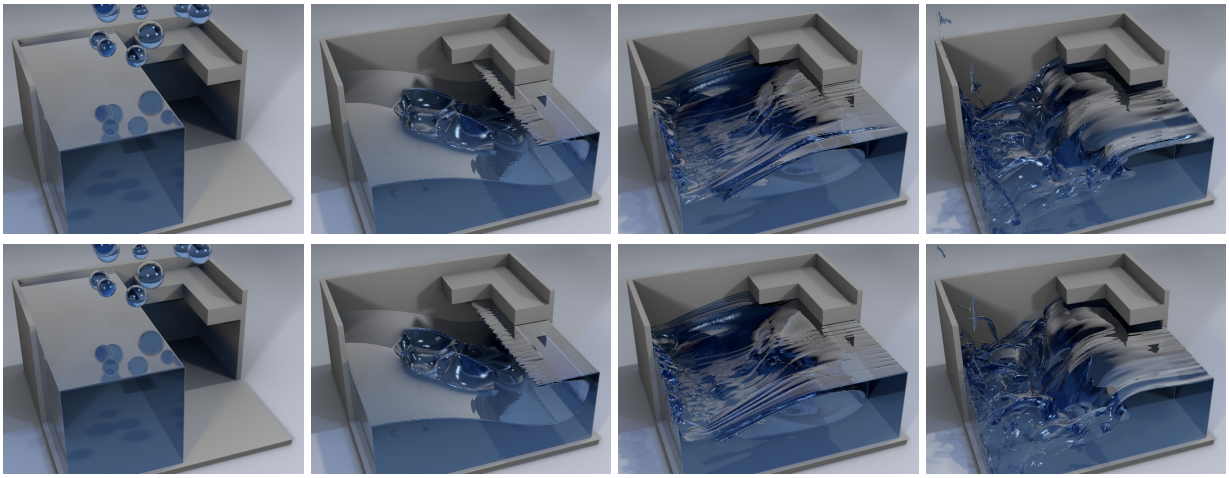


Figure 3.11.: Two dambreak simulations: **(top)** With coarsening in the fluids interior (650k to 1.15M cells). **(bottom)** Without coarsening (3.75M cells) the grid is equivalent to a uniform grid ($\approx 256^3$). Note that the unevenness in the splash coming back off the wall is owed to the random placement of level set particles around the surface.

3.8. Conclusion

To the best of our knowledge, our method presents the first multigrid approach for liquid simulation on a spatially adaptive octree grid using a hexahedral FEM discretization. By using an octree simulation grid in combination with finite elements, the number of simulation elements can be made proportional to the liquid boundary area, and oscillatory velocity modes at level transitions can be avoided. The use of a hexahedral discretization of the simulation domain severely simplifies the construction of the adaptive simulation grid, including a consistent hierarchy of coarse grids for a geometric multigrid solver, and gives rise to regular numerical stencils and efficient realizations of multi-level matrix assembly and multigrid solve. Our experiments have demonstrated excellent convergence rates of the multigrid solver used to enforce incompressibility, enabling efficient simulations of complicated liquid domains consisting of millions of simulation elements.

For the simulation of liquids with free surfaces, a limitation of the current method is the time step constraint (CFL number $c_{CFL} \leq \omega$) that it imposes. This constraint is a result of our particular choice of grid refinement strategy, i.e., the use of a fixed-width refinement band. In a next step it will be interesting to explore more sophisticated refinement strategies in order to loosen the time step constraints and/or to reduce the number of required cells in the grid. One possibility would be to locally vary the width of the refinement band based on the velocities in a close vicinity around the surface.

Part II.

Ensemble Visualization

4

Fundamentals of Variability Plots

In the second part of this thesis, we introduce the general concept of *variability plots* for the visualization of scalar and vector field ensembles. These plots are designed to support in the analysis of spaghetti plots and can act as an additional abstraction layer or even as a full replacement. Before we introduce our specific applications and extensions of variability plots in the following chapters, we first discuss some fundamental, common aspects.

Independent of the underlying type of line data (iso-contours or parameterized curves such as streamlines), the creation of any variability plot follows the same basic scheme. In the first steps, we perform an automated analysis of the input line ensemble given by a spaghetti plot. We first represent all lines in a simple but high-dimensional Euclidean space, and then use *principal component analysis* (PCA) to project them to a low-dimensional feature space, in which we can identify significantly differing trends via clustering. For this, we use *agglomerative hierarchical clustering* (AHC), because it is fast and results in an entire hierarchy of clusters, which allows a user to intuitively change the number of clusters after an initial clustering has been computed. After the trends have been determined, we exploit the invertibility of PCA to transform representatives of each trend from the PCA based feature space back to the original domain space, which ultimately results in an abstract visualization of the major trends in the input line ensemble. On the one hand, we transform the geometric median of each trend back to the domain space, which yields artificial median lines. On the other hand, we fit a *multivariate normal* (MVN) distribution to each trend and transform a corresponding confidence ellipse back to the domain space. This results in confidence regions which intuitively illustrate the spatial standard deviation of each trend.

In the following, in Section 4.1, we first explain how weather forecast ensembles help to assess the uncertainty in numerical weather prediction, which is one of our most important fields of application. Then, in Section 4.2, we provide a detailed summary of a selection of methods which are closely related to variability plots. Last but not least, in Section 4.3, we summarize the mathematical

background of the three central building blocks of variability plots: We discuss a) the basics of PCA and two applications of PCA which are related to our method, b) AHC and the L-method, which can be used to automatically determine an appropriate number of clusters, and c) MVN distributions and confidence ellipses.

4.1. Ensemble Weather Forecasting

We demonstrate the effectiveness of our proposed variability plots on several datasets from numerical ensemble simulations. Even though we partly use artificial datasets and a fluid simulation ensemble created using the simulation method introduced in Part I (see Figs. 1.1+5.10), the majority of our examples is based on ensemble weather forecasts from the European Centre for Medium Range Weather Forecasts (ECMWF).

In recent years, numerical weather prediction has become one of the most important fields of application for ensemble visualization research. This is basically the result of advances in computing power and parallel high-performance computing, which nowadays enable meteorologists to rely heavily on ensemble simulations for assessing the uncertainty in weather forecasts. Ensemble simulations offer a huge advantage over single simulation runs or simple statistical models, since these are simply not sufficient to capture a chaotic system like the earths atmosphere, whose evolution is highly sensitive to initial conditions.

In order to create an ensemble weather forecast, the current state of the atmosphere is first determined approximately and the initial conditions for the numerical simulation are described through a corresponding probability distribution. The evolution of this distribution is then estimated using Monte-Carlo sampling, i.e., several possible initial states are selected and corresponding numerical simulations are run, yielding an ensemble of forecasts in which each run describes a possible future scenario. For instance, the ECMWF operates the Ensemble Prediction System (ENS), which routinely generates ensembles of 51 perturbed forecasts [BBW*06]. The resulting data is not only multivariate (containing fields like, e.g., temperature, pressure and wind speed), but also varies over five dimensions (3×space, 1×time and one dimension for the different ensemble members), which results in a massive amount of information. In operational practice, however, forecasters still rely on simple techniques like plotting summary statistics (such as mean and standard deviation) or spaghetti plots [Wil11]. As a consequence, the variability in an ensemble forecast often cannot be conveyed appropriately and the resulting visualizations can be plagued by heavy visual clutter. This clearly motivates the need for advanced ensemble visualization techniques in the context of numerical weather prediction.

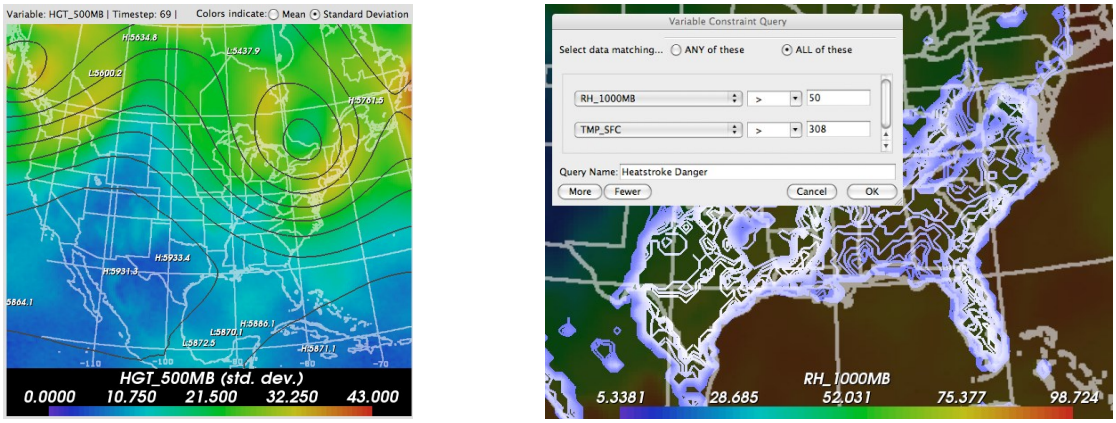


Figure 4.1.: Two visualizations of the Ensemble-Vis tool [PWB*09]. **(left)** Simple visualization of summary statistics in which the iso-contours represent the ensemble mean and the color-coding represents the corresponding standard deviation. **(right)** Advanced probability map visualization showing iso-contours of heatstroke danger, i.e., of the probability that the relative humidity is above 50% and the temperature above 95° Fahrenheit (reprinted from Potter et al. [PWB*09], © 2009 IEEE).

4.2. Alternatives to Spaghetti Plots

Variability plots are built upon the idea of spaghetti plots, i.e., the visualization of a set of line based features which are extract from an ensemble. As a consequence of the popularity of spaghetti plots, a number of other, alternative approaches have been proposed in the visualization community so far. In this section, we summarize the ones most closely related to variability plots, which all aim at reducing visual clutter and adding visual abstraction. We focus on approaches which are designed for ensembles of scalar fields and hence are based on spaghetti plots of iso-contours.

The first approach we want to mention because of its simplicity is the general concept of *probability maps*. Such a map indicates, for every point in the domain, the probability that the scalar value at this point is higher than a given threshold (or alternatively, lower). This probability can be estimated from a scalar field ensemble by simply counting the occurrences of values higher and lower than the threshold at every grid point, and the resulting scalar probability field can be visualized using color mapping or through iso-contours. Strictly speaking, a computation of iso-contours is not required, but the usage of an iso-value-like threshold closely relates this approach to spaghetti plots. Fig. 4.1 (right) depicts an advanced probability map visualization created in the Ensemble-Vis tool developed by Potter et al. [PWB*09]. Additionally, for comparison purposes, a simple summary statistics visualization of an ensemble mean field and corresponding standard deviation is shown in Fig. 4.1 (left).

Other approaches try to summarize the variability in a scalar field ensemble by transforming iso-contours into a continuous representation, or by using glyphs and other representative objects. Pfaffelmoser and Westermann [PW13] use a Gaussian kernel to smooth an ensemble of iso-contours and

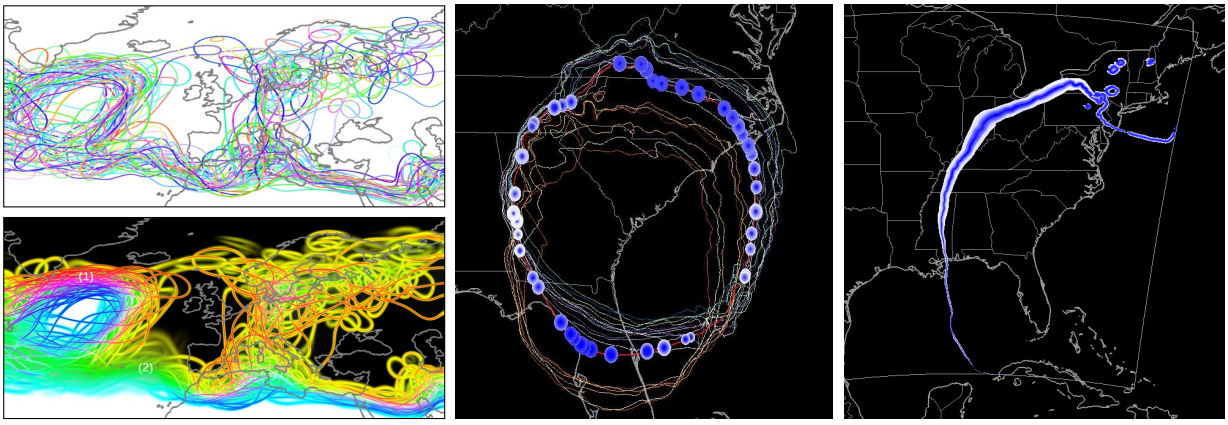


Figure 4.2.: (left) Visualization of an ensemble of iso-contours by Pfaffelmoser and Westermann [PW13]. An input spaghetti plot of iso-contours is transformed into a continuous representation in the form of a spatial probability density function and cumulative distribution function, which are visualized through a special color mapping (reprinted from Pfaffelmoser and Westermann [PW13], © 2013 Eurographics Association). **(middle+right)** Uncertainty glyphs and ribbons introduced by Sanyal et al. [SZD*10]. The glyph/ribbon size corresponds to the overall local uncertainty and the white-to-blue shading illustrates the distribution of deviations from the ensemble mean (reprinted from Sanyal et al. [SZD*10], © 2010 IEEE).

to transform it into a continuous spatial probability density function and a corresponding cumulative distribution function. In combination with a special color mapping, they are able to visualize the spatial spread and positional uncertainty of the contours in an effective way. Furthermore, their approach reveals topological differences between the iso-contours which are not visible in a standard spaghetti plot. An example from their work is shown in Fig. 4.2 (left), where black and white coloring indicates regions in which all ensemble members are “below” or “above” the given iso-value, respectively, and in places with a high occurrence probability of iso-contours the spatial probability density function is mapped to the remaining colors. Sanyal et al. [SZD*10], on the other hand, introduce *graduated uncertainty glyphs* and *ribbons* to encode the statistical distribution of ensemble values at selected locations in the domain. For a given location, the size of a circular glyph corresponds to the overall uncertainty at this point, and the shading of the glyph illustrates the distribution of deviations from the ensemble mean. The authors also apply this shading principle to ribbons along iso-contours of, e.g., the ensemble mean field, which results in an abstract, band-like visualization similar to the *confidence bands* in our contour variability plots (cf. Chapter 6). However, while our confidence bands are based on the common notion of standard deviation, the graduated uncertainty glyphs and ribbons are not based on a common intuition and require detailed knowledge about their computation in order to be interpreted. Furthermore, the size of the glyphs and ribbons corresponds to a scalar uncertainty measure and does not represent a spatial spread, which makes it hard to quantify the resulting visualizations. Two examples for graduated uncertainty glyphs and ribbons are shown in Figs. 4.2 (middle+right).

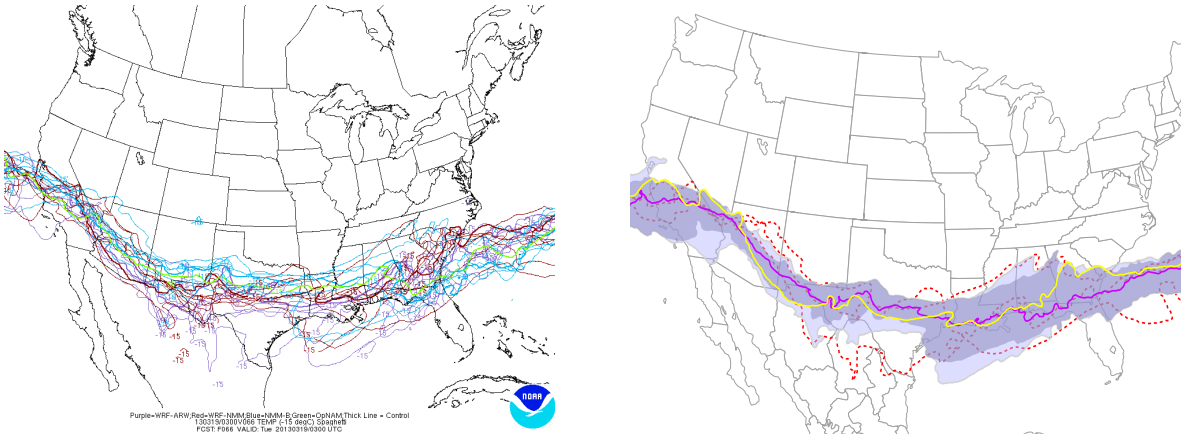


Figure 4.3.: Contour boxplots presented by Whitaker et al. [WMK13]. **(left)** Input ensemble of iso-contours. **(right)** Contour boxplot with median contour (yellow), artificial mean contour (purple), outlier contours (red) and filled regions showing the 50% and 100% data depth bands (reprinted from Whitaker et al. [WMK13], © 2013 IEEE).

The approach most closely related to our variability plots is the concept of *contour boxplots*, which was introduced by Whitaker et al. [WMK13] and later extended to *curve boxplots* by Mirzargar et al. [MWK14]. These plots are based upon the notion of statistical data depth, which, in the basic one-dimensional case, relates the relevance of a scalar measurement to its depth within a set of other measurements. This results in an center-outward ordering with the median being the most important data point. In order to extend this notion to ensembles of iso-contours and curves, for both cases, a global data depth measure is defined. This assigns a scalar depth score to each contour/curve, which indicates how likely it is, in average, that it is lying between other, randomly selected contours/curves. The ensemble members are then sorted according to this depth score, and the member with the highest score is chosen as the most representative, median member. Furthermore, after excluding outliers with small scores, band-shaped regions enclosing the 50% and 100% best members are extracted. In the case of contours, an additional, artificial mean line can be computed as the 50% iso-contour of the corresponding probability map (as discussed at the beginning of this section). The final contour/curve boxplots are then composed by showing median and outlier members as well as the 50% and 100% bands (and optionally, the artificial mean contour). An example contour boxplot is shown in Fig. 4.3.

Compared to our variability plots, contour/curve boxplots are similar in appearance and purpose, but also differ in several major aspects. First of all, contour/curve boxplots are based on the notion of data depth, whereas variability plots are based on the notion of standard deviation. Like in a comparison of standard boxplots and error plots for ensembles of scalar values, this difference does not lead to clear advantages or disadvantages of either method, but, rather, leaves the choice of an appropriate method to the specific application case. An important consequence of the different

notions, however, is that the lines and band boundaries in contour/curve boxplots correspond to actual ensemble members, whereas the median lines and the boundaries of the confidence regions in variability plots are smooth, artificial shapes. Variability plots also differ in that they use a clustering of the input ensemble into significantly differing trends. This may be seen as a major advantage over contour/curve boxplots, since the clustering space we use for variability plots is tightly tied to the mathematical model which is used for generating median lines and confidence regions. Nevertheless, it would also be possible to extend contour/curve boxplots with a clustering step, and to show an overlay of individual boxplots of every cluster.

4.3. Mathematical Background

The computation of variability plots in Chapters 5+6 and the time-hierarchical clustering in Chapter 7 make use of several standard algorithms and mathematical techniques. In this section, we formally discuss the mathematical background of the three central building blocks of our method and show typical applications and examples.

4.3.1. Principal Component Analysis and its Applications

PCA is ubiquitous in almost all scientific fields in which data has to be analyzed. At the beginning of our work on ensemble visualization, we performed several experiments in which we applied PCA to sets of streamlines. The results of these experiments led to several interesting results, which provided the initial motivation for streamline variability plots. In the following, we first provide a formal definition of PCA in general and then show some properties and applications which are related to our approach.

The input to PCA is a set of n data points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^m$ in an m-dimensional Euclidean space. Since PCA is often used for the analysis of measured or observed data, these points are also called *observations* and the different dimensions are also referred to as *variables*. Essentially, PCA determines a new orthonormal coordinate system, which is centered around the mean point $\bar{\mathbf{x}} = \frac{1}{n}\sum_i \mathbf{x}_i$, and whose axes are aligned with directions of small and large variance in the data. Let $\mathbf{u}_1, \ldots, \mathbf{u}_m \in \mathbb{R}^m$ denote the unit-length basis vectors of this coordinate system, which are called *principal components* (PCs), and let $c_{i,j}$ with $i = 1, \ldots, n$ and $j = 1, \ldots, m$ denote the scalar coefficients by which the basis vectors are weighted and which are called *principal component scores*. Then, the original points can be written in the PCA basis as follows:

$$\mathbf{x}_i = \bar{\mathbf{x}} + \sum_{j=1}^{m} c_{ij}\mathbf{u}_j \tag{4.1}$$

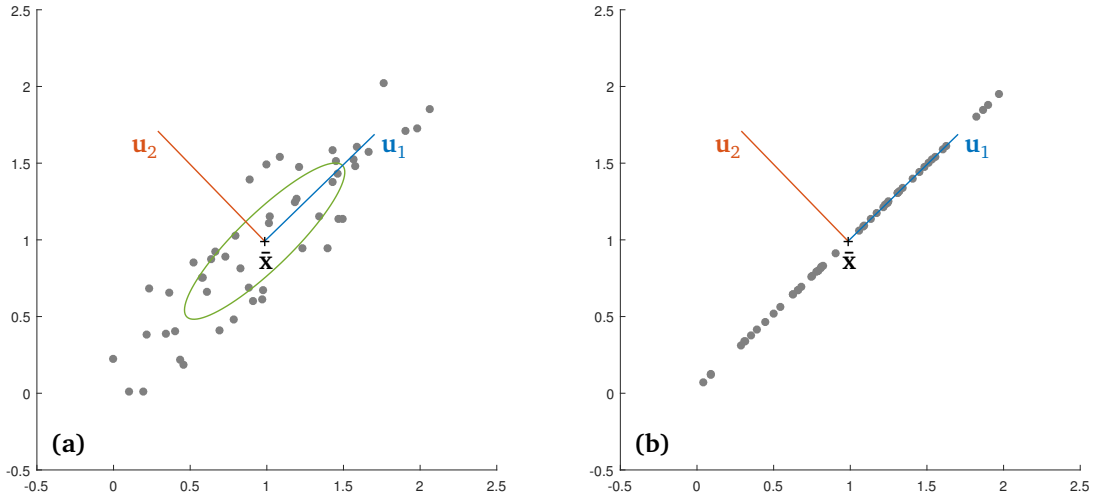


Figure 4.4.: Principal component analysis in 2D. **(a)** For an input set of points (gray), PCA determines a new coordinate system centered at their mean $\bar{\mathbf{x}}$ with orthonormal axes $\mathbf{u}_1$ and $\mathbf{u}_2$, where $\mathbf{u}_1$ is the direction of the largest variance in the data. Intuitively, PCA can be thought of fitting an ellipse (green) to the points. **(b)** Dimension reduction by projecting the points onto the first principal component $\mathbf{u}_1$.

The PCs are sorted in descending order of importance, such that $\mathbf{u}_1$ is aligned with the direction of the largest variance in the data, $\mathbf{u}_2$ is aligned with the direction of the second largest variance in the data (while being orthogonal to $\mathbf{u}_1$), and so on. For our application, it is important to note that the PCA basis does not require a full set of m basis vectors if the number of input points is less or equal than the number of input dimensions. In general, $\min(m, n-1)$ basis vectors are sufficient. An example for a PCA on a two-dimensional point cloud is depicted in Fig. 4.4 (a). Intuitively, the PCA can also be thought of fitting an m-dimensional ellipsoid to the data such that its main axes are aligned with the PCs and that the radius of the main axes is equal to the standard deviation in the respective direction. This is illustrated by the green ellipse.

A PCA can be computed by means of an eigenvector decomposition or through a singular value decomposition. To demonstrate this, we first center the data at the origin, i.e., we define centered points $\mathbf{y}_i := \mathbf{x}_i - \bar{\mathbf{x}}$, and then rewrite Eq. (4.1) in matrix form, which expresses the PCA in the form of a matrix decomposition:

$$Y = UC^T, \tag{4.2}$$

where the centered data matrix $Y \in \mathbb{R}^{m \times n}$ has columns $\mathbf{y}_i$, the PC matrix $U \in \mathbb{R}^{m \times m}$ has columns $\mathbf{u}_j$ and the score matrix $C \in \mathbb{R}^{n \times m}$ has entries $(C)_{i,j} := c_{i,j}$. For the first method of computing the PCA, let $\Sigma \in \mathbb{R}^{m \times m}$ denote the data's covariance matrix, which is defined as $\Sigma := \frac{1}{n-1} YY^T$. Then, the eigenvector decomposition $\Sigma = U\Lambda U^T$ yields the PCs as eigenvectors of Σ, and each eigenvalue

on the diagonal of Λ is equal to the variance in the direction of the corresponding PC. Since U is an orthonormal basis, the score matrix can be obtained by a simple change of bases: $C = Y^T U$. Alternatively, we can perform a singular value decomposition (SVD) of the centered data matrix, i.e., $Y = USV^T$, which yields the PCs as left-singular vectors of Y. This time, the variance corresponding to the i-th PC can be computed from the corresponding singular value as $\frac{1}{n-1}(S)^2_{i,i}$ and the score matrix can be obtained in the same way as before, or, alternatively, through $C^T = SV^T$. In practice, the SVD method typically offers the better performance and is thus preferred. It can be further accelerated by replacing the full SVD by a so-called *thin SVD* (or *economy-size SVD*), which means that only the relevant parts of U, S and V are computed.

A standard application of PCA is dimensionality reduction. Initially, the data points $\mathbf{x}_i$ are transformed into corresponding points $\mathbf{c}_i := \{c_{i,1}, \ldots, c_{i,m}\}$, which have the same dimensionality. In the PCA based representation, however, we can exploit the fact that the coordinates are sorted in descending order of "importance". This allows us to drop trailing coordinates, where the result is always an optimal approximation of the original points in a least squares sense. More precisely, this means that for *any* number $r < m$ of remaining PCs, the approximation

$$\mathbf{x}_i \approx \mathbf{x}_i^r := \bar{\mathbf{x}} + \sum_{j=1}^{r} c_{ij}\mathbf{u}_j \tag{4.3}$$

is optimal in the sense that the error $\sum_{i=1}^{n} \left\|\mathbf{x}_i^r - \mathbf{x}_i\right\|^2$ is minimized. There is a variety of different applications for this kind of dimensionality reduction. E.g., projections to the first two or three PCs are often used to create two- or three-dimensional visualizations of high dimensional data points, and our variability plots use PCA to automatically find low-dimensional feature spaces for potentially complicated line geometry. Fig. 4.4 (b) shows a two-dimensional example for dimensionality reduction in which the set of points from Fig. 4.4 (a) is projected to the first principal component.

Another popular application of PCA is the extraction of dominant modes of variation from time series or ensembles of scalar and vector fields. An understanding of this idea is very helpful for the interpretation of the individual dimensions of the low-dimensional feature spaces that we obtain through PCA in variability plots. The concept of mode extraction has led to a multitude of synonyms for PCA like, e.g., *proper orthogonal decomposition* (POD), which is often used in the context of engineering applications such as computational fluid dynamics, or *empirical orthogonal functions* (EOF), which is often used in the context of meteorological applications. Starting with a set of scalar or vector fields, which all have to be sampled on the same grid, the scalar degrees of freedom at the grid points are interpreted as variables, and the individual fields are interpreted as observations. To extract the dominant modes, a PCA is then be performed in the same way as before. However, instead of investigating the resulting scores $c_{i,j}$, we are now interested primarily in the PCs $\mathbf{u}_j$. These can be

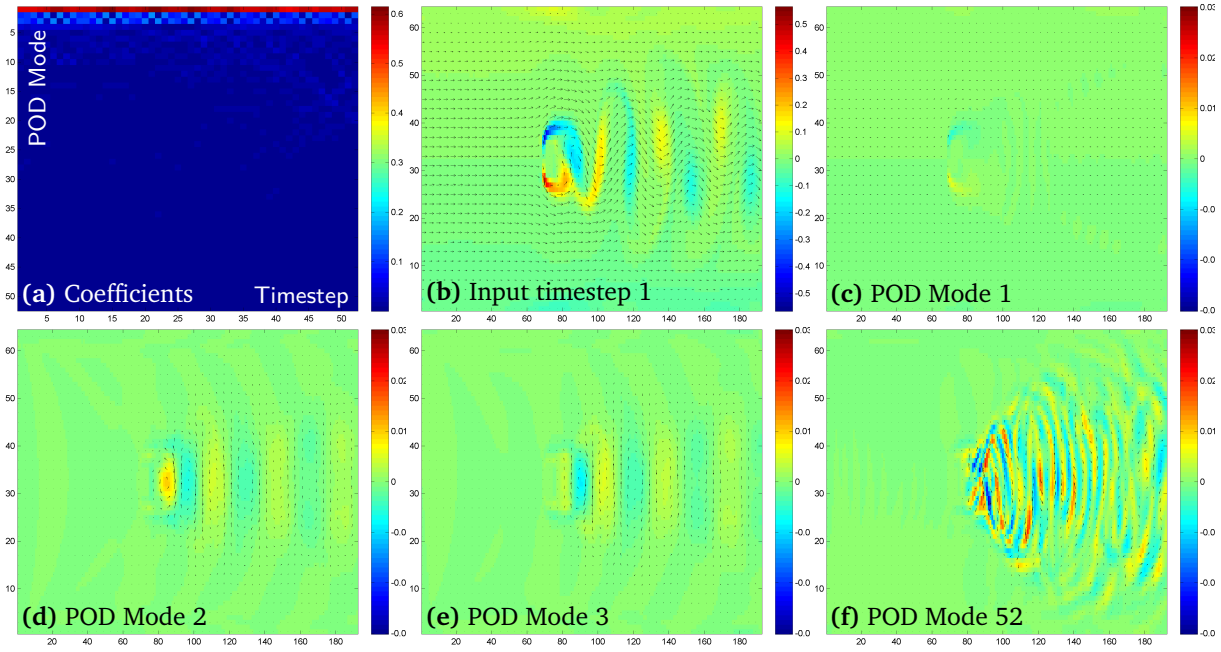


Figure 4.5.: Principal component analysis of an oscillating channel flow around an obstacle (von Kármán vortex street). **(a)** Visualization of score matrix C (absolute values). **(b)** First timestep of input sequence. **(c)** Mode 1, which corresponds to the mean flow from left to right. **(d+e)** Modes 2+3, which are alternating (c.f. rows 2+3 of (a)) and form the periodic vortex shedding pattern behind the obstacle. **(f)** Last mode with least contribution, but highest spatial frequency. The backgrounds of (b-c) are color-coded according to vorticity.

interpreted as scalar or vector fields themselves (analogous to the input fields) and, in this context, are often referred to as *modes*. These modes form a basis from which the initial input fields can be re-composed as linear combinations, and, because of the variance based ordering of PCs, the first few modes are often responsible for most of the structure in the input data.

Fig. 4.5 shows the result of a PCA on a time sequence of vector fields from a numerical simulation of a channel flow around an obstacle. Both the scores $c_{i,j}$ as well as several PCs (or modes) $\mathbf{u}_j$ are visualized. Note that the scaling of the modes can be chosen arbitrarily for visualizations, since the $\mathbf{u}_j$ are basis vectors and only their "direction" is of interest. The most interesting characteristic of the flow is represented by modes 2+3, which alternately contribute to the flow field (as can be seen in rows 2+3 in Fig. 4.5 (a)) and thus form the periodic pattern of shedding vortices behind the obstacle. Compared to this pair of modes, the remaining modes are much less important. However, Fig. 4.5 (f) hints at another important property of PCA when applied to spatially correlated data (i.e., variables corresponding to neighboring grid points are strongly correlated if the input fields are sufficiently smooth): Higher modes contribute less but exhibit higher spatial frequencies. This effect can be even better observed when analyzing turbulent flows, which are typically a superposition of an entire spectrum of turbulent vector fields with different frequencies. For this, consider the proper orthogonal

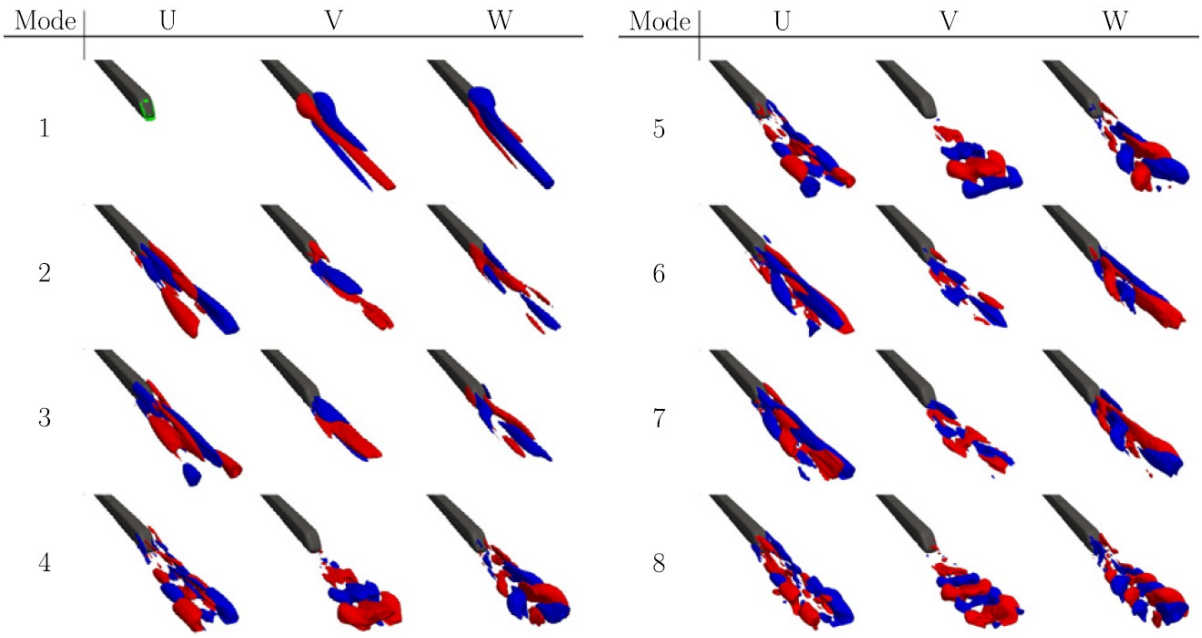


Figure 4.6.: Proper orthogonal decomposition of the three-dimensional, turbulent flow behind a high-speed train [MEH12]. The first eight modes are shown, where mode 1 is the mean flow. Each mode is a vector field which is visualized by means of iso-surfaces of positive (red) and negative speed (blue) in the U, V, and W-components. The train geometry is rendered in gray, and the flow moves from top-left to bottom-right around the rear part of the train. (Reprinted from [MEH12], © 2012, with permission from Elsevier Ltd.)

decomposition of the turbulent flow behind a high-speed train shown in Fig. 4.6, which was presented in the work by Muld et al. [MEH12]. In this case, we can clearly see the increase in spatial frequency over the first few modes. Furthermore, the modes appear in similar pairs (mode 2+3, 4+5, 6+7), which again indicates periodic patterns in the individual frequencies of the turbulent flow.

The effect of increasing spatial frequency in modes can also be observed when PCA is applied to streamlines, which are also a form of spatially correlated data. An example for the "modes" in this context is shown in Fig. 5.3 in Chapter 5.

4.3.2. Agglomerative Hierarchical Clustering

Clustering is an indispensable tool for data analysis because it allows us to find groups of similar objects and, thus, structure in the data. Depending on the application at hand, we can choose from a variety of different standard methods, which includes k-means, hierarchical clustering, expectation maximization (i.e., fitting a Gaussian mixture model) and density based methods like DBSCAN. As indicated before, we use AHC for variability plots, because it is fast and allows a user to quickly and intuitively change the number of clusters after an initial clustering has been performed. In the following, we define several variants of AHC, explain the use of dendrograms, and, in the upcoming

section, we show how we can use the L-method [SC04] to automatically determine an optimal number of clusters. Note that a comparison of AHC to other methods in the context of streamline clustering is presented in Chapter 5.

Let $\{\mathbf{x}_1, ..., \mathbf{x}_n\}$ be a set of n objects for which a clustering should be computed. The basic idea of AHC is to arrange these objects in a hierarchy from which clusterings with different numbers of clusters can be derived. This hierarchy is created in a bottom-up fashion (as opposed to *divisive* hierarchical clustering), and takes the form of an unbalanced binary tree, in which the leafs correspond to the individual objects and the root node represents a cluster which comprises all objects. AHC starts with each object in its own, individual cluster, and successively merges pairs of clusters until only one big cluster remains. To decide which clusters should be merged in each step, AHC makes use of a *distance metric* and a so-called *linkage criterion*. The distance metric can be generally expressed through a distance matrix $D \in \mathbb{R}^{n \times n}$, where $(D)_{i,j}$ is the distance between the objects $\mathbf{x}_i$ and $\mathbf{x}_j$. If the $\mathbf{x}_i$ are points in a Euclidean space, a common choice is to use Euclidean distances, but any other distance metric like, e.g., Hausdorff distances for sets or even a custom metric for more complex objects is possible as well. The linkage-criterion, on the other hand, defines a measure for the distances between entire clusters which is based on the entries of the distance matrix D. During the creation of the cluster hierarchy, we can keep track of these inter-cluster distances by means of a linkage matrix $L \in \mathbb{R}^{k \times k}$, which contains the pair-wise distances for a current number of k remaining clusters. Starting with $L := D$, in each merging step, the smallest distance entry $(L)_{i,j}$ in L is determined, the corresponding clusters i and j are merged, and the corresponding rows and columns in L are collapsed (which reduces the size of L by one row and column). Even though the inter-cluster distances in L are defined as a function of D, in practice, the entries of L can be updated recursively when clusters are merged. This is possible through update rules which only depend on the current state of L.

Since the linkage matrix plays a central role in our extension of AHC to the time-hierarchical clustering presented in Chapter 7, we discuss the aspect of linkage criteria in detail. For this, we consider four popular linkage criteria: single linkage, complete linkage, average linkage and Ward's linkage (also called Ward's method [Jr.63]).

Single/Complete In the following, let $\Delta(A, B) := (L)_{A,B}$ denote the linkage distance between two clusters $A, B \subseteq \{1, \ldots, n\}$. For single and complete linking, it is defined as the minimum and maximum distance, respectively, between individual objects of both clusters:

$$\Delta(A, B) = \min_{i \in A, j \in B} (D)_{i,j} \qquad \text{and} \qquad \Delta(A, B) = \max_{i \in A, j \in B} (D)_{i,j} \tag{4.4}$$

It can easily be seen that, when merging two clusters B and C, the distance of the new cluster $B \cup C$ to an unchanged cluster A can be computed from the current distances using the following update rule:

$$\Delta(A, B \cup C) = \min\big(\Delta(A,B), \Delta(A,C)\big) \quad \text{and} \quad \Delta(A, B \cup C) = \max\big(\Delta(A,B), \Delta(A,C)\big) \tag{4.5}$$

Single linkage is known for its ability to handle clusters of any shape and size, but it is susceptible to outliers and the so-called *chaining effect*, which results in the creation of big clusters which are connected through a chain of objects. Complete linkage, on the other hand, is more robust against outliers and strongly favors spherical clusters, which can cause large, non-spherical groups of objects to be broken down.

Average For average linking, the inter-cluster distance is defined as the average of all distances between individual objects of two clusters:

$$\Delta(A,B) = \frac{1}{|A||B|} \sum_{i \in A} \sum_{j \in B} (D)_{i,j} \tag{4.6}$$

In this case, the corresponding update rule also includes the clusters' cardinalities. It can easily be verified that it is:

$$\Delta(A, B \cup C) = \frac{|B|}{(|B| + |C|)} \Delta(A,B) + \frac{|C|}{(|B| + |C|)} \Delta(A,C) \tag{4.7}$$

Average linking also favors spherical clusters, but in a weaker sense than complete linkage. It thus offers a good compromise between single and complete linkage, and we use it for streamline variability plots (see Chapter 5) as well as contour variability plots (see Chapter 6).

Ward The derivation of Ward's linking is slightly more involved. Here, the underlying idea is to minimize the intra-cluster variances, which are only defined for Euclidean distances. Thus, the input points are required to be points $\mathbf{x}_i \in \mathbb{R}^d$ in a Euclidean space, and, for any resulting clustering $\{C_1, \ldots, C_k\}$, our goal is to minimize the following sum of squared errors:

$$SSE\big(\{C_1, \ldots, C_k\}\big) = \sum_{j=1}^{k} \sum_{i \in C_j} ||\mathbf{x}_i - \mathbf{m}_j||^2, \tag{4.8}$$

where $\mathbf{m}_j := \frac{1}{|C_j|} \sum_{i \in C_j} \mathbf{x}_i$ denotes the mean of cluster j. The minimization of Eq. (4.8) is performed in a greedy manner by choosing to merge, in each step, the pair of clusters which causes the smallest increase of the SSE. Consequently, the linkage distance between two clusters is defined as the SSE

increase which is caused by merging them:

$$\Delta(A,B) = SSE(\{A\cup B,\dots\}) - SSE(\{A,B,\dots\}) \tag{4.9}$$

$$= \sum_{i\in A\cup B} ||\mathbf{x}_i - \mathbf{m}_{A\cup B}||^2 - \sum_{i\in A} ||\mathbf{x}_i - \mathbf{m}_A||^2 - \sum_{i\in B} ||\mathbf{x}_i - \mathbf{m}_B||^2 \tag{4.10}$$

This expression can be transformed[1] into two equivalent representations, i.e., the linkage distance $\Delta(A,B)$ can also be expressed in terms of a) the cluster means of A and B and b) the initial distances $(D)_{i,j}$:

$$\Delta(A,B) = \frac{|A||B|}{|A|+|B|} ||\mathbf{m}_A - \mathbf{m}_B||^2 \tag{4.11}$$

$$= \frac{1}{|A|+|B|}\left(\sum_{i\in A}\sum_{j\in B} (D)_{i,j}^2 - \frac{|B|}{2|A|}\sum_{i\in A}\sum_{j\in A}(D)_{i,j}^2 - \frac{|A|}{2|B|}\sum_{i\in B}\sum_{j\in B}(D)_{i,j}^2 \right) \tag{4.12}$$

Furthermore, we can use Eq. (4.11) to derive an update rule for Ward's linking, which ultimately yields the following formula (a so-called *Lance-Williams formula*):

$$\Delta(A\cup B, C) = \frac{|A|+|C|}{|A|+|B|+|C|}\cdot\Delta(A,C) + \frac{|B|+|C|}{|A|+|B|+|C|}\cdot\Delta(B,C) - \frac{|C|}{|A|+|B|+|C|}\cdot\Delta(A,B)$$

Note that, in contrast to the other linkage criteria, the initial linkage matrix of Ward's linking is not equal to the distance matrix (i.e., $L \neq D$), which has to be accounted for in practical implementations. This can be seen by rewriting the linkage distance Eq. (4.11) with single member clusters:

$$(L)_{i,j} = \Delta\big(\{i\},\{j\}\big) = \frac{1}{2}(D)_{i,j}^2$$

Ward's linking behaves similar to average linking and favors spherical clusters. However, it tends to produce more equally sized clusters, especially if the input points are distributed evenly. For this reason, we replace the average linking from Chapter 6 with Ward's linking in Chapter 7.

In Fig. 4.7, a two-dimensional example for AHC is shown, where Fig. 4.7 (a) depicts the set of input points $\mathbf{x}_i$ with corresponding index i. For these points a cluster hierarchy was created using average linking and Euclidean distances as distance metric. The resulting cluster hierarchy is illustrated in Fig. 4.7 (b), where each merge is represented by a shaded region. These regions are nested due to the hierarchal structure of the clustering, and numbered according to the order in which the merges occurred. As shown in Fig. 4.7 (c), hierarchical clusterings are often plotted in the form of so-called

[1] Essentially, we have to use $\mathbf{m}_{A\cup B} = (|A|\mathbf{m}_A + |B|\mathbf{m}_B)/(|A|+|B|)$, and we have to make heavy use of scalar products, similar to $||\mathbf{a}-\mathbf{b}||^2 = \langle a-b, a-b\rangle = \langle a,a\rangle + 2\langle a,b\rangle + \langle b,b\rangle$.

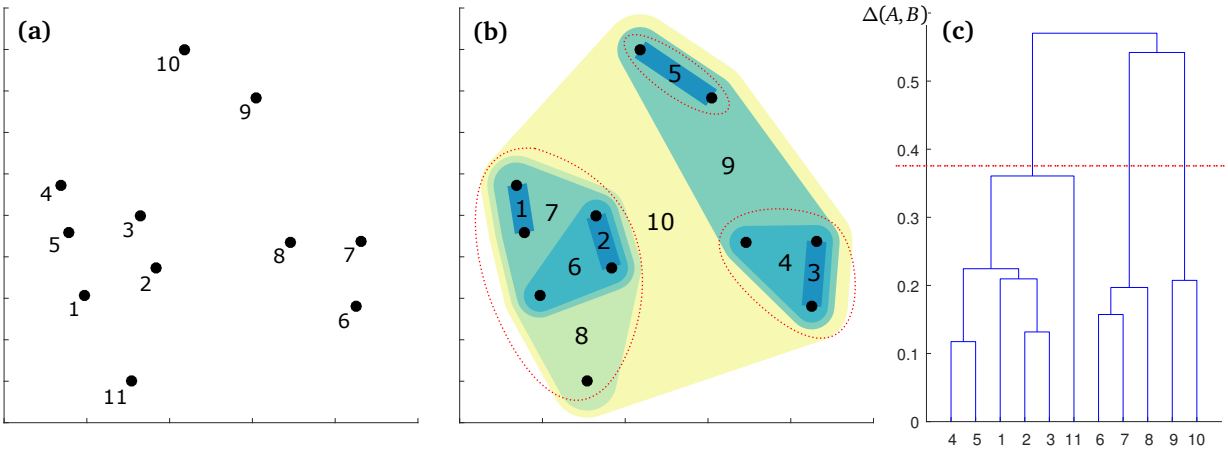


Figure 4.7.: Agglomerative hierarchical clustering in 2D using Euclidean distances and average linking. **(a)** Input points (with indices). **(b)** Graphical illustration of the cluster hierarchy with numbers corresponding to the order of merges. **(c)** Dendrogram. A clustering with, e.g., three clusters can be obtained by "cutting" the hierarchy at the corresponding level (dotted, red lines in (b+c)).

dendrograms, which basically show a rendering of the binary clustering tree. Additionally, the y-axis encodes information about the similarity of merged clusters. Each merge is associated with a horizontal bar, which connects two clusters A and B and is drawn at a height which corresponds to the cost of merging them. In AHC, we can choose this cost as the linkage distance, i.e., $y = \Delta(A, B)$. To create a final clustering based on the cluster hierarchy and a given number of clusters, intuitively, we can cut off the clustering tree at the appropriate height, which is illustrated for the case of three clusters by the dotted, red lines in Figs. 4.7 (b+c).

4.3.3. L-Method

After a hierarchy of clusters has been generated, the remaining task is to determine an appropriate or even optimal number of clusters. To do this in variability plots, we use the L-method which was developed by Salvador and Chan [SC04]. This method is based on the analysis of an evaluation graph, in which different numbers of clusters are compared to the quality of the corresponding clustering, measured by, e.g., an error function or a distance metric. An example is shown in Fig. 4.8 (a). Such evaluation graphs typically consist of two distinctive parts: A steep region to the left and a flat region to the right, which are connected by a transition region that forms a knee in the graph. The left region has the characteristic property that increases in the number of clusters result in significant quality improvements. At the knee, however, further increases start to have only little effect and, thus, the knee's location is a suitable choice for the number of clusters. In order to find the knee, the L-method searches for the best fit of a pair of lines to the graph.

Let $e : \{k_{min}, ..., k_{max}\} \rightarrow \mathbb{R}$ denote the evaluation function shown in the graphs, which maps a

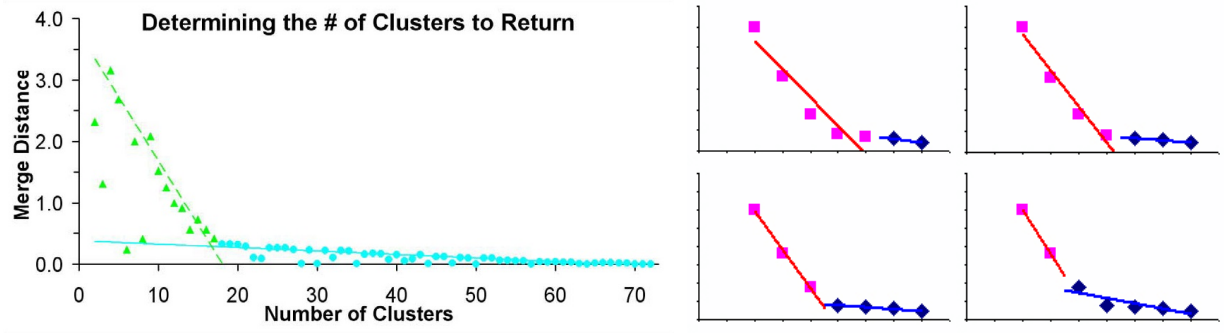


Figure 4.8.: L-method for determining the number of clusters in hierarchical clustering methods [SC04]. **(a)** Evaluation graph, which plots the number of clusters against a cluster quality measure. The position of the knee in the graph gives the optimal number of clusters. **(b)** The knee is extracted by finding the best-fit pair of lines to the graph. (reprinted from Salvador and Chan [SC04], © 2004 IEEE).

range of candidate numbers of clusters to a corresponding quality measure. In the case of AHC, we can use the linkage distances, i.e., $e(k) := \Delta(A,B)$, where A and B are the clusters which were merged in order to get from $k+1$ to k clusters. Consequently, small values of $e(k)$ indicate that k clusters are a good approximation, and large values of $e(k)$ indicate the opposite. In order to find the best-fit pair of lines, the evaluation graph is split at different positions c, and two lines are fitted to the points with $k = k_{min}, \ldots, c$ and $k = c+1, \ldots, k_{max}$. This is illustrated in Fig. 4.8 (b) The optimal split position is then the position c_{opt} which results in the smallest overall approximation error RMSE_c:

$$c_{opt} = \underset{c \in \{k_{min}+1, \ldots, k_{max}-2\}}{\arg\min} \mathrm{RMSE}_c \tag{4.13}$$

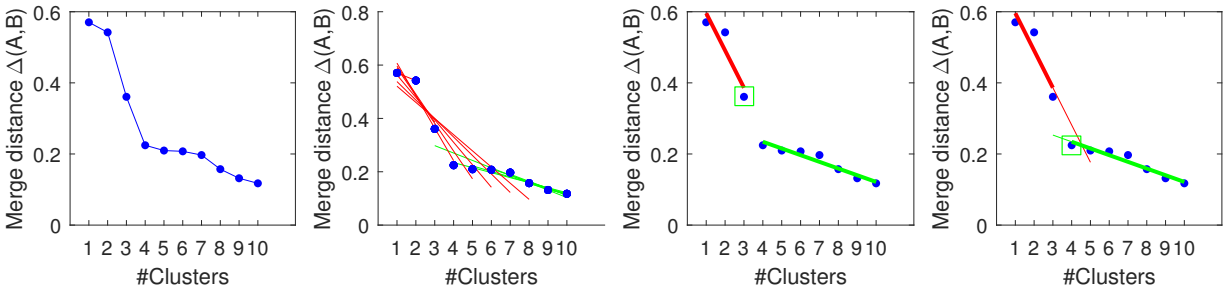
RMSE_c is the error corresponding to a split at c, and it is defined as the average of the root-mean-square errors of the lines fitted to the points left and right of c, weighted with the length of these lines:

$$\mathrm{RMSE}_c = \frac{c - k_{min} + 1}{k_{max} - k_{min} + 1} \cdot \mathrm{RMSE}(k_{min}, c) + \frac{k_{max} - c}{k_{max} - k_{min} + 1} \cdot \mathrm{RMSE}(c+1, k_{max}) \tag{4.14}$$

The individual error $\mathrm{RMSE}(k_l, k_r)$ is defined as the root-mean-square error of the best-fit line through the points $\big(k_l,\, e(k_l)\big)^T, \big(k_l+1,\, e(k_l+1)\big)^T, \ldots, \big(k_r,\, e(k_r)\big)^T$. Based on this, in the standard L-method, the optimal number k_{opt} of clusters is chosen as the optimal split position:

$$k_{opt} := c_{opt} \tag{4.15}$$

The L-method is claimed to be parameter free, but, effectively, its outcome does depend on the choice



(a) Evaluation graph **(b)** Possible fit lines **(c)** Best fit (standard) **(d)** Best fit (extension)

Figure 4.9.: L-method applied to the example from Fig. 4.7, using $k_{min} = 1$ and $k_{max} = 10$. **(a)** Evaluation graph. **(b)** All possible pairs of best-fit lines. **(c)** Best fit according to the standard L-method, which always chooses the last point of the left line. A number of three clusters is chosen (green box) **(d)** Best fit according to our extension of the L-method, which chooses the first point of the right line if this point is closer to the intersection of both lines. A number of four clusters is chosen.

of the clustering quality measure and of the minimum and maximum numbers of clusters k_{min} and k_{max}. Instead of simply including all possible numbers of clusters, which is the default option, we can also limit the range of possible choices. For instance, we typically choose $k_{max} = 20$ for variability plots, because we are only interested in coarse clusterings with a small number of clusters.

Specifically for such applications in which the desired number of clusters is very small, we have developed an extension of the L-method. It addresses the problem that $c_{opt} + 1$ is sometimes a better choice for k_{opt} than c. This can be neglected if c is large, but it can make a huge difference if the choice is between, e.g., 3 and 4 clusters. Fig. 4.9 demonstrates this problem for the clustering hierarchy from Fig. 4.7, which has $n = 11$ points. The pair of best-fit lines according to Eq. (4.13) is shown in Fig. 4.9(c), where $c_{opt} = 3$. Thus, the standard L-method chooses $k_{opt} = 3$ in this case, even though the knee of the graph is clearly at $c_{opt}+1 = 4$. As a remedy for this subtle problem, we always consider c_{opt} and $c_{opt}+1$ when choosing k_{opt}. We compute the x-coordinate $\kappa \in \mathbb{R}$ of the intersection between the best-fit pair of lines, and then choose k_{opt} depending on whether c_{opt} or $c_{opt} + 1$ is closer to κ:

$$k_{opt} := \min(c + 1, \max(c, \text{round}(\kappa))) \tag{4.16}$$

The effect of this extension to the standard L-method can be seen in Fig. 4.9(d), where now a number of four clusters is chosen instead of three.

4.3.4. Multivariate Normal Distribution

For variability plots we heavily rely on MVN distributions, which we use to extend the notion of standard deviation from a multi-dimensional, PCA based feature space to the domain space of the original objects which are described by this feature space. In the following, we summarize the mathemati-

cal background of MVN distributions, and, in the upcoming chapters, we then combine multiple MVN distributions to represent more complex, multi-modal distributions (similar to Gaussian mixture models). We also show how to sample confidence ellipses with random points, which is required for the computation of streamline variability plots in Chapter 5.

We start with the well-known one-dimensional normal distribution (or Gaussian distribution) in order to show how it generalizes to multiple dimensions. For a given mean value μ and variance σ^2, we can measure the deviation of an observation x from the mean value in units of standard deviation, which is called *Mahalanobis distance*:

$$d(x, \mu, \sigma^2) = \left| \frac{x - \mu}{\sigma} \right| \tag{4.17}$$

Based on this distance, the probability density function (PDF) of the one-dimensional normal distribution is defined as:

$$\mathcal{N}(x, \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2} d\left(x, \mu, \sigma^2\right)^2 \right) \tag{4.18}$$

For the generalization to d dimensions, the variance σ^2 is replaced by its multi-dimensional equivalent—the covariance matrix Σ. The Mahalanobis distance then becomes:

$$d(\mathbf{x}, \mu, \Sigma) = \sqrt{(\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu)} \tag{4.19}$$

And the corresponding PDF changes to the general MVN distribution:

$$\mathcal{N}(\mathbf{x}, \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^d \det(\Sigma)}} \exp\left(-\frac{1}{2} d(\mathbf{x}, \mu, \Sigma)^2 \right) \tag{4.20}$$

An example for a two-dimensional MVN PDF is depicted in Fig. 4.10(a). The MVN distribution has an ellipsoidal shape and, thus, the underlying math is closely related to PCA. For instance, the main axes of corresponding ellipsoids can be extracted from the covariance matrix Σ through an eigenvector decomposition (cf. Section 4.3.2). The Mahalanobis distance Eq. (4.19) now describes the multi-dimensional distance of an observation $\mathbf{x}$ to the mean point μ in "ellipsoidal" units of standard deviation. Its iso-contours are called confidence ellipses, and can be used to visualize MVN distributions in an intuitive way. Formally, we can define a confidence ellipse $\mathcal{E}(\Sigma, \mu, \alpha)$ as the set of points whose Mahalanobis distance is equal to a given iso-value α:

$$\mathcal{E}(\Sigma, \mu, \alpha) = \{\mathbf{x} \in \mathbb{R}^d \mid d(\mathbf{x}, \mu, \Sigma) = \alpha\} \tag{4.21}$$

A visualization of an MVN distribution through several confidence ellipses is depicted in Fig. 4.10 (b).

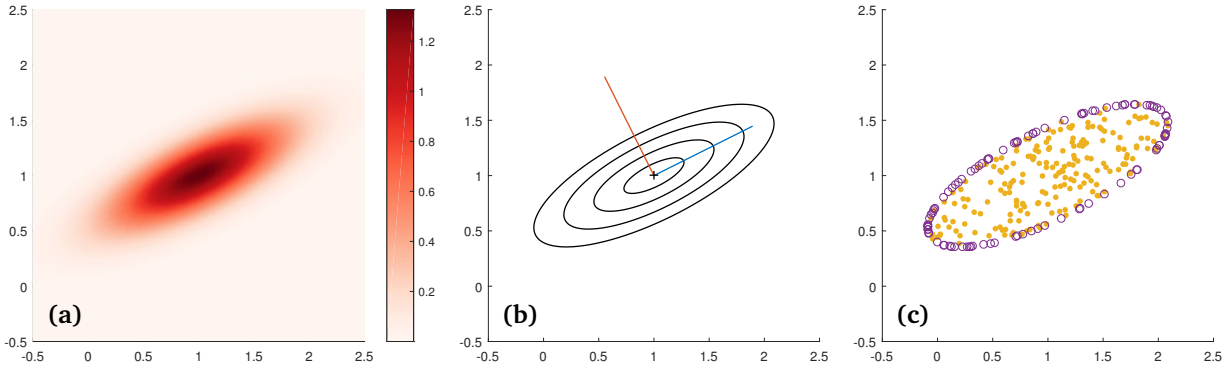


Figure 4.10.: Multivariate normal distribution in 2D. **(a)** Probability density function. **(b)** Confidence ellipses to $\alpha = 0.5, 1.0, 1.5, 2.0$ standard deviations, and their main axes (blue/red). **(c)** Random sampling of the confidence ellipse to $\alpha = 2.0$ with uniformly distributed points (yellow) and random sampling of its boundary (purple).

For the streamline variability plots presented in Chapter 5, we need to be able to uniformly sample the interior of a confidence ellipse with random points. This is related to drawing random points from arbitrary MVN distributions $\mathcal{N}(\mathbf{x}, \mu, \Sigma)$, which can be achieved in two basic steps: First, we draw a random point $\mathbf{s} \in \mathbb{R}^d$ from the standard MVN distribution, such that $\mathbf{s} \sim \mathcal{N}(\mathbf{x}, 0, ID)$ (where ID is the identity matrix). Note that this can be implemented by drawing and concatenating d random numbers from a standard normal distribution $\mathcal{N}(x, 0, 1)$. Then, we use the eigenvector decomposition $\Sigma = U\Lambda U^T$ of the covariance matrix, and compute the desired random number $\mathbf{r} \sim \mathcal{N}(\mathbf{x}, \mu, \Sigma)$ as:

$$\mathbf{r} = \mu + U\sqrt{\Lambda}\,\mathbf{s} \tag{4.22}$$

Intuitively, the spherical distribution of $\mathbf{s}$ is first scaled by the standard deviations contained in the diagonal matrix $\sqrt{\Lambda}$, and then U performs a rotation into the coordinate system of the ellipsoid corresponding to $\mathcal{N}$.

Now, in order to draw random points uniformly from the interior of a confidence ellipse $\mathcal{E}(\Sigma, \mu, \alpha)$, we can use a similar two-step procedure. In the first step, we draw a uniformly distributed point $\mathbf{s}' \in \mathbb{R}^d$ inside of the unit-hypersphere (i.e., $||\mathbf{s}'|| \leq 1$), which can be achieved by normalizing $\mathbf{s} \sim \mathcal{N}(\mathbf{x}, 0, ID)$ from above and rescaling it with an appropriate radius:

$$\mathbf{s}' = u^{1/d} \frac{\mathbf{s}}{||\mathbf{s}||}, \tag{4.23}$$

where u is uniformly distributed in $[0; 1]$. Since the standard MVN distribution is radially symmetric, the normalization $\mathbf{s}/||\mathbf{s}||$ results in a uniform distribution of points on the surface of the unit-hypersphere. Then, for the adjustment of the corresponding radii, consider the cumulative distribution function (CDF) of the desired distribution of $||\mathbf{s}'||$, which is the probability that $||\mathbf{s}'||$ is

smaller than a radius R. For a uniform distribution of $\mathbf{s}'$ in the unit-hypersphere, this probability has to be proportional to the volume of a sphere with radius R in d dimensions, which means that $P(||\mathbf{s}'|| \leq R) = const \cdot R^d$. Thus, rescaling with the radius $u^{1/d}$ according to the inverse of this CDF gives the desired uniform distribution of $\mathbf{s}'$ in the unit-hypersphere.

In the second step, we apply the same transformation as in Eq. (4.22), but we add a scaling with the desired size α of the confidence ellipse in units of stand deviation:

$$\mathbf{r}' = \mu + \alpha U \sqrt{\Lambda}\, \mathbf{s}' \tag{4.24}$$

The resulting distribution of $\mathbf{r}'$ fills $\mathcal{E}(\Sigma, \mu, \alpha)$, and it is also uniform because a linear transformation of a uniform distribution results in another uniform distribution. This uniformity, however, requires Σ to have full rank, which is not necessarily the case since (e.g., we could be sampling a two-dimensional ellipse on a plane in three dimensions). If Σ has rank $c < d$, we need to explicitly create c-dimensional random points according to the dimensionality of the resulting ellipse. More precisely, let $\mathbf{s}'_c \in \mathbb{R}^c$ denote the result of drawing according to Eq. (4.23) in c dimensions. Then we can modify Eq. (4.24) to

$$\mathbf{r}' = \mu + \alpha U_c \sqrt{\Lambda_c}\, \mathbf{s}'_c, \tag{4.25}$$

where Λ_c is the diagonal submatrix of Λ which contains only the non-zero eigenvalues, and U_c only contains the eigenvectors (=columns) corresponding to these non-zero eigenvalues.

Let us note that it is possible to modify the whole procedure such that only the boundary of the confidence ellipse is sampled, which might be used as an alternative to the default uniform sampling of its interior in Chapter 5. To achieve this, we can simply omit the scaling factor $u^{1/d}$ in Eq. (4.23). However, the resulting distribution of points will not be uniform with respect to $\mathbb{R}^d$, which can be problematic if d is large and/or the eigenvalues of Λ differ greatly in scale. An example for the uniform sampling of a confidence ellipse's interior and a sampling of only its boundary is depicted in Fig. 4.10 (c).

5

Streamline Variability Plots

In this chapter, we present the first application of variability plots to ensembles of streamlines, which are extracted from an ensemble of vector fields by seeding one streamline in each ensemble member from a single user-selected location. While we focus specifically on the case of streamlines, the resulting method is general in the sense that it is applicable to any set of parameterized curves, even if they do not pass through a common location. The parameterization enables us to interpret the vertex positions of each curve as high-dimensional feature vectors, which serve as the starting point for the subsequent PCA based analysis of the ensemble distribution. In the following, we pay particular attention to the characteristics of the PCA based feature space, and we analyze the clustering in this feature space by comparing it to alternative clustering approaches for line geometry. Finally, we demonstrate the potential of *streamline variability plots* in a number of real-world examples and compare our results to curve boxplots [MWK14].

5.1. Introduction

To analyze the uncertainty that is represented by an ensemble, the variability of the ensemble members need to be characterized, and the major trends and outliers in the shape and spatial location of relevant features need to be determined. A well-known visual analysis technique for ensembles of features are so-called spaghetti plots, overlaid plots—typically in 2D—of features like particle trajectories or iso-contours in individual members of an ensemble field. Especially in the atmospheric domain, spaghetti plots are one of the most popular means for variability analysis. Spaghetti plots, on the other hand,

This chapter is based on material that has been originally published in F. FERSTL, K. BÜRGER, R. WESTERMANN: Streamline Variability Plots for Characterizing the Uncertainty in Vector Field Ensembles. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization)*, 22(1):767-776, 2016.

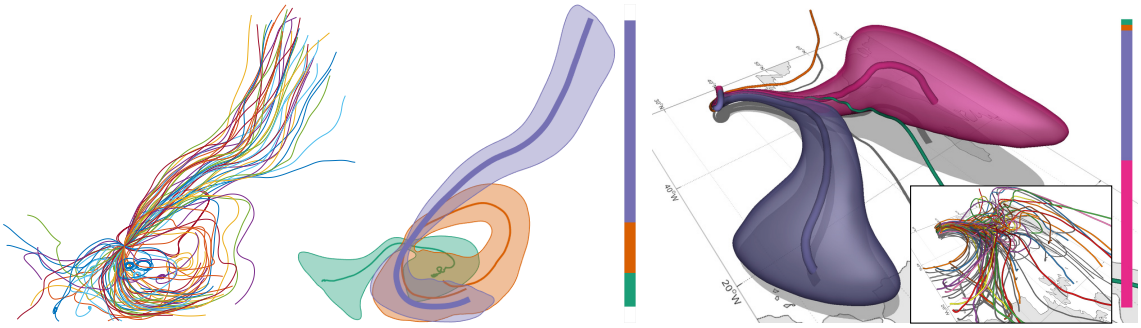


Figure 5.1.: From a set of streamlines in an ensemble of vector fields **(left)**, our method generates an abstract visualization of the major trends in this set **(middle)**. For each trend, a region of high confidence and a representative streamline-median is extracted. The relative strength of a trend is indicated by the thickness of its median line and by the bar plot on the right. Our method works in 2D and 3D **(right)**, as well as for particle trajectories in time-dependent fields.

can produce visual clutter when many features overlap, and they cannot easily convey major trends, outliers, and statistical properties of the feature distribution. To overcome some of these limitations, Whitaker et al. [WMK13] and Mirzargar et al. [MWK14] have recently introduced contour and curve boxplots, respectively, for the visualization of curve-like features based on the concept of statistical data depth.

Our contribution: We introduce a new method to statistically model the variability of specific features among the ensemble members, so that the probability of a particular feature situation can be estimated from the ensemble. We concentrate on the visual analysis of streamlines in 2D and 3D flow fields, and visualize the statistical properties of streamlines passing through a selected location. We first transform the feature representation into a low-dimensional Euclidean space, in which a distance metric as well as an ordering of the features along each dimension is given. We employ this to derive a statistical model of the distributions of clusters of similar streamlines in the Euclidean space, and we propose a method to transform the resulting distributions into confidence regions—so called *lobes*—of the streamlines in the spatial domain. We call the resulting visualizations *streamline variability plots* (see Fig. 5.1). Our particular contributions are

- the use of principal component analysis (PCA) to convert streamlines into a structure preserving Euclidean space (PCA-space), and the clustering in this space to detect trends and outliers in the set of streamlines,
- a new concept of streamline-median, which is based on the existing concept of the (multidimensional) geometric median,
- a non-parametric probabilistic model of the clustered streamline distributions in PCA-space, and a new approach to transform the probabilistic model in this space into a streamline distribution in the spatial domain,

- a visualization method for confidence lobes in 2D and 3D to indicate an estimated range of locations which includes all streamlines within a prescribed standard deviation.

In particular we demonstrate, that this new approach adheres to the requirements on uncertainty visualization techniques proposed by Whitaker et al. [WMK13], as it conveys statistical properties of the shapes of streamlines, provides a qualitative abstract and quantitative statistical interpretations of streamlines, and reveals major trends and outliers in the initial data. We show that all involved operations can be computed fast enough to allow for an interactive exploration of even 3D vector-valued ensembles to identify the sources and evolution of uncertainty in streamlines.

5.2. Related Work

Streamline variability plots use streamline clustering and probabilistic streamline estimation for ensemble visualization, taking into account the similarity and frequency of occurrence of streamlines over multidimensional intervals. The technique has overlap with techniques in uncertainty and ensemble visualization, and curve clustering:

Uncertainty and Ensemble Visualization For the most recent survey on the topic let us refer to the book by Bonneau et al. [BHJ*14]. To visualize the effect of uncertainty on the position and structure of relevant features such as iso-contours, previous works have used confidence envelopes [PWL97, ZWK10], surface displacements [GR04], and made use of the concept of animation [Bro04, LLPY07]. The concept of numerical condition was introduced to extract level-set features in uncertain scalar fields [PH11], and it was further extended to account for correlations in the data [PWH11, PRW11], as well as to also consider non-parametric models for uncertainty [PH13].

The concepts of stream lines and critical points has been generalized to uncertain (Gaussian) vector field topology, in order to segment the topology by integrating particle density functions [OGHT10]. Probabilistic local features, such as critical points, have been extracted from Gaussian distributed vector fields using Monte Carlo sampling [PPH12]. In a fuzzy topology, the topological decomposition is performed by growing streamwaves, based on a representation for vector fields called edge maps [BJB*11].

Obermaier and Joy [OJ14] have classified ensemble visualization techniques into feature based and location based approaches. The latter analyze and compare data properties at fixed locations in the domain using descriptive statistics. Feature based approaches analyze domain-specific features which are first extracted from the individual ensemble members. The visualization of feature variability in ensemble fields is often performed via spaghetti plots of selected contour lines or threshold probabilities of 2D fields such as surface wind speed [PWB*09, Wil11]. Glyphs and confidence ribbons were

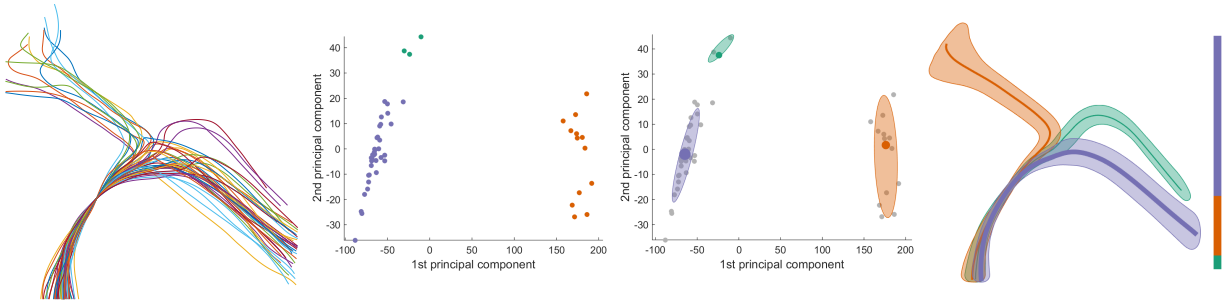
introduced to emphasize the Euclidean spread of contour ensembles [SZD*10]. Recently, Whitaker et al. [WMK13] and Mirzargar et al. [MWK14] built upon the ordering of multivariate data using the concept of statistical band depth to enable an improved visualization of the uncertainty in spaghetti plots of ensembles of curves. Locations in 3D flow fields were clustered based on the divergence of transport patterns to analyze trends in flow ensembles [HOGJ13].

Curve Clustering Our work is related to clustering approaches for curves in 2D and 3D space, which group a set of curves into similar subsets based on a given similarity measure. For a general overview of clustering techniques let us refer to the overview article by Jain [Jai10]. For the clustering of curves, most often geometric similarity measures have been employed, for instance, based on Euclidean distances [CCK07, RT12], curvature and torsion signatures [YWSC12, MJL*13], predicates for stream- and pathlines based on flow properties along these lines [BPMS12], or user-selected streamline predicates [SS06]. For a good overview of similarity measures using geometric distances between curves let us refer to the comparative study by Zhang et al. [ZHT06].

Integral curves in flow fields have been clustered using a two-stage approach, by first performing a geometry based coarse grouping of streamlines, and then clustering in a low-dimensional Euclidean space comprising streamline properties based on shape and velocity [CYY*11]. The pairwise Hausdorff metric between streamlines has been employed to project streamlines into an Euclidean space and perform spectral graph clustering in this space [RT12]. For curves, Agglomerative Hierarchical Clustering (AHC) with different cluster proximity measures has shown to be effective [MJL*13, YWSC12]. Different clustering approaches and similarity measures for fiber tracts in Diffusion Tensor Imaging (DTI) data have been evaluated [MVvW05], among them shared nearest neighbor clustering and AHC. A geometry based similarity metric considering partial intervals for fiber tracts in DTI data has been used in AHC to cluster such tracts [ZCL08]. A reduction technique called Laplacian eigenmaps has been applied to transform fiber tracks to a low dimensional Euclidean space [BPKW03]. Recently, an evaluation of different clustering approaches for streamlines using geometry based similarity measures has been performed [OLK*14].

In some of the previous works, curves have been reduced to low-dimensional representations, for instance by using single measures of geometric similarity. This can result in a significant amount of information that is lost, and usually the initial data can no longer be reconstructed from the reduced representation. It is worth noting that our approach overcomes both of these limitations.

Related to the clustering of integral lines in flow fields are approaches performing clustering of vector fields based on local coherent regions, e.g., by merging locations which are similar in position and orientation of the vectors [TvW99], by splitting regions where the differences between streamlines in these regions and streamlines in an approximated flow field are large [HWHJ99], by using anisotropic diffusion to automatically cluster regions of strong correlation in the flow data [GPR*00],



(a) Input **(b)** PCA + Clustering **(c)** Confidence Ellipses & Geom. Medians **(d)** Variability Plot

Figure 5.2.: Method overview: **(a)** The initial set of streamlines. **(b)** PCA transforms lines into an Euclidean space—PCA-space—in which clustering can be performed. **(c)** Multivariate normal distributions—represented by confidence ellipses and geometric (cluster) medians—are fitted to the points in PCA- space. **(d)** Medians and ellipses are transformed back to the domain space and yield the variability plot of the streamline ensemble.

and by clustering trajectories into sets of vector fields [FKSS13]. For an overview of approaches for vector field clustering let us also refer to the survey by Salzbrunn et al. [SJWS08].

Mostly related to our approach is the method initially proposed by Bashir et al. [BKS03], and later evaluated in the report by Zhang et al. [ZHT06], where PCA was used in combination with Euclidean k-means clustering to group pedestrian trajectories which were extracted from surveillance videos. Streamline variability plots build upon this approach, yet we propose a number of modifications and extensions to better reveal trends and enable the construction of probabilities of occurrence of streamlines.

5.3. Overview

We take as input a set of n streamlines of m vertices each, which were generated by starting a particle integration at the same location in an ensemble of n vector fields (see Fig. 5.2 (a)). We will also show an example where the streamlines are generated by slightly varying the start position to indicate the effect of these variations on the streamline distribution. In all of our examples, we use a fix-step numerical integration scheme for computing the streamlines, and we parameterize the streamlines via the integration time along the curves. Our method performs a number of operations on the initial streamline set, which are illustrated in Fig. 5.2.

Each trajectory can be seen as a point in a $(d \cdot m)$-dimensional vector space (d is the dimension of the streamline vertices), and this high dimensionality makes processing methods such as finding similarities difficult. Therefore, we first reduce the dimensionality in a statistically optimal way, by projecting the streamlines onto a low-dimensional orthogonal subspace that captures as much of the variation of the initial streamlines as possible. This is illustrated in Fig. 5.2 (b). In a least-squares

sense, the best way to do the projection is PCA, which transforms the streamlines into a simpler representation in an Euclidean space. We will subsequently call this space the *PCA-space*.

To perform a streamline PCA, streamlines are linearized into the rows of a $n \times (d \cdot m)$ matrix. Therefore, all streamlines should have the same number of vertices. However, this is not always the case, because streamlines leave the domain early or might terminate in critical points. Our approach is to fill the missing positions in the matrix by repeating the last streamline vertex. This vertex repetition doesn't introduce new information, because the additional vertices are perfectly correlated, and exactly this can be handled well by PCA. Even though more advanced possibilities exist, for instance, to continue the streamlines with the speed and direction of the last vertex, we have found that neither of them has a significant impact on the clustering and, most importantly, on the appearance of the resulting variability plot.

In the PCA-space the streamlines are clustered into major trends using an appropriate clustering scheme, and for each cluster a multivariate normal distribution—represented by a confidence ellipse and a geometric cluster median in Fig. 5.2 (c)—is fitted to the points. Here, a confidence ellipse describes the set of points that are closer to the mean than a given amount of standard deviations.

Conceptually, the statistical distribution of each cluster is now transformed back into the high-dimensional input space, yielding the corresponding set of streamlines. This is illustrated in Fig. 5.2 (d), where the streamlines correspond to the cluster medians, and the lobes correspond to the streamlines that are within a selected range of standard deviations. Since the operations in PCA-space are performed for each cluster separately, one lobe and one line are generated for every cluster, giving the final streamline variability plot.

5.4. PCA

PCA is a powerful technique for extracting structure from possibly high-dimensional data sets. It is performed by solving an Eigenvalue problem or, alternatively, by computing a singular value decomposition. PCA is an orthogonal transformation of the coordinate system in which the initial data is described. It is often the case that in the new coordinate system a small subset of coordinates is sufficient to account for most of the structure in the data.

PCA is a standard technique in statistics and many other fields, and we only briefly describe the underlying principles here (for more details, see the introduction in Section 4.3.1). We will, however, put special emphasis on the discussion of how the results of a streamline PCA can be interpreted, and how these results can be employed for streamline clustering.

In the following discussion, the i-th streamline is denoted $\mathbf{s}_i$, and every streamline comprises m vertices. Let us note that, in common PCA terminology, each streamline corresponds to an observation, and each vertex corresponds to (multiple) variables. PCA transforms the n streamlines into an

equivalent $(n-1)$-dimensional representation by computing their principal component scores, i.e., the scalar values by which each principal component is weighted to obtain the streamline as a linear combination of these components.

PCA starts by subtracting from every vertex the mean value of all vertices. In our application this has the effect that every streamline is characterized by its offset from the mean streamline $\bar{\mathbf{s}} = \frac{1}{n}\sum_i \mathbf{s}_i$. Consequently, this leads to the following representation, which we will refer to as the PCA-space representation of the streamlines:

$$\mathbf{s}_i = \bar{\mathbf{s}} + \sum_{j=1}^{n-1} c_{ij}\mathbf{u}_j. \tag{5.1}$$

The unit vectors $\mathbf{u}_j$ are the principal components (PC), and the coefficients c_{ij} are the principal component scores. The scores are sorted in descending order of importance, such that $\mathbf{u}_1$ is the direction in which the points $\mathbf{s}_i$ have the largest variance, $\mathbf{u}_2$ is the direction in which the points $\mathbf{s}_i$ have the second largest variance, and so on. Because all $\mathbf{s}_i$ were zero-centered beforehand, we only need up to $n-1$ basis vectors to represent all streamlines. An important property of PCA in our application is that the PCs form an orthonormal basis of the streamline space $\mathbb{R}^{d\cdot m}$. This means that the PCA-space, in which each streamline i is uniquely defined by its PC scores c_{ij}, is an Euclidean space which is equivalent to the original streamline space. I.e., many operations like clustering based on Euclidean distances give identical results in this alternative representation. Let us also note here, that through Eq. (5.1) it is possible to transform a point in the PCA-space into the corresponding representation in the streamline space. We will make use of this to generate streamline variability plots from a statistical model of the scores in the PCA-space.

In many situations where PCA is used for a statistical data analysis, only the PC scores are investigated and visualized. On the other hand, the PCs themselves are often helpful to analyze specific physical features in multiple spatially correlated physical fields. For instance, in fluid mechanic, where PCA is known as Proper Orthogonal Decomposition (POD) [Lum67], periodic patterns can be extracted from turbulent flows as principal components of the time-varying velocity field [MPO07, MEH12]. In meteorology, where PCA is known under the term Empirical Orthogonal Functions (EOF) [Wil11], PCA is used to extract atmospheric phenomena as principal components of scalar field ensembles like geopotential height and temperature [TW98]. Also refer to the concrete examples shown in Section 4.3.1.

The mentioned applications exploit the property of PCA to capture the dominant low-frequency structures in the first PCs, while random fluctuations are expressed in the remaining modes. This effect can also be observed when decomposing streamlines into their PCs, since streamlines can also be considered a type of spatially correlated data. Once a PCA of streamlines has been computed, the

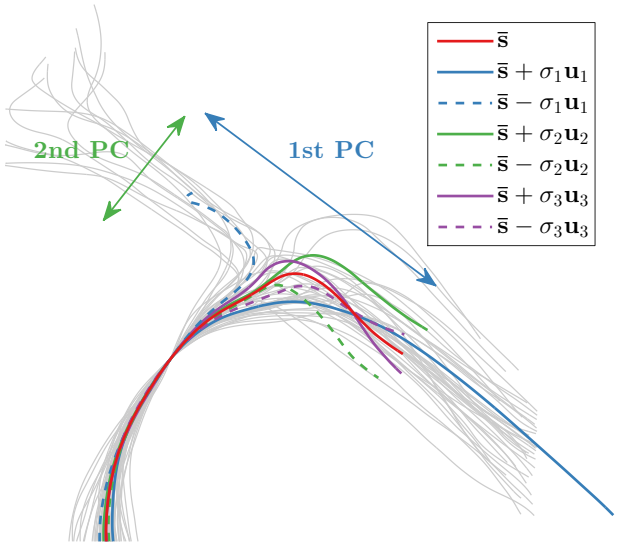


Figure 5.3.: Mean curve $\bar{\mathbf{s}}$ (red) and first three principal components (PC) $\mathbf{u}_1$-$\mathbf{u}_3$ of the 2D streamlines in gray. PCs have to be considered as offsets to the mean curve, so $\bar{\mathbf{s}} \pm \sigma_j \mathbf{u}_j$ are shown instead of $\mathbf{u}_j$. Scaling factors σ_j are the standard deviations of the corresponding PC scores c_{ij}, which emphasize their relative importance. The first PC captures the general deviation of the streamlines in top-left/bottom-right direction, the second PC captures the less important deviation in bottom-left/top-right direction (two-sided arrows). The major trends in the set of streamlines are well represented by the first two PCs.

streamline representation can be reduced to an optimal low-rank approximation, by using only the dominant PCs. This is demonstrated in Fig. 5.3, where the first three PCs of a set of 2D streamlines are shown. It can be observed that the first PCs correspond to streamlines exhibiting very low frequency variations. Furthermore, the third PC crosses over the mean line while the first two PCs do not, which indicates increasing spatial frequency in the higher modes. If more PCs were shown, ever more oscillations around the mean line could be observed.

To obtain an optimal low-rank approximation, one just has to restrict the sum in Eq. (5.1) to the first r components. It can be shown, that the resulting approximations are optimal in a least squares sense, i.e., they minimize the reconstruction error

$$\sum_{i=1}^{n} \left\| \left(\bar{\mathbf{s}} + \sum_{j=1}^{r} c_{ij}\mathbf{u}_j \right) - s_i \right\|^2 .$$

This leaves the question how to determine the appropriate number of components. We use one of the most common cutoff-criteria, by looking at the amount of explained variance that is represented by different choices of r. Let $\sigma_j^2 = \mathrm{var}(c_{1j}, \ldots, c_{nj})$ denote the variance of the j-th PC (notice that the corresponding mean values are all zero). Then the amount of explained variance by the first r

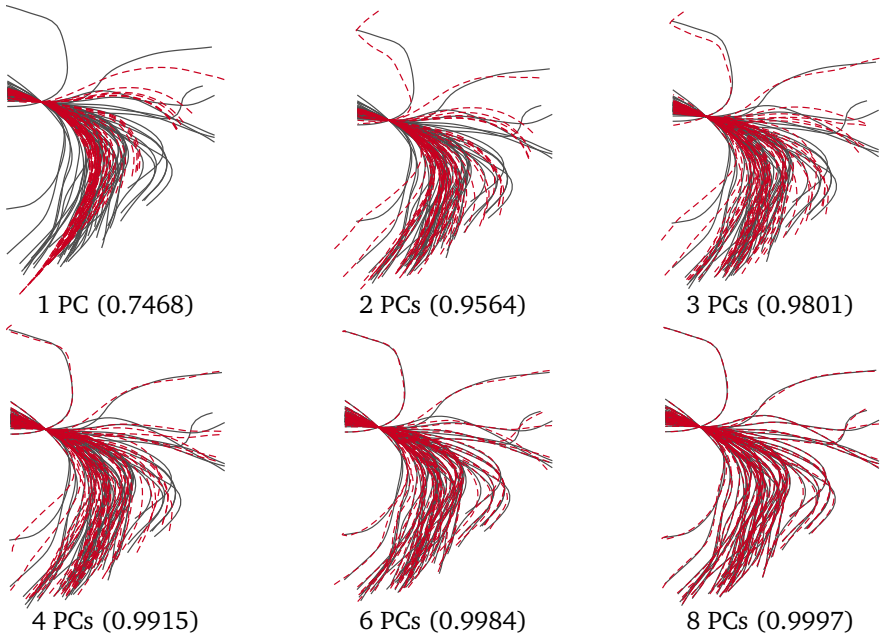


Figure 5.4.: Reconstruction quality: The original streamlines are shown in gray, reconstructions using increasing numbers r of PCs are shown as dashed, red lines. The corresponding amount of explained variance $\mathrm{ex}(r)$ is given in parenthesis (see Eq. (5.2)).

components is

$$\mathrm{ex}(r) = \sum_{j=1}^{r} \sigma_j^2 \Big/ \sum_{j=1}^{n-1} \sigma_j^2. \tag{5.2}$$

For a given explained variance threshold τ, the number of components is chosen as the smallest r for which $\mathrm{ex}(r)$ is greater or equal than τ. Since we perform two different tasks in rank-reduced spaces, which require different degrees of precision, we also use two different thresholds in our approach. On the one hand, for clustering (see Section 5.5), we have found $\tau_1 = 0.99$ to be sufficient. For generating the final plots via splatting of streamlines into a discrete grid (see Section 5.6), on the other hand, slightly more detail is often required, and we use $\tau_2 = 0.999$. In several of our experiments this leads to only three or four PCs that had to be considered, and we never used more than eight PCs in any of our experiments. The resulting approximation errors are depicted in Fig. 5.4.

5.5. Clustering

Once the streamlines have been transformed into the reduced PCA-space of rank r_1, each streamline is represented by a tuple $\mathbf{c}_i = (c_{i1}, \ldots, c_{ir_1})$, i.e., by a (r_1)-dimensional point in PCA-space. Our goal

now is to derive a statistical model of the streamline distribution in this space. We could model our multidimensional data via a single multivariate normal (MVN) distribution, yet often the data includes significantly different trends, showing up as multiple distinct clusters in the PCA-space. A common remedy to this problem is to use a Gaussian mixture model (GMM), which represents the multi-dimensional probability density function (PDF) by a weighted sum of multiple MVN distributions. The Gaussian mixture model is parameterized by the mean vectors, covariance matrices, and mixture weights from all component densities.

A straight forward approach to find a GMM for our data is to fit a given number of MVN distributions to the data using the Expectation-Maximization (EM) algorithm. The EM algorithm can be interpreted as a more general version of the k-means clustering algorithm, which can be applied to MVN distributions. In our application, however, the EM algorithm leads to several problems:

i. Each fitted GMM corresponds to a clustering, yet this clustering often fails to represent the observed trends. Instead, the clusters often overlap in PCA-space and do not show the expected degree of separation. In addition, the multi-dimensional confidence ellipses corresponding to the clusters tend to span empty regions in the PCA-space. If we were to draw new points from the resulting PDF, some points would correspond to streamlines which have low similarity to existing ones.

ii. As mentioned earlier, when visualizing ensembles one often only has few streamlines. This number is very small so that many of the determined clusters do not fully span the reduced PCA-space. As a consequence, the EM algorithm becomes numerically unstable and requires a large regularization parameter.

iii. The need to specify the number of clusters k beforehand requires to run the EM algorithm repeatedly and then to choose the number of clusters based on a score like, e.g., the Akaike information criterion (AIC). Moreover, due to the random nature of the method, it may have to be run several times for each k. Both properties in combination can quickly let the clustering become a performance bottleneck.

In account of these reasons we decided to separate the clustering from the procedure that fits the MVN distributions: We first cluster the streamlines, and then fit an MVN distribution to each cluster. Note that, strictly speaking, we are not using a full GMM, because we do not compute weights for the individual components. Instead, we visualize the MVN distribution of each cluster individually and combine this with a separate visualization of the cluster sizes, i.e., via the thickness of the median lines and the bar plot.

For the clustering to work in combination with the MVN distributions, we have to ensure that it favors compact, elliptical clusters. Since the PCA-space is an Euclidean space, we can draw upon

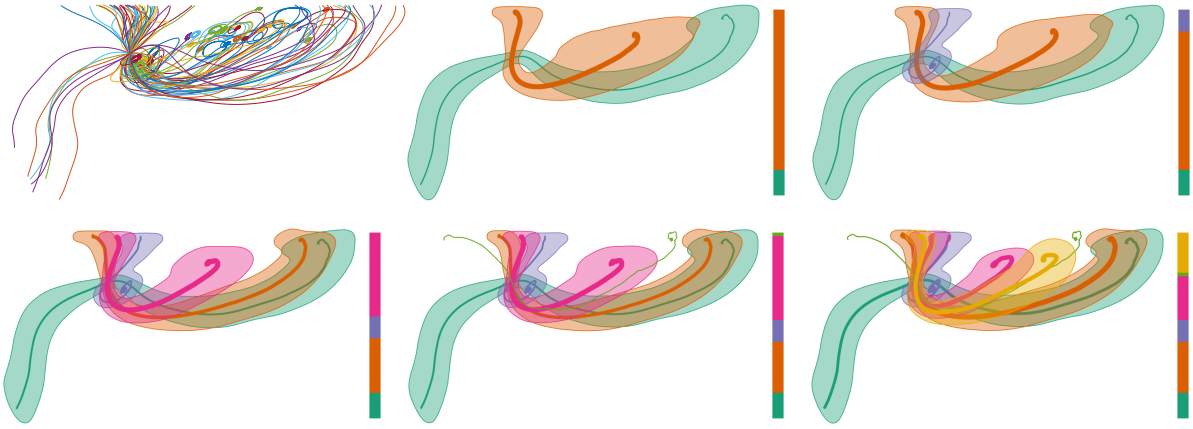


Figure 5.5.: Cluster refinement: For the set of streamlines in the first image, the number of clusters is incrementally increased from 2 to 6. The L-method initially guesses 3 clusters.

many existing clustering approaches. We have performed several experiments with different standard clustering algorithms, and based on the results we ultimately favor Agglomerative Hierarchical Clustering (AHC) in PCA-space.

An introduction to AHC is given in Section 4.3.2. Here, for the sake of completeness, we will only provide a brief summary. AHC creates an unbalanced, binary clustering tree in a bottom-up manner. Starting with each point in a separate cluster (with cardinality one), pairs of clusters are merged successively until all points are contained in one large cluster. The pair of clusters that is combined in each step is determined by a similarity criterion, the so-called linkage criterion, as well as a distance metric, which defines the pair-wise distances between raw data points. As distance metric we use the Euclidean distances in the reduced PCA-space. Common choices for the linkage criterion include single-linkage, complete-linkage, average linking [Sok58] and Ward's Method [Jr.63]. We observed that single-linkage and complete-linkage, favoring connectedness and sphericalness, respectively, yield clusters which do not reveal the major trends in the streamline distribution effectively. Instead, we found average linking, which merges clusters based on their average point-to-point distances, to work best in our examples. Ward's method, which tries to minimize the total within-cluster variance, often delivers similar results. Specifically, AHC in combination with average-linkage yields clusters which very well satisfy our compactness requirement.

The clustering tree resulting from AHC can be split easily into the desired number of clusters (k), and thus allows for an intuitive adaption of the number of clusters. When the number of clusters is changed, the resulting new trend distribution changes in a very coherent and intuitive way: The clusters split and merge recursively according to the binary clustering tree, instead of re-forming completely every time. This effect is demonstrated in Fig. 5.5.

In general, we target a rather small number of less than five clusters, because we are looking for

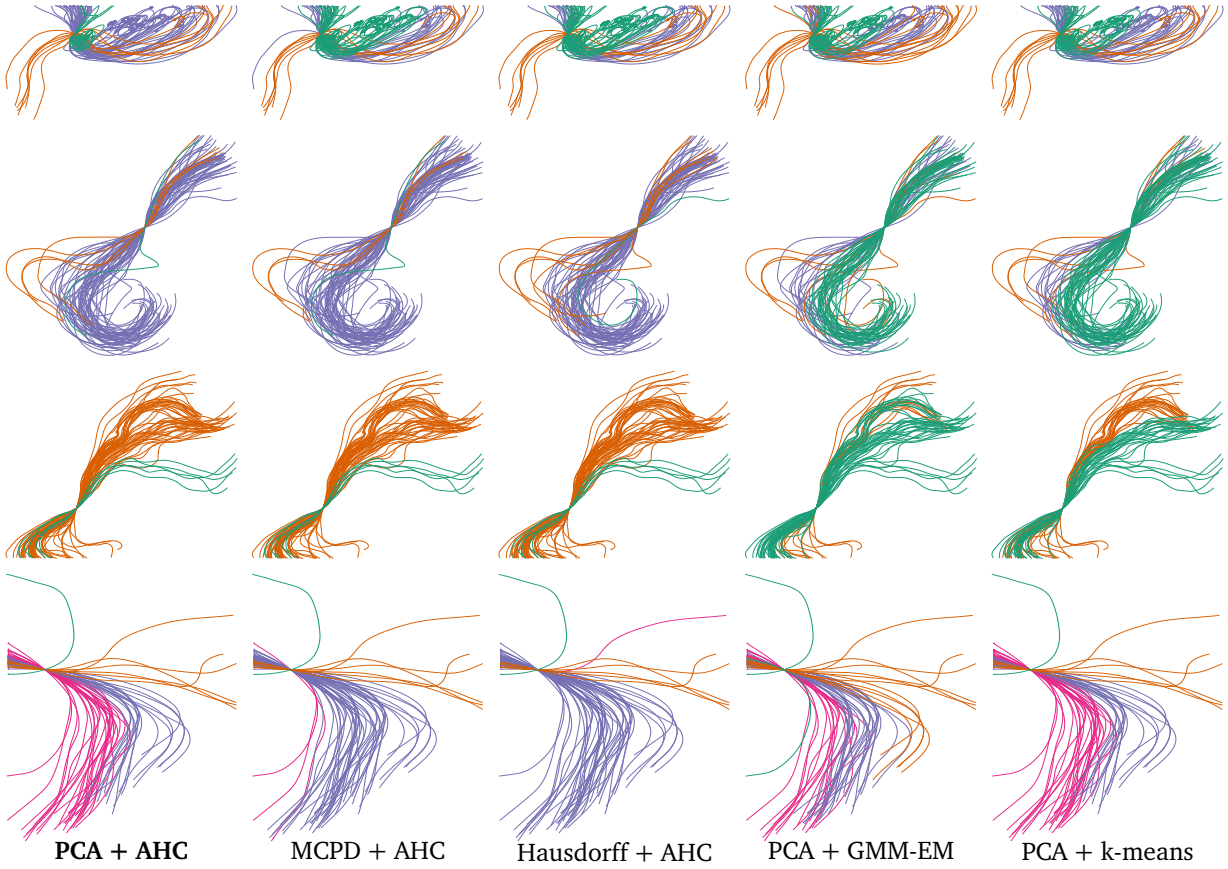


Figure 5.6.: Comparison of different clustering approaches using different sets of 2D streamlines. From left to right: ***PCA + AHC***, our hierarchical clustering based on the Euclidean distances in rank-reduced PCA-space. *MCPD + AHC*, hierarchical clustering using the mean-of-closest-point distance [CGG04]. *Hausdorff + AHC*, hierarchical clustering using the Hausdorff distance. *PCA + GMM-EM*, clustering via the EM algorithm for Gaussian mixture models using the Euclidean distances in rank-reduced PCA-space. *PCA + k-means*, k-means clustering using the Euclidean distances in rank-reduced PCA-space. Average linkage was used in all AHC methods. For the examples in each row, the same number of clusters was prescribed.

the major trends in the streamlines and noticed that the visualizations can become populated when more clusters are used. We found that in most of our cases the L-Method [SC04] provides a very good initial guess for k, where we let the method choose $k \in \{2, ..., 10\}$. Since this method can yield slightly inconsistent results for very low numbers of clusters, we have slightly adapted it (see Section 4.3.3). Nonetheless, since no automatic criterion can give perfect guesses in all possible cases, we have to manually adjust k by ± 1 in some cases to get the most intuitive clustering.

In Fig. 5.6, we compare our clustering results with those obtained by alternative clustering approaches for streamlines or other types of line data. The comparison indicates that k-means clustering and the EM algorithm for GMMs (both performed in the Euclidean PCA-space) often prefer equally sized clusters over separating trends of diverging streamline sets. On the other hand, AHC

based on mean-of-closest-point-distances [CGG04] and Hausdorff distances often tends to misclassify individual streamlines. Our clustering seems to best extract the dominant *trends* in the data, and it is able to robustly handle complex situations. This can be seen, for instance, in the top row of Fig. 5.6, where our approach is the only one that can separate the small set of highly curved streamlines colored in green in the left image. It is also important to note that, irrespectively of the quality of the individual clustering approaches, most of them are not suited for the construction of variability plots as proposed in our work. As we will explain next, these plots are constructed by using the multivariate distribution of streamlines in some (rank-reduced) space, i.e., the PCA-space, excluding those clustering approaches not working in such a space.

5.6. Generation of Streamline Variability Plots

Up to this point, the set of streamlines has been partitioned in PCA-space, and the distribution of each cluster has been approximated with an MVN distribution. Then, our principle idea is to transform a geometric representation of these distributions in the PCA-space back to the domain space in which the original streamlines reside, in order to obtain for each cluster a confidence lobe illustrating the variance and spread of the respective trend. This transformation is possible through Eq. (5.1), which tells us that an arbitrary point in the (rank-reduced) PCA-space can be transformed to a corresponding streamline. If the point was taken in agreement with one of the MWN distributions, then the resulting streamline follows the statistics of the corresponding cluster, even though no such streamline existed in the initial set. By this, we can, in principle, generate arbitrary many new streamlines whose shapes follow the statistical properties encoded by the different clusters in PCA-space.

In the generation of these streamlines a certain approximation error is introduced, because we truncate the PCs used for reconstruction (c.f. Fig. 5.4), yet it is important to recall that this error is bounded through Eq. (5.2) and restricted to the high-frequency details that are captured by the higher PCs. This means that all major trends will be captured if we use a "sufficient" number of PCs. On the other hand, the restriction to the first r_2 PCs yields new streamlines exhibiting a certain amount of smoothness, providing a visually appealing cluster representation.

From a statistical point of view, the number of samples that are used to fit the MVN distributions—i.e., the cardinality of the clusters—is relatively small compared to the number of dimensions (r_2) of the rank-reduced PCA-space. Therefore, small variations in the samples can significantly change the geometric representations of the MVN distributions in PCA-space. On the other hand, all of our experiments have shown that these changes have only a minor effect on the geometry of the corresponding lobes.

Confidence Lobes

To represent the MVN distributions in PCA-space, we use confidence ellipses (or contours). Let μ_k and Σ_k denote the mean and covariance matrix of cluster k, respectively. Then the confidence ellipses are defined as iso-contours of the so-called Mahalanobis distance

$$d_k = \sqrt{(\mathbf{x}-\mu_k)\Sigma_k^{-1}(\mathbf{x}-\mu_k)},$$

where $\mathbf{x}$ denotes an arbitrary point in $\mathbb{R}^{r_2}$. Intuitively, the Mahalanobis distance d_k indicates for the point $\mathbf{x}$ how many standard deviations it is away from μ_k. A confidence level, i.e., a level-set in the distance field, is specified via an iso-value in numbers of standard deviation (also see Section 4.3.4).

In principle, it would be very appealing to do so by specifying a quantile, for instance, so that $d_k \leq \alpha$ corresponds to the 50% innermost points. Unfortunately, this leads to a very counter-intuitive behavior of the generated lobes, because it makes the threshold α sensitive to the dimension r_2 of the reduced PCA-space. In particular, increasing r_2 to make the line approximation more accurate causes the corresponding confidence regions to grow, since the threshold α has to be increased [1]. Therefore, we threshold d_k against a fixed α, which can be chosen and varied when creating the confidence lobes.

If we create a lobe with, e.g., $\alpha = 1.0$, it will cover the range of locations containing all streamlines that are within one standard deviation of each trend. While small thresholds like $\alpha = 1.0$ lead to very tight confidence lobes, higher thresholds with $\alpha \geq 2.0$ lead to convex-hull like shapes which sometimes "overshoot". This effect is demonstrated in Fig. 5.7, where for a set of streamlines the confidence lobes for different thresholds α are shown. We use a default value of $\alpha = 1.5$ in all of our experiments in this chapter, if not stated otherwise.

To transform a confidence ellipse in PCA-space to its corresponding domain space representation, we must determine the locations in domain space which are covered by at least one streamline that corresponds to a point in the interior of this ellipse in PCA-space. Since determining this is not directly possible for a given point in domain space, we follow a different approach using Monte-Carlo sampling and line drawing.

We draw random points uniformly from within the confidence ellipse using a random number parameterized with the corresponding μ_k and Σ_k, and compute their streamline representations using Eq. (5.1). Each of these streamlines is splatted additively into the cells of a uniform grid discretizing the domain space. Note that the original streamlines are not used in this process. In this way we build a visitation map as proposed by Buerger et al. [BFMW12], and we then extract the confidence lobe using the smallest iso-value that still allows for smooth iso-contours.

[1] The Mahalanobis distance d_k is distributed with a χ^2_K-distribution, which changes depending on the degrees of freedom K. Here, K corresponds to r_2.

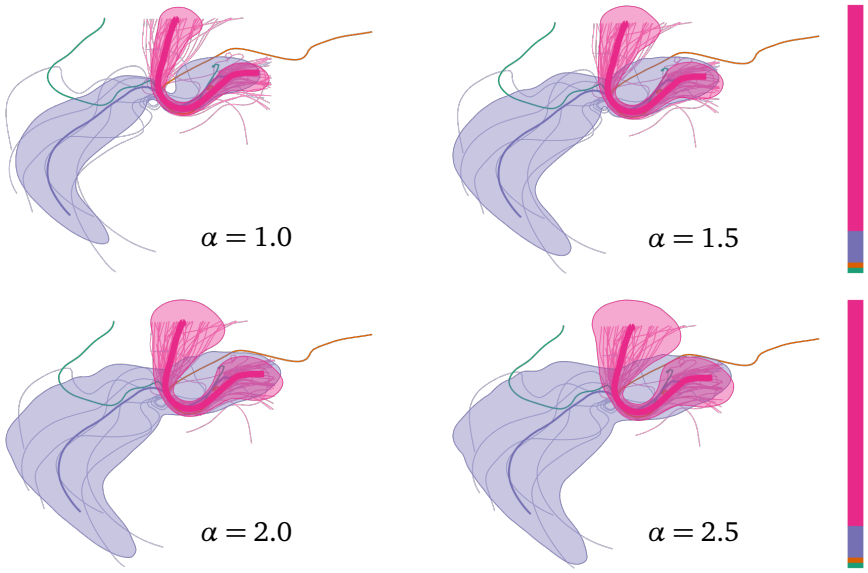


Figure 5.7.: Effect of the Mahalanobis threshold α on the confidence lobes: The larger α, the larger the lobes become. For $\alpha >= 2.0$ the lobes contain almost all associated lines but also tend to "overshoot". Orange and green trends contain single outliers, and hence no lobes are generated.

The procedure of uniformly sampling a confidence ellipse is described in Section 4.3.4. Note that it would also be possible to sample the boundary of the confidence ellipse instead of the volumetric region enclosed by it. However, this makes is harder to draw points that are distributed uniformly with respect to the Euclidean space into which the ellipse is embedded.

Building the visitation map is performed via rasterization of the streamline vertices, which are treated as particles, or splat-kernels, of a certain diameter. This approach yields sufficient results in our application, and it is both faster and easier to implement than accurate line-splats using point-to-line distances. We use a small bi-/trilinear splat-kernel with a support of 4 and 8 texels in 2D and 3D, respectively, and use a second pass after splatting to smooth the visitation map in order to increase the quality of the resulting iso-contours. We use visitation maps—realized as 2D and 3D accumulation textures—with a resolution of roughly 200 texels along the longest dimension. In 2D, we use a fixed number of 1000 streamlines to generate the visitation map for each cluster, and we draw for each lobe the outer contour and uniquely color its interior. In 3D, we found 5000 streamlines to be sufficient to obtain representative lobes, and we generate the visitation map directly on the GPU and render the lobes via iso-surface ray-casting. If the vertex density along the streamlines is too low, we add new vertices by linearly interpolating between consecutive vertices.

Streamline-Median

The abstract shapes of the confidence lobes are further enhanced by a single streamline representing the corresponding trend as accurate as possible. We therefore introduce a new concept of streamline-median. Since the reduced PCA-space is an Euclidean space, we build upon existing concepts here. Specifically, we use the so-called geometric median, which is an extension of the one-dimensional median to multiple dimensions. Given a set of points, the geometric median is defined as the point in space—not necessarily coinciding with one of the initial points—which minimizes the sum of Euclidean distances to all initial points. It can be calculated iteratively using Weiszfeld's algorithm.

We hence determine the geometric median for every cluster in the PCA-space and reconstruct a streamline—the streamline-median—from it. This means that the streamline-medians in our visualizations are not streamlines from the initial set, but they are artificial streamlines being closer to all initial streamlines than any other streamline. On the other hand, following the same argumentation as for constructing the confidence lobes, we know that this artificial streamline shows the general trend represented by a cluster and is free of high-frequent details which are not common to all cluster members. When drawing the streamline-median, we further use its thickness to give a qualitative visual cue indicating the relative strength of the trend. I.e., the more initial streamlines follow the trend, the thicker the streamline-median is drawn.

We further annotate each streamline variability plot to the right. The color bar shows the relative number of trajectories represented by each lobe, further enhancing the plot about qualitative information.

5.7. Results

In this section, we analyze the performance of our method, show additional results and perform a comparison of streamline variability plots to curve boxplots.

Datasets

The 2D streamline examples we use in this chapter were created from wind fields of numerical weather prediction data obtained from the ECMWF Ensemble Prediction System (ENS), which comprises 51 ensemble members. We use forecast runs initialized at 00:00 UTC on October 15th and 17th, 2012, and perform massless particle integration to obtain streamlines in a single steady forecast at a later time. Each 2D streamline is comprised of 300 vertices. Additional 2D examples are shown in Fig. 5.8. In the 3D examples (see Fig. 5.9), pathlines were computed for the first 144 hours of the forecast and for each ensemble member using the LAGRANTO model [SW15], which considers air masses and specific meteorological aspects rather than massless particles. We perform 200 integration steps to

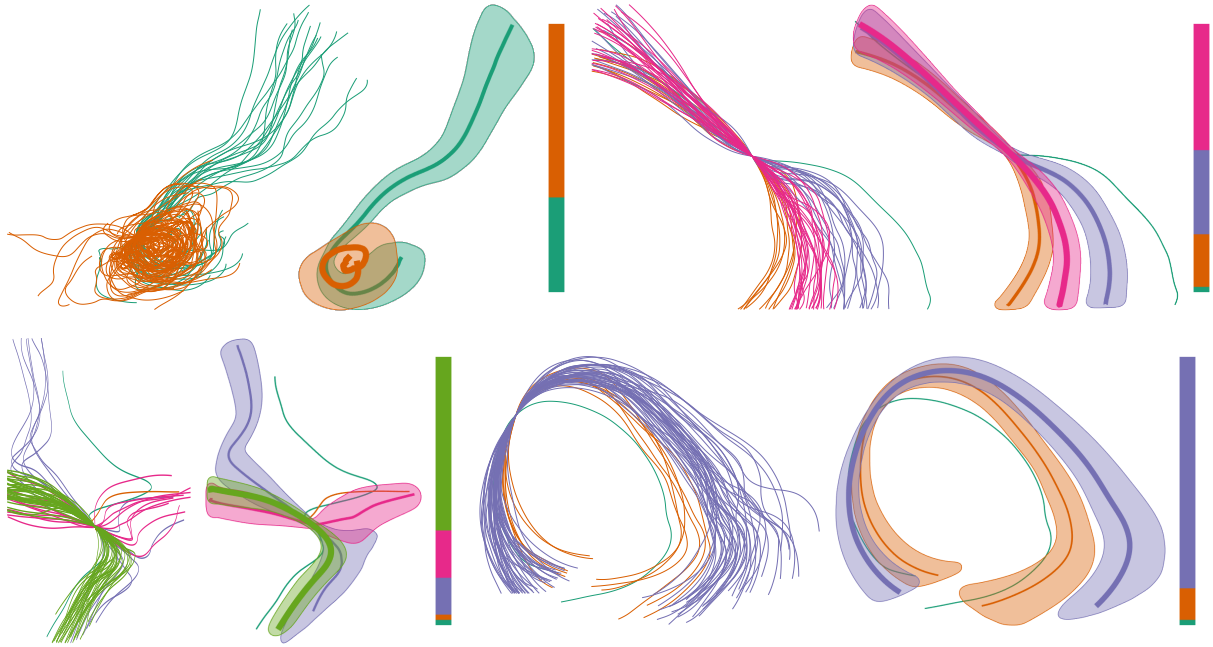


Figure 5.8.: 2D streamline variability plots and corresponding spaghetti plots.

generate these pathlines in 3D. In addition, we further use an ensemble of 50 steady 3D flows from a simulation of a fluid flow around a cylindrical obstacle with varying Reynolds numbers (see Fig. 5.10). Streamlines are computed using 500 numerical integration steps.

Implementation & Performance

All the results in this chapter were generated on a standard desktop PC equipped with an Intel Xeon X5675 processor with 3.0GHz×6 cores, 8GB RAM and a NVIDIA Geforce GTX 680. We use the Matlab implementations of PCA and AHC, and our own CPU implementation to fit an MVN to the data in PCA-space and generate new random variables respecting these distributions (see Section 4.3.4), including the streamline-median. In 2D, splatting of streamlines into a 2D grid, extracting iso-contours in this grid, and drawing the resulting filled confidence lobes are performed on the CPU. In 3D, splatting is performed on the GPU, where the confidence lobes are directly rendered via iso-surface ray-casting.

In all of our test scenarios, the time required to compute a PCA, cluster the data, and fit an MVN to this data was below 50ms, even for a bundle consisting of 200 streamlines with 500 vertices each. The most time consuming step is splatting the generated lines into the visitation map. On the CPU, roughly 10000 trajectories can be splatted into the 2D map per second, and splatting into a 3D map on the GPU can be performed at a rate of roughly 50000 trajectories per second. This performance gain is mainly due to the fast memory interface on the GPU and the possibility to process many trajectories

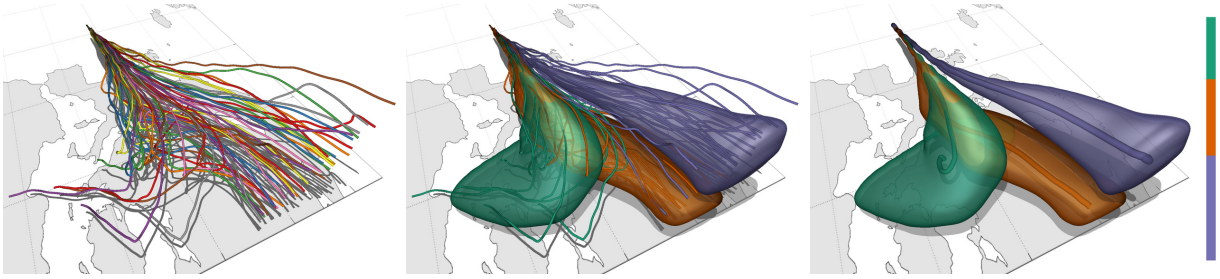


Figure 5.9.: Variability plot of pathlines in a time-varying 3D ensemble comprising 51 members. **(left)** Spaghetti plot. **(middle)** Pathlines colored by cluster membership, and confidence lobes. **(right)** Variability plot with confidence lobes and pathline-medians.

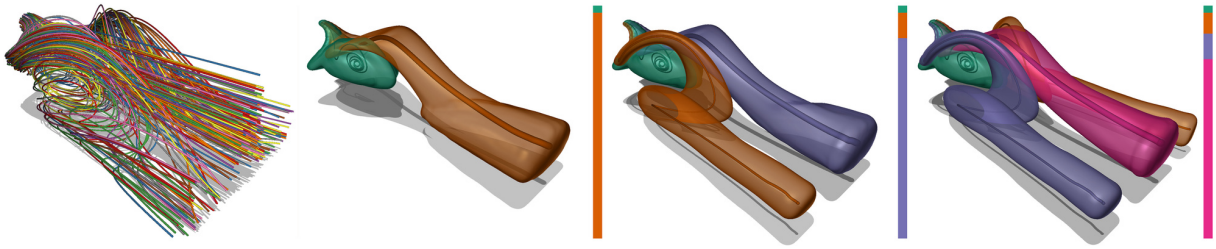


Figure 5.10.: Variability plots of 200 streamlines with jittered positions in an ensemble of 50 steady 3D flows around an obstacle. **(left)** Spaghetti plot of the streamlines. **(middle-right)** Variability plots are shown, where the number of clusters increases incrementally from 2 to 4. The initial guess for the number of clusters is 3.

in parallel. Overall, all variability plots shown in this paper were generated in less than 500ms, given the initial set of trajectories, and were rendered at interactive rates.

3D Variability Plots

In the following we further demonstrate the effectiveness of variability plots to depict the major trends in 3D trajectories. It should be emphasized that the generation and visualization of 3D variability plots does not require any specific algorithmic modifications of our approach, besides the use of a 3D visitation map into which the trajectories are splatted and from which the lobes are rendered.

Fig. 5.9 shows a 3D variability plot for an ensemble of 51 pathlines in the ECMWF ensemble. It can be seen that the major trends in the data are very well separated by the variability plot, and that the abstract representation using lobes and streamline-medians provides a fairly uncluttered view compared to the spaghetti plot. The particular example demonstrates, that the variability plot can not only convey the major trends but also give a very good indication of where these trends start to separate. Especially this property is extremely helpful in meteorological applications, where the locations need to be analyzed where the divergence of predicted forecasts is going to start.

Fig. 5.10 shows variability plots of 200 streamlines, which were generated by seeding in each of the

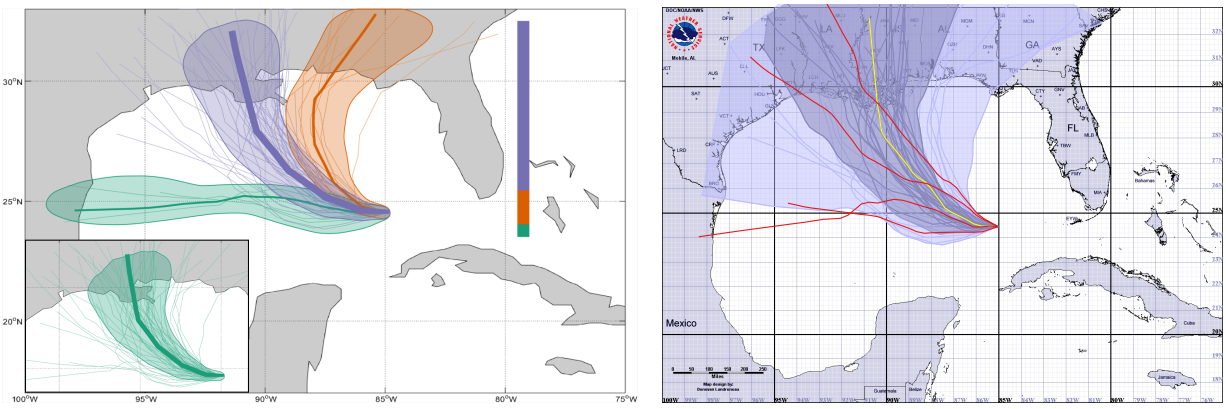


Figure 5.11.: Comparison of a streamline variability plot **(left)** to the curve boxplot from [MWK14] **(right)** for an ensemble of 50 simulated hurricane tracks, generated by the method presented by Cox et al. [CHL13]. The inset shows a second variability plot, where the number of clusters was manually set to one. In the boxplot, the light and dark cones are the bands which contain 50% and 100% of the curves, respectively. Red lines show outliers and the yellow line is the global median line. In both figures the original tracks are shown in the background of the plots, where the colors correspond to band-depth and clusters, respectively.

50 ensemble members 4 streamlines that were randomly jittered around a common seed point. The number of clusters is adjusted incrementally from 2 to 4, while the initial guess by the L-Method is 3. Here, it can be seen how an increasing amount of clusters can lead to an increasing amount of detail in the variability plot, until a good representation of the trends is reached. The most representative plot is the last one, which reveals four significantly different trends in the streamlines and effectively conveys the symmetry in the data well.

Comparison to Curve Boxplots

Fig. 5.11 compares a streamline variability plot to a curve boxplot from [MWK14] for a set of simulated hurricane tracks. The boxplot illustrates the distribution of trajectories via two nested bands containing 50% and 100% of the trajectories, respectively. In addition, a representative median trajectory, i.e., the deepest trajectory from the initial set, and outliers are depicted. The boxplot has been generated using the entire set of trajectories, and no initial clustering was performed.

The streamline variability plot reveals three trends in the hurricane trajectories. A small number of trajectories deviates to the west and north-east—which is captured by two minor trends—while the dominant third trend in the center contains over 75% of the trajectories. The confidence lobes of the two smaller trends—in relation to the regions that are covered by the corresponding trajectories—are more wide-spread than the trend in the center. This means that there is a higher intra-cluster variance in the smaller trends, whereas the trajectory distribution of the dominant trend is more focused towards the region that is indicated by its confidence lobe and median line. A second variability plot

is also shown, for which the number of clusters was manually set to one. This results in a single, wider lobe which is similar to the 50% band of the boxplot. The difference is, that the boxplot cone shows the region that contains 50% of the innermost curves, while our lobe shows the region that contains all curves that are within the range of $\alpha = 1.5$ standard-deviations. Furthermore, the length of the curves is conveyed differently in the two visualizations. On the one hand, the "length" of the boxplot cone corresponds to a maximum curve length, because it is constructed as a hull around all represented curves, and the yellow global median line shows the length of a single representative line. On the other hand, in the variability plot, the "length" of a lobe is typically similar to the length of its streamline-median, which is approximately equal to the median length of all represented curves.

The comparison of boxplots and variability plots clearly indicates the different use cases of both approaches. The boxplot intends to visualize the entire spread, or enclosed spatial band, of a set of trajectories, in addition indicating the percentage of trajectories being contained in sub-bands as well as outliers. The variability plot intends to detect and reveal the major trends in the data, and then plots these trends via confidence lobes to depict the probability of occurrence of the trajectories. Thus, the variability plots support a probabilistic analysis of the shapes of the major trends, rather than emphasizing the overall spread of the data. The clustering of the initial trajectories into major trends, on the other hand, can be used as a preprocess to curve boxplots, so that each trend could be represented by a separate boxplot. Thus, we could even combine both approaches, by replacing individual confidence lobes in our plots with curve boxplots.

5.8. Conclusion

In this chapter we have presented a new approach for the visual exploration of the major trends in sets of streamlines extracted from ensemble flow fields. Our approach is specifically tailored to the visualization of trends in rather small sets of streamlines, as it is typically the case when dealing with routinely simulated meteorological ensembles. Even from such sets we can faithfully reconstruct confidence lobes showing the probability of occurrence of streamlines over the domain. By using stochastic models of clusters in PCA-space, we can generate new streamlines exhibiting the statistical properties of the shapes and positions of the major trends. The method is applicable to 2D and 3D data, and the abstract visualizations we present allow to communicate effectively salient characteristics of the data distributions even in 3D.

6

Contour Variability Plots with Visualization of Global Correlations

In this chapter, we present the second application of variability plots to ensembles of iso-contours, which are extracted from an ensemble of scalar fields according to a user-selected iso-value. The major difference to the application to streamlines in the previous chapter is the initial representation of the input data, which provides the starting point for the PCA based analysis of variability plots. In the case of iso-contours, we use signed distance functions for the purpose of this initial representation. This does not only result in a high-quality clustering of the input ensemble of iso-contours into trends, but allows for two important improvements. First, we can perform the transformation of confidence ellipses from the PCA based feature space to the initial domain space analytically, not requiring an expensive random sampling and kernel density estimate as in streamline variability plots. Second, we can exploit the resulting properties of the PCA based feature space to estimate the conditional probability of the occurrence of a contour at one location given the occurrence at some other location. Based on these probabilities, we extend the basic visualization of *contour variability plots* by a special color-coding which reveals global correlations between the occurrences of iso-contours at different locations in the domain. We demonstrate the use of the resulting visualization for ensemble exploration in a number of 2D and 3D examples, using artificial and meteorological data sets.

6.1. Introduction

To analyze the *uncertainty* that is represented by an ensemble of iso-contours, the variability of the ensemble members needs to be characterized, and the major trends and outliers in the shape and

This chapter is based on material that has been originally published in F. FERSTL, M. KANZLER, M. RAUTENHAUS, R. WESTERMANN: Visual Analysis of Spatial Variability and Global Correlations in Ensembles of Iso-Contours. *Computer Graphics Forum (Proceedings of EuroVis)*, 35(3):221–230, 2016.

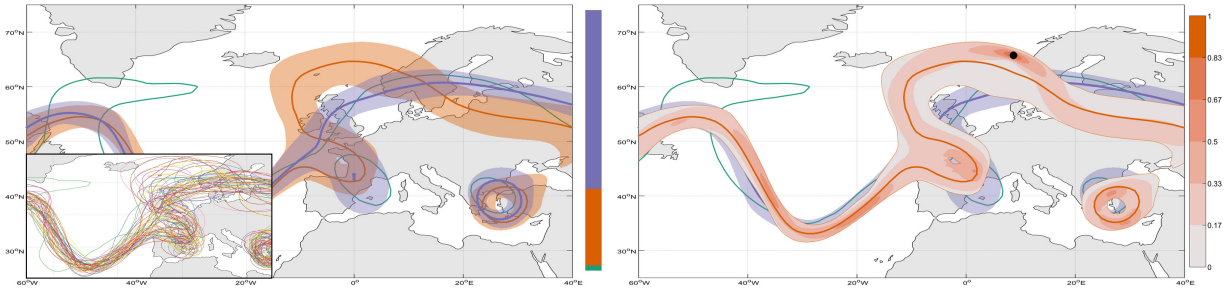


Figure 6.1.: From a set of iso-contours (inlay), we compute an abstract visualization of the uncertainty that is carried by this set in the form of a contour variability plot **(left)**. The visualization is further refined by showing correlations, i.e. probabilities for the joint occurrence of iso-contours in a selected region (black circle) and other locations along the main trends using color coding **(right)**.

spatial location of the contours need to be determined. Similar to parameterized curves like streamlines, a popular approach to visually analyze an ensemble of iso-contours is a so-called spaghetti plot, which shows the contours simultaneously in a single image.

The inset in Fig. 6.1 (left) shows a spaghetti plot of iso-contours—distinguished by color—in a 2D geopotential height field. Since iso-contours can have very complicated shape and topology, spaghetti plots produce visual clutter when many contours overlap, and major trends, outliers, and statistical properties of the contour distribution cannot easily be conveyed. Thus, it is difficult to draw conclusions from a spaghetti plot on the different topological and structural characteristics of the contour set. For iso-contours in 3D fields, this problem becomes even more severe.

To overcome the limitations of spaghetti plots, Whitaker et al. [WMK13] and Mirzargar et al. [MWK14] have introduced visual abstractions for curve-like features based on the concept of statistical data depth, which enables the computation of a global ranking of such features and the use of this ranking to derive probabilistic indicators of the feature occurrence around the median. In contrast to this, in Chapter 5 we proposed to model statistically the distribution of streamlines, and to derive clusters indicating the major trends represented by an ensemble. By visualizing these clusters instead of the entire set of streamlines, the main trends can be conveyed effectively to the user.

In this chapter we show how to adapt the idea of variability plots to iso-contours in multi-dimensional scalar fields. This is challenging, because it first requires transforming the contours into a representation from which their similarity can be inferred. In contrast to ensembles of streamlines, which start at the same seeding position and can be computed using the same path-parametrization, such a transformation is not immediately given for iso-contours. In principle, geometry based approaches for curve clustering [OLK*14] can be employed, yet they have difficulties when the contours exhibit different topologies and no consistent parameterization is given, which is common in real-world applications.

In addition to analyzing the major trends given by an ensemble of iso-contours, it is also interesting

to investigate the relationship between the occurrence of a contour at different locations. In particular answering the question about the probability that a contour going through a specific location is also going through some other location can help to further explore the distribution of the iso-contours within each cluster. Especially when dealing with iso-contours or pathlines in time-varying fields, the analysis of such joint occurrences enables the exploration of interesting interrelations between the data properties at different locations and simulation times, and it can even help making predictions on the possible occurrences of features. To the best of our knowledge, a visual analysis approach for this kind of dependencies has not been proposed so far.

In this work, we address the aforementioned problems and propose new methods to explore the distribution of iso-contours in multi-dimensional scalar fields. Our specific contributions are:

- Building upon a distance field transformation of iso-contours [TN14, BM10] to obtain a consistent contour parametrization, we transform the contour ensemble into a reduced order space. Analogous to streamline variability plots, we extract the major trends via clustering in the reduced order space, yet instead of random sampling fitted multivariate normal distributions we show how to derive analytically the corresponding visual abstractions in the spatial domain. Altogether, this process yields *contour variability plots*.
- We provide a mathematical formulation for the problem of computing the joint occurrences of contours at different locations via integration in the reduced order space over a region bounded by hyper-planes.

In addition, we provide an interactive picking mechanism and graphically highlight joint occurrences with the picked region using color coding.

6.2. Related Work

Contour variability plots belong to the broader class of uncertainty visualization techniques. They fall into the category of ensemble visualization and use curve clustering for computing confidence bands for sets of iso-contours.

In a number of works, overviews and taxonomies of uncertainty visualization techniques have been given [PWL97, JS03, BHJ*14, OJ14]. Related to our method are uncertainty visualization techniques which encode visually the positional variation that is caused by the uncertainty on specific features, for instance, the positional variability of surfaces in space. Techniques include the visualization of confidence surfaces [PWL97], surface diffusion techniques [GR04] as well as surface animations [Bro04]. Some more recent approaches [PRW11, PH11, PWH11] model the uncertainty stochastically and derive probability distributions for particular stochastic events associated to iso-surfaces.

The visualization of feature variability in ensemble fields is often performed via spaghetti plots of selected contour lines or threshold probabilities of 2D fields such as surface wind speed [PWB*09, Wil11]. Glyphs and confidence intervals were introduced to emphasize the Euclidean spread of contour and curve ensembles [SZD*10, MWK14] and, specifically, contour boxplots [WMK13] have been applied to weather forecast data [QM16]. Locations in 3D flow ensembles are characterized based on the divergence of transport patterns [HOGJ13].

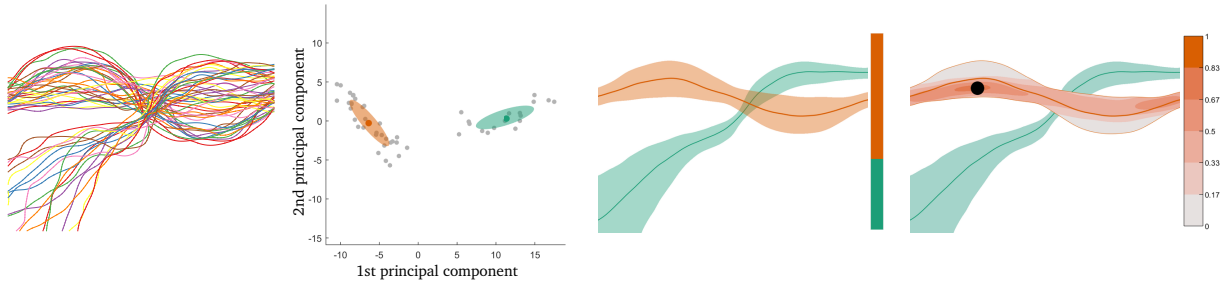
The analysis of mutual data dependencies has mostly been addressed via the visualization of data correlations. For instance, via color mapping and slicing [JPR*04], by means of correlation fields and multi-field graphs [STS06], via glyph based visualizations of local covariance structures [KWL*04], or by correlation clustering to group objects with similar pair-wise correlations [BBC04, PW12].

Contour variability plots are also related to clustering approaches for parametrized curves in 2D and 3D space, which are most often based on geometric similarity measures. For an overview of similarity measures using geometric distances between curves let us refer to the comparative study by Zhang et al. [ZHT06] and, alternatively, for an overview on probabilistic model based curve clustering algorithms let us refer to the work by Gaffney [Gaf04]. Agglomerative Hierarchical Clustering (AHC) with different cluster proximity measures has been shown for curves by McLoughlin et al. [MJL*13]. The overview article by Oeltze et al. [OLK*14] evaluates different clustering approaches for streamlines using geometry based similarity measures. Thomas et al. [TN14] cluster iso-contours in a rotation and scale invariant feature space to find (self-) symmetries in iso-surfaces. To find similarities between different iso-contours of the same scalar field, Carr et al.[CBB06] use histograms and iso-surface statistics whereas Bruckner and Möller [BM10] use distance functions and information-theoretic measures.

An alternative to geometry based curve clustering approaches are dimension-reduction techniques, where the initial curves are represented in some low-dimensional subspace. For shape analysis, Rathi et al. [RDT06] use kernel PCA on distance functions for shape analysis, and Leventon et al. [LGF00] employ the power of PCA on distance functions and use a multivariate normal distribution as a statistical shape model. Fofonov et al. [FML16] use distance functions and dimension reduction to visualize a set of contour time series as 2D or 3D trajectories.

6.3. Overview

Given an ensemble of 2D or 3D scalar fields $\mathbf{s}_1, ..., \mathbf{s}_n \in \mathbb{R}^M$ (defined on the same grid comprised of M vertices), we consider the iso-contours which are extracted from each ensemble member $\mathbf{s}_i$ for the same iso-value v. Note that we are not considering value uncertainty in this work (i.e., the uncertainty in the choice of the iso-value itself) and therefore do not consider the possibility that different ensemble members can contain similar iso-contours with different iso-values. A comparative analysis of iso-contours is difficult, because they often consist of many different, disconnected parts



(a) Input in domain space

(b) Statistical model in PCA-space $\mathbb{R}^r$ ($\approx$ SDF-space $\mathbb{R}^M$)

(c) Contour variability plot

(d) Correlation visualization

Figure 6.2.: Overview: **(a)** Input set of iso-contours. **(b)** Statistical model of the corresponding signed distance functions in the principal component basis, represented by confidence ellipses and geometric cluster medians. **(c)** Contour variability plot, obtained by transforming ellipses and medians back to domain space (with cluster strengths indicated by a bar plot). **(d)** Visualization of global correlations in the orange cluster by picking a circular region (black) and color coding the conditional probability of contours going through other regions, given that they go through this picked region.

and can vastly differ in length, i.e., they vastly differ in the number of degrees of freedom. To account for this, we fix this number to the number of grid vertices M, by representing the contours implicitly via signed distance functions (SDF). For each contour i of scalar field $\mathbf{s}_i$ we compute a corresponding SDF $\mathbf{d}_i \in \mathbb{R}^M$, which specifies at each grid point of $\mathbf{s}_i$ the signed distance to the closest point on the contour (with the sign indicating on which side of the iso-contour the point is located). While, in principle, each $\mathbf{d}_i$ is a grid of signed distance values, it can also be interpreted as a vector with M entries and, hence, as a point in the Euclidean space $\mathbb{R}^M$ which directly corresponds to contour i. We will subsequently refer to $\mathbb{R}^M$ as *SDF-space*.

By using the SDF representations as high-dimensional feature vectors, we can, in principle, perform the same steps as for generating streamline variability plots. An overview of the resulting algorithm for ensembles of iso-contours is shown in Fig. 6.2. We call the resulting confidence regions *bands* (as opposed to *lobes* for streamlines), which can be generated in 2D and 3D using equivalent operations.

From the SDF representation of the input contours (Fig. 6.2 (a)), we compute a statistical model of the contour distribution. This is required to obtain a mechanism for generating "continuous" distributions in the spatial domain. To this end, we perform a principal component analysis (PCA) of the M-dimensional point set $\mathbf{d}_1, ..., \mathbf{d}_n$. This is depicted in Fig. 6.2 (b), where the gray points are the points $\mathbf{d}_i$ in the resulting principal component basis (for illustration purposes the depiction is restricted to the projection to the first two components), and each of those points corresponds to one contour from Fig. 6.2 (a). Subsequently, we determine a clustering of the transformed points in the so-called *PCA-space*. By fitting a multivariate normal distribution (MVN) to each cluster, clusters can be represented by their geometric median and a confidence ellipse with a user-defined size in units of standard deviation (Fig. 6.2 (b)). E.g., the points in Fig. 6.2 (b) are grouped into two clusters,

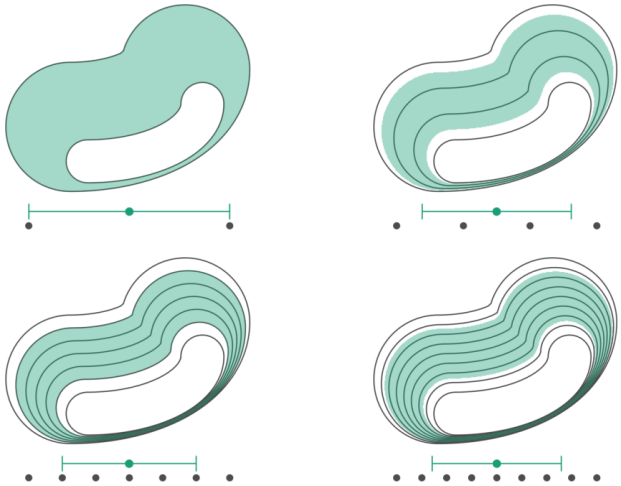


Figure 6.3.: Standard deviation of distance functions: Four different ensembles of equally spaced iso-contours (black lines) and corresponding confidence bands (colored green) are shown. Bands indicate the regions within one standard deviation from the mean contour. Below each image is a corresponding 1D plot of equally spaced points and an error bar showing mean and standard deviation.

and each cluster is represented by a confidence ellipse with a size of one standard deviation (colored ellipses) and its geometric median (colored dots).

For streamline variability plots, we had to perform a dense sampling of the ellipses to generate new streamline realizations following the main trends in the data, and these realizations were transformed into the spatial domain to form the variability lobes. We show in our work that for sets of iso-contours, represented via distance functions, the shape of the corresponding bands is fully determined by a box-shaped region in PCA-space. From this observation we derive a method to analytically transform the confidence ellipses in PCA-space into their corresponding representations in the spatial domain. This transformation yields SDFs from which a *contour variability plot* can be derived (Fig. 6.2 (c)). Conceptually, the visualized bands correspond to the boundaries of confidence ellipses in PCA-space and illustrate the standard deviations of the per-cluster sets of contours, which is illustrated in Fig. 6.3.

Since box-shaped regions can be used in PCA-space instead of oriented ellipses to generate exactly the same bands, some information that is given by the ellipses' orientation is lost. The orientation is determined by the off-diagonal entries of the covariance matrix of the derived clusters, and these entries indicate the pair-wise correlations in the clustered data. It is important to note that neglecting this information does not affect the shapes of the bands generated, yet the missing information can be revealed by another method allowing us to perform a refined cluster analysis including joint probabilities for the occurrence of iso-contours at different locations. We compute these probabilities by performing partial set operations in PCA-space, which take into account the orientation of the confidence ellipses. The derived information about the inherent dependencies between different locations

are finally used to indicate the clusters' sub-structures (Fig. 6.2 (d)).

6.4. Generation of Contour Variability Plots

In the following we assume that a PCA of the SDFs $\mathbf{d}_i$ has been computed (see Section 4.3.1), and that the full set of $r := n-1$ principal components (PCs) is used (the latter assumption will be relaxed later on). Thus, each $\mathbf{d}_i$ is represented exactly by a corresponding $\mathbf{c}_i \in \mathbb{R}^r$:

$$\mathbf{d}_i = R(\mathbf{c}_i) := \bar{\mathbf{d}} + U\mathbf{c}_i \quad (i = 1, ..., n). \tag{6.1}$$

Here, the vector $\bar{\mathbf{d}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{d}_i$ denotes the mean SDF, and the columns of $U \in \mathbb{R}^{M\times r}$ are the PCs, which form an orthonormal basis of a subspace of the SDF-space. Let us also introduce the function $R(\mathbf{c}_i) : \mathbb{R}^r \to \mathbb{R}^M$ to denote the corresponding reconstruction operation from PCA-space to SDF-space. R can be applied to arbitrary points $\mathbf{c} \in \mathbb{R}^r$, but then the resulting reconstructions $R(\mathbf{c})$ are, in general, not valid distance functions. However, they have similar properties like smoothness and monotonicity, and their zero-contours can be interpreted as artificial iso-contours.

In addition to the PCA, we assume that the $\mathbf{c}_i$ have been clustered, and for each cluster k an approximating MVN distribution has been computed. Each MVN distribution is parameterized by a covariance matrix $\Sigma_k \in \mathbb{R}^{r\times r}$ and mean value $\mu_k \in \mathbb{R}^r$. Since the number of points in a cluster is relatively small compared to the dimension r, usually the covariance matrix does not have full rank. However, we formally assume that Σ_k^{-1} exists, which does not pose a problem because it is not required in our final computation scheme.

To compute the standard deviation of a set of iso-contours, we represent the MVN distribution of each cluster in PCA-space using a single confidence ellipse. From this ellipse, the band representing the standard deviation of the corresponding set of iso-contours can be computed analytically, not requiring an exhaustive sampling and line rasterization process as for streamline variability plots.

Band Computation

The boundary $\mathcal{E}$ of a confidence ellipse for any MVN distribution $\mathcal{N}(\mathbf{x}, \mu, \Sigma)$ with covariance matrix Σ and mean μ to a given number of standard deviations α can be defined as the set of points $\mathbf{x}$ with Mahalanobis distance α (see Section 4.3.4):

$$\mathcal{E}(\Sigma, \mu, \alpha) = \left\{\mathbf{x} \;\middle|\; (\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu) = \alpha^2\right\}.$$

If we now consider the boundary of the confidence ellipse $\mathcal{E}_k := \mathcal{E}(\Sigma_k, \mu_k, \alpha) \subset \mathbb{R}^r$ representing a cluster k in PCA-space, a set of SDFs $R(\mathcal{E}_k) \subset \mathbb{R}^M$ are obtained when this ellipse is transformed to

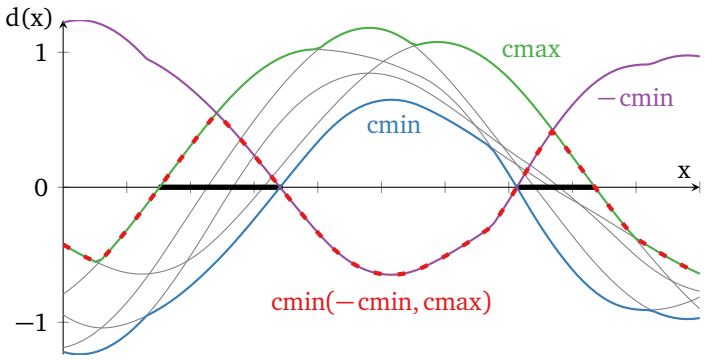


Figure 6.4.: Illustration of $L(\mathcal{D})$ (Eq. (6.2)). For a set of one-dimensional SDF-like functions $\mathcal{D} = \{d_1, ..., d_5\}$ (gray lines), the "band" covered by their zero-contours (thick, black lines) is the region where $L(\mathcal{D}) = \text{cmin}\{-\text{cmin}_i\, d_i,\ \text{cmax}_i\, d_i\} \geq 0$.

SDF-space. The corresponding band is defined as the set of points in domain space which is covered by at least one zero-contour of all SDFs in $R(\mathcal{E}_k)$.

In general, the extraction of a band from a (finite or infinite) set of distance functions $\mathcal{D} \subset \mathbb{R}^M$ can be realized analytically by determining a scalar function $L(\mathcal{D}) \in \mathbb{R}^M$ as

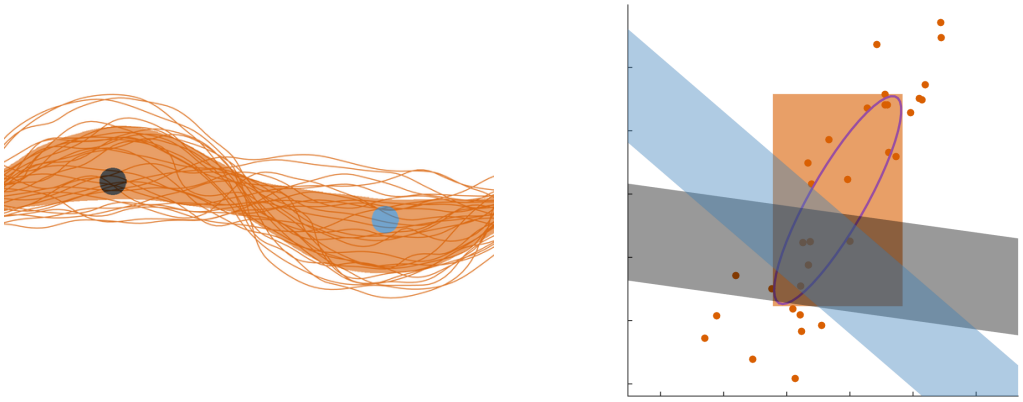
$$L(\mathcal{D}) = \text{cmin}\left\{-\underset{\mathbf{d}\in\mathcal{D}}{\text{cmin}}\{\mathbf{d}\},\ \underset{\mathbf{d}\in\mathcal{D}}{\text{cmax}}\{\mathbf{d}\}\right\}, \tag{6.2}$$

where cmin and cmax denote component-wise minimum and maximum operators. By construction, $L(\mathcal{D})$ is positive inside the band, negative outside the band, and its zero-contour corresponds to the band's boundary. This is illustrated in Fig. 6.4 for a 1D example.

To evaluate Eq. (6.2) for a set $\mathcal{D}$ of SDFs we require the minimum and maximum values in each of the M dimensions ($\hat{=}$ at each grid point). This means that each band function $L(\mathcal{D})$ is fully defined by the axis-aligned bounding box of $\mathcal{D}$ in SDF-space and, hence, a band in domain space corresponds to an axis-aligned box in SDF-space (see Fig. 6.5).

Because R is an orthonormal transformation, $R(\mathcal{E}_k)$ in SDF-space is also a confidence ellipse. Formally, this ellipse can also be determined directly in SDF-space by fitting a MVN distribution with $\Sigma_k^M \in \mathbb{R}^{M\times M}$ and $\mu_k^M \in \mathbb{R}^M$ to the SDFs $\mathbf{d_i}$ of cluster k. This gives the M-dimensional confidence ellipse $\mathcal{E}_k^M := \mathcal{E}(\Sigma_k^M, \mu_k^M, \alpha)$, which is equal to $R(\mathcal{E}_k)$. Consequently, the corresponding band functions $L(R(\mathcal{E}_k))$ and $L(\mathcal{E}_k^M)$ are equal, too.

In practice, on the other hand, the calculation of Σ_k^M is not feasible due to its large size. However, as shown before, $L(\mathcal{E}_k^M)$ is fully defined by the axis-aligned bounding box of $\mathcal{E}_k^M$ in $\mathbb{R}^M$ and, in turn, the bounding box of a confidence ellipse is fully defined by its mean and the diagonal of its covariance



Domain space SDF-space $\mathbb{R}^M$

Figure 6.5.: Domain and SDF space: Each contour in an ensemble (orange lines) corresponds to a point in SDF-space (orange points). A band of standard deviation corresponds to an axis-aligned rectangular region in SDF-space (orange regions), which is determined by the confidence ellipse that is fitted to the points in SDF-space (purple line). The contours going through the black and blue circle, respectively, form slabs in SDF-space. From their intersection, a joint occurrence in the two circles can be computed.

matrix. Hence, for the computation of $L(\mathcal{E}_k^M) = L(R(\mathcal{E}_k))$, we can simplify Eq. (6.2) to

$$L\left(\mathcal{E}_k^M\right) = \text{cmin}\left\{-\left(\mu_k^M - \alpha\sqrt{\text{diag}(\Sigma_k^M)}\right), \mu_k^M + \alpha\sqrt{\text{diag}(\Sigma_k^M)}\right\}. \tag{6.3}$$

This means that instead of transforming a confidence ellipse from PCA-space to domain space, we can construct a band locally (directly from all $\mathbf{d}_i$ in the corresponding cluster). The band is the region enclosed by the zero-contours of two fields which are computed point-wise as "$mean \pm \alpha \cdot standard\ deviation$".

6.5. Analysis of Global Correlations

In the previous section we have shown the construction of the confidence bands in domain space from axis aligned bounding boxes in SDF-space. We now demonstrate that by considering in addition the orientation and extent of the confidence ellipses in PCA-space, we can derive information concerning the probabilities of joint occurrences of iso-contours. Underlying this insight is the fact that the ellipses encode global correlations between the different SDFs $\mathbf{d}_i$, giving rise to our intended contour analysis.

By using the MVN distribution of a single cluster, we investigate the event of iso-contours going through a given location in the domain. Therefore, let us assume that $\mathbf{y}$ is the position of a grid point with index $j \in \{1, ..., M\}$ in the 2D or 3D domain. Then, all SDFs $\mathbf{d} \in \mathbb{R}^M$ corresponding to contours

through $\mathbf{y}$ have to be zero at that grid point j, i.e., $(\mathbf{d})_j = 0$. If $\mathbf{y}$ does not fall exactly onto a grid point, we can use linear interpolation. Let $\mathbf{w}_y \in [0;1]^M$ denote the linear interpolation weights for point $\mathbf{y}$, where $\mathbf{w}_y$ has at most four or eight non-zero entries in 2D or 3D, respectively, and the weights sum up to one. Then, all contours through $\mathbf{y}$ have to satisfy $\mathbf{d}^T\mathbf{w}_y = 0$, meaning that the set of contours through $\mathbf{y}$ corresponds to a hyper-plane in SDF-space (with normal $\mathbf{w}_y$).

To compute the probability of the occurrence of an event, we have to extend the hyper-plane to a volumetric region. This is because the probability that a contour is going exactly through a given point is zero. A first idea would be to vary $\mathbf{y}$, but this would change the normal of the corresponding hyper-plane. Instead, we use the fact that the SDFs store distances to the contours and consider the event of crossing a circular region rather than a point location.

Let $I(\mathbf{y}, s)$ denote the event of an iso-contour coming closer to the point $\mathbf{y}$ than a prescribed radius s. Then the following set of SDFs corresponds to $I(\mathbf{y}, s)$:

$$\{\mathbf{d} \in \mathbb{R}^M \;\big|\; |\mathbf{d}^T\mathbf{w}_y| \leq s\}$$

This is a subset of $\mathbb{R}^M$ that is enclosed between two parallel hyper-planes, and we will refer to such a subset as *slab* in the following. A slab can be "restricted" from SDF-space to PCA-space by substituting $\mathbf{d} = R(\mathbf{c}) = \bar{\mathbf{d}} + U\mathbf{c}$ according to Eq. (6.1), resulting in a r-dimensional slab denoted by $\mathcal{S}(\mathbf{y}, s)$:

$$\mathcal{S}(\mathbf{y}, s) = \{\mathbf{c} \in \mathbb{R}^r \;\big|\; |\mathbf{c}^T(U^T\mathbf{w}_y) + \bar{\mathbf{d}}^T\mathbf{w}_y| \leq s\}$$

For a set of contours, represented by an MVN distribution $\mathcal{N}(\mathbf{c}, \mu_k, \Sigma_k)$ of a cluster k in PCA-space, we can now calculate the probability of $I(\mathbf{y}, s)$ by integrating over the corresponding slab:

$$P\big(I(\mathbf{y}, s)\big) = \int_{\mathcal{S}(\mathbf{y},s)} \mathcal{N}(\mathbf{c}, \mu_k, \Sigma_k)\; d\mathbf{c}.$$

Furthermore, to investigate correlations between different locations in the domain, i.e., to determine probabilities of joint occurrence of contours at these locations, we can consider a second circular region at some other point location $\mathbf{z}$ with radius t. To calculate the joint probability of the event $I(\mathbf{y}, s) \wedge I(\mathbf{z}, t)$, which indicates that a contour comes within a radius s of $\mathbf{y}$ *and* within a radius t of $\mathbf{z}$, we have to integrate over the union of the two corresponding slabs (cf. Fig. 6.5):

$$P\big(I(\mathbf{y}, s) \wedge I(\mathbf{z}, t)\big) = \int_{\mathcal{S}(\mathbf{y},s)\cap\mathcal{S}(\mathbf{z},t)} \mathcal{N}(\mathbf{c}, \mu_k, \Sigma_k)\; d\mathbf{c} \tag{6.4}$$

This r-dimensional integral is difficult to compute directly, yet it can be transformed analytically into

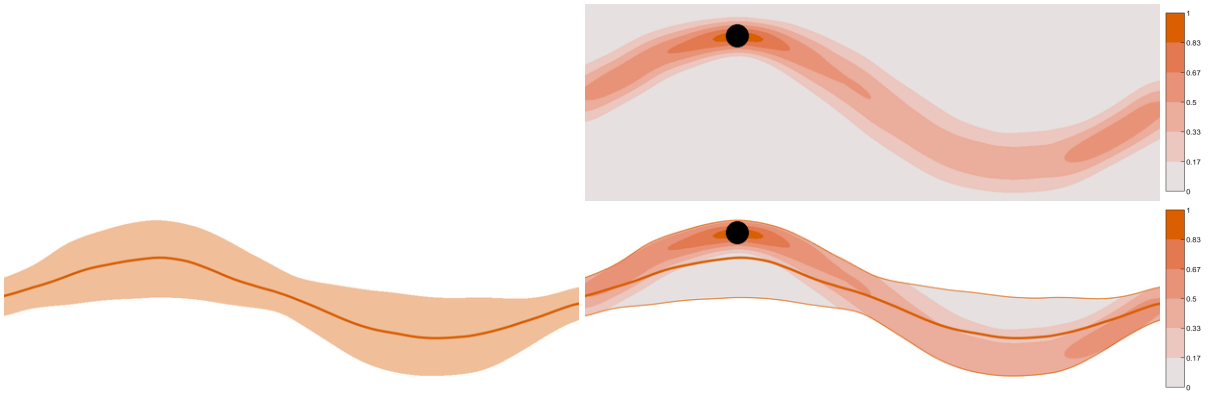


Figure 6.6.: Coloring of bands for the contour cluster in Fig. 6.5. Left: A unicolored variability plot with band and median. Right: Color coding of the probability function $f(\mathbf{z})$ at every domain point (top), and at points in the band (bottom).

the following integral over a rectangular region of a 2D MVN distribution:

$$\int_{\beta(\mathbf{w}_y)-s}^{\beta(\mathbf{w}_y)+s} \int_{\beta(\mathbf{w}_z)-t}^{\beta(\mathbf{w}_z)+t} \mathcal{N}\big((x_1,x_2)^T, \mathbf{0}, \Sigma_k'\big)\, dx_2\, dx_1$$
$$\text{with } \beta(\mathbf{w}) = -\mu_k^T(U^T\mathbf{w}) - \bar{\mathbf{d}}^T\mathbf{w} \tag{6.5}$$
$$\text{and } \Sigma_k' = \begin{pmatrix} \mathbf{w}_y^T U\Sigma_k U^T\mathbf{w}_y & \mathbf{w}_z^T U\Sigma_k U^T\mathbf{w}_y \\ \mathbf{w}_z^T U\Sigma_k U^T\mathbf{w}_y & \mathbf{w}_z^T U\Sigma_k U^T\mathbf{w}_z \end{pmatrix}.$$

For the sake clarity, we defer the proof of this statement to Section 6.7, yet what can be seen immediately is that a change of variables can be performed such that the normals of the two r-dimensional slabs are aligned with the first two coordinate axes. This allows collapsing all remaining dimensions of the MVN distribution, leaving a 2D problem. The computation of 2D rectangular normal probabilities like this is a standard problem in computational statistics and can be performed, e.g., using the method by Genz et al. [Gen04], which is included in a number of statistics libraries. To visualize the result of Eqs. (6.4+6.5), we choose locations $\mathbf{y}$ and $\mathbf{z}$ and prescribe the corresponding radii s and t. The user selects $\mathbf{y}$ and lets compute the joint probabilities for all other regions. For simplicity, we always use equal radii in this work, i.e., $s = t$. Additionally, to obtain probability values in the range $[0;1]$, we normalize the computed values with $P\big(I(\mathbf{y},t)\big)$. This yields the following function $f(\mathbf{z})$, which is defined over the entire domain and describes the conditional probability that contours come within radius t of $\mathbf{z}$, given that they also come within radius t of $\mathbf{y}$:

$$f(\mathbf{z}) = \frac{P\big(I(\mathbf{y},t) \wedge I(\mathbf{z},t)\big)}{P\big(I(\mathbf{y},t)\big)} \tag{6.6}$$

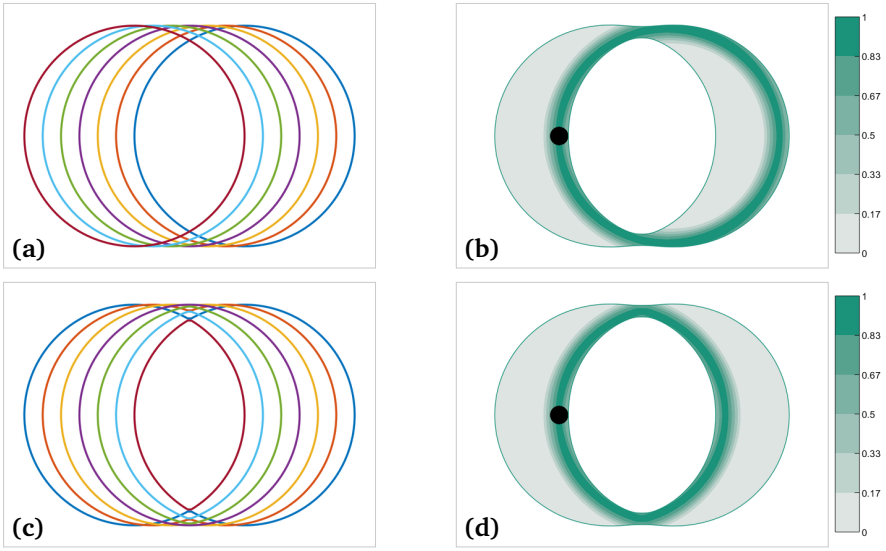


Figure 6.7.: Color coding of conditional probabilities. **(a)** Spaghetti plots comprised of shifted circles of the same size, and **(c)** the same circles cut off and mirrored at the center vertical axis. **(b+d)** The bands corresponding to one standard deviation $\alpha = 1$ (bounded by green lines). Only when coloring the bands according to $f(\mathbf{z})$ in a region around point $\mathbf{y}$ with radius t (black circle), the contours' structures can be revealed.

We optionally use $f(\mathbf{z})$ to color-code the bands in our visualizations based on a user defined circle $(\mathbf{y}, t)$, which is shown in Fig. 6.6. In this way, we can highlight regions which are likely to contain contours that run through the selected region and, therefore, we can reveal correlations between different regions in the domain which remain hidden when the bands are visualized in a single color. An example for such a scenario is shown in Fig. 6.7. Note that computing the normalization factor $P\big(I(\mathbf{y}, t)\big)$ requires an additional integration over a single slab. This can be achieved either by reducing the integration to a 1D problem (analogous to the reduction of Eq. (6.4) to 2D) or by simply making one of the slabs in Eqs. (6.4+6.5) infinitely large.

6.6. Plot Computation and Composition

In this section we give additional details concerning the PCA computation and the clustering process in PCA-space, which is mostly analogous to streamline variability plots. Furthermore, we describe how to compose and generate the final contour variability plots.

PCA: In practice, we do not use the full set of $(n-1)$ PCs, but we reduce the PCA-space to a dimension $r \leq n-1$. To determine r, we use the common *explained variance* criterion: Let σ_j denote the variance of the PCA coefficients along the j-th PC. Then the total amount of explained variance by r PCs is defined as: $\text{ex}(r) = \sum_{j=1}^{r} \sigma_j / \sum_{j=1}^{n-1} \sigma_j$. Based on a threshold $\tau \in [0,1]$, we choose the smallest r which satisfies $\text{ex}(r) \geq \tau$. Since the input iso-contours often contain many small details and the

dimensionality of the PCA-space only has a minor impact on the performance of our implementation, we usually only drop the very insignificant PCs. I.e., we use $\tau = 0.999$ in all our experiments in this chapter, which, for the meteorological ensembles, leads to roughly 35-45 required PCs out of 51.

Clustering: To separate the major trends in a contour ensemble, the resulting PCA coefficients $\mathbf{c}_1, ..., \mathbf{c}_n \in \mathbb{R}^r$ are clustered. As discussed in Section 5.5 for the clustering of streamlines, we do not fit multiple MVN distributions directly to the data using the commonly applied Expectation Maximization algorithm, because it leads to numerical problems due to the high dimensionality of the PCA-space and does not ensure a good spatial separation of the clusters in this space. Instead, we use agglomerative hierarchical clustering (AHC) with average linking, which is initialized with the Euclidean distances in PCA-space as distance measure. Furthermore, we perform an automated guess for the optimal number of clusters using the L-method [SC04] with an extension that is described in Section 4.3.3. This initial guess can by changed by the user, causing clusters to split and merge recursively according to the AHC hierarchy.

Median Contour: For each cluster of input contours, we draw an artificial median contour (which does not exist in the original set of contours) as a single representative for that cluster. We compute the *geometric median* $\hat{\mathbf{c}}_k$ of the corresponding points in PCA-space and transform it back to the initial domain by extracting the zero-contour of $R(\hat{\mathbf{c}}_k)$. We draw the median contour with a line width corresponding to the relative size of the cluster, and each plot is augmented with an additional **bar plot** to show these sizes exactly.

Bands: For each cluster, a band corresponding to a user-defined size α in units of standard deviation is drawn (we use $\alpha = 1$ in all our examples). Bands are computed according to Eq. (6.3) and visualized as filled, transparent polygons in 2D, and using iso-surface raycasting in 3D. Since thin bands can often not be represented by a single scalar field, we use two intermediate fields and delay the outermost $cmin$ operation in Eq. (6.3) until the values have been interpolated from these intermediate fields.

Outliers are detected by our hierarchical clustering as clusters of cardinality one. The contour in an outlier cluster is drawn directly and no band is shown (cf. Figs. 6.1, 6.9 (g+h), 6.10 (c+d)). To this end, we do not explicitly search for outliers.

Correlation Color Coding: The user can click on a lobe in order to pick a circle of interest (always shown in black), defining the circle $(\mathbf{y}, t)$. The picked lobe is then visualized on top of all others and color coded with $f(\mathbf{z})$ according to Eq. (6.6), using the same radius t for both involved circles. The values that are shown are therefore actual probability values (and not a probability density), indicating at every point $\mathbf{z}$ the conditional probability that a line through the black circle is also going through a circle of the same size centered at $\mathbf{z}$.

6.7. Integration of MVN distribution

In this section, we provide the proof for the identity between Eq. 6.4 and Eq. 6.5, which we have skipped in Section 6.5 for reasons of clarity. Let $\mathcal{N}(\mathbf{x}, \mu, \Sigma)$ denote the MVN density with mean $\mu \in \mathbb{R}^r$ and covariance matrix $\Sigma \in \mathbb{R}^{r\times r}$:

$$\mathcal{N}(\mathbf{x}, \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^n \det(\Sigma)}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)\Sigma^{-1}(\mathbf{x}-\mu)\right)$$

Then, we are interested in integrating $\mathcal{N}$ over an intersection of two slabs (= regions between two parallel hyper-planes):

$$\begin{aligned}\hat{\mathcal{S}}_a :=& \{\mathbf{x} \in \mathbb{R}^r \;\big|\; |\mathbf{a}^T\mathbf{x} + c| \leq s\} \\ \hat{\mathcal{S}}_b :=& \{\mathbf{x} \in \mathbb{R}^r \;\big|\; |\mathbf{b}^T\mathbf{x} + d| \leq t\}\end{aligned} \tag{6.7}$$

Here, we assume $\mathbf{a}, \mathbf{b} \in \mathbb{R}^r$ and $c, d, s, t \in \mathbb{R}$ to be fixed parameters. For simplicity reasons, $\hat{\mathcal{S}}$ is defined slightly different to $\mathcal{S}$ in Section 6.5 (we use $\mathbf{a} = U^T\mathbf{w}_y$, $\mathbf{b} = U^T\mathbf{w}_z$, $c = \bar{\mathbf{d}}^T\mathbf{w}_z$ and $d = \bar{\mathbf{d}}^T\mathbf{w}_z$ here).

For the integration, we will perform a change of variables which aligns $\mathbf{a}$ and $\mathbf{b}$ with the first and second coordinate axis, respectively, allowing us to collapse the other dimensions of the MVN distribution. Therefore, we define the affine transformation $q(\mathbf{y}) = T\mathbf{y} + \mu$, where the $r \times r$ matrix $T := (S^{-1})^T$ is given through $S := (\mathbf{a}, \mathbf{b}, \mathbf{t}_3, \ldots, \mathbf{t}_r)$. The columns $\mathbf{t}_i \in \mathbb{R}^r$ can be chosen arbitrarily, such that S has full rank and $\det(S) > 0$. Let $\mathbf{e}_i$ denote the i-th unit vector, then, by construction, the following holds:

$$S\mathbf{e}_1 = \mathbf{a} \wedge S\mathbf{e}_2 = \mathbf{b} \quad \Rightarrow \quad T^T\mathbf{a} = \mathbf{e}_1 \wedge T^T\mathbf{b} = \mathbf{e}_2 \tag{6.8}$$

To perform the change of variables, we also need to apply q^{-1} to the integration regions $\hat{\mathcal{S}}(\mathbf{a}, c)$ and $\hat{\mathcal{S}}(\mathbf{b}, d)$, which can be achieved by substituting $\mathbf{x} = T\mathbf{y} + \mu$ in Eq. (6.7):

$$\begin{aligned}\hat{\mathcal{S}}'_a :=& q^{-1}(\hat{\mathcal{S}}_a) = \{\mathbf{y} \in \mathbb{R}^r \;\big|\; |\mathbf{a}^T T\mathbf{y} + \mathbf{a}^T\mu + c| \leq s\} \\ \hat{\mathcal{S}}'_b :=& q^{-1}(\hat{\mathcal{S}}_b) = \{\mathbf{y} \in \mathbb{R}^r \;\big|\; |\mathbf{b}^T T\mathbf{y} + \mathbf{b}^T\mu + d| \leq t\}\end{aligned}$$

Using Eq. (6.8), we can simplify this to

$$\begin{aligned}\hat{\mathcal{S}}'_a =& \{\mathbf{y} \in \mathbb{R}^r \;\big|\; |(\mathbf{y})_1 + \mathbf{a}^T\mu + c| \leq s\} \\ \hat{\mathcal{S}}'_b =& \{\mathbf{y} \in \mathbb{R}^r \;\big|\; |(\mathbf{y})_2 + \mathbf{b}^T\mu + d| \leq t\}.\end{aligned} \tag{6.9}$$

Finally, we can tackle the integral which we are interested in:

$$\begin{aligned}
&\int_{\hat{\mathcal{S}}_a \cap \hat{\mathcal{S}}_b} \mathcal{N}(\mathbf{x}, \mu, \Sigma) d\mathbf{x} \\
=&\frac{1}{\sqrt{(2\pi)^n \det(\Sigma)}} \int_{\hat{\mathcal{S}}_a \cap \hat{\mathcal{S}}_b} \exp\left(-\tfrac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1} (\mathbf{x}-\mu)\right) d\mathbf{x}
\end{aligned}$$

Using the previously defined transformation $q(\mathbf{y})$, we substitute $\mathbf{x} = T\mathbf{y} + \mu$:

$$\begin{aligned}
=&\frac{\det(T)}{\sqrt{(2\pi)^n \det(\Sigma)}} \int_{\hat{\mathcal{S}}'_a \cap \hat{\mathcal{S}}'_b} \exp\left(-\tfrac{1}{2}(T\mathbf{y})^T \Sigma^{-1} (T\mathbf{y})\right)\, d\mathbf{y} \\
=&\frac{1}{\sqrt{(2\pi)^n \det(T^{-1}\Sigma(T^T)^{-1})}} \\
&\qquad \cdot \int_{\hat{\mathcal{S}}'_a \cap \hat{\mathcal{S}}'_b} \exp\left(-\tfrac{1}{2}\mathbf{y}^T (T^T \Sigma^{-1} T)\mathbf{y}\right) d\mathbf{y} \\
=&\int_{\hat{\mathcal{S}}'_a \cap \hat{\mathcal{S}}'_b} \mathcal{N}\left(\mathbf{y}, 0, T^{-1}\Sigma(T^T)^{-1}\right)\, d\mathbf{y} \\
=&\int_{\hat{\mathcal{S}}'_a \cap \hat{\mathcal{S}}'_b} \mathcal{N}(\mathbf{y}, 0, S^T \Sigma S)\, d\mathbf{y}
\end{aligned}$$

The integration region is now bounded by hyper-planes which are orthogonal either to the first or to the second coordinate axis (see Eq. (6.9)). Hence, regardless of the original dimensionality r, we can collapse all remaining dimensions of the MVN distribution, which gives the two-dimensional integral

$$= \int_{-s-\mathbf{a}^T\mu-c}^{s-\mathbf{a}^T\mu-c} \int_{-t-\mathbf{b}^T\mu-d}^{t-\mathbf{b}^T\mu-d} \mathcal{N}\left((y_1, y_2)^T, 0, \Sigma'\right) dy_2\, dy_1,$$

with integrations bounds according to Eq. (6.9) and with $\Sigma' \in \mathbb{R}^{2\times 2}$ being the upper left 2×2 submatrix of $S^T \Sigma S$:

$$\Sigma' = \begin{pmatrix} \mathbf{a}^T\Sigma\mathbf{a} & \mathbf{a}^T\Sigma\mathbf{b} \\ \mathbf{a}^T\Sigma\mathbf{b} & \mathbf{b}^T\Sigma\mathbf{b} \end{pmatrix}.$$

This is the expression we were looking for. Additionally, we can transform it into a normalized ρ-form by rescaling along both coordinate axes. This is the format that is typically accepted by libraries which

can calculate 2D rectangular normal probabilities:

$$\int_{\hat{\mathcal{S}}_a \cap \hat{\mathcal{S}}_b} \mathcal{N}(\mathbf{x}, \mu, \Sigma) d\mathbf{x}$$
$$= \int_{\frac{-s-\mathbf{a}^T\mu-c}{\mathbf{a}^T\Sigma\mathbf{a}}}^{\frac{s-\mathbf{a}^T\mu-c}{\mathbf{a}^T\Sigma\mathbf{a}}} \int_{\frac{-t-\mathbf{b}^T\mu-d}{\mathbf{b}^T\Sigma\mathbf{b}}}^{\frac{t-\mathbf{b}^T\mu-d}{\mathbf{b}^T\Sigma\mathbf{b}}} \mathcal{N}\big((y_1, y_2)^T, 0, \Sigma''\big)\, dy_2\, dy_1,$$

$$\Sigma'' = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \quad \text{with} \quad \rho = \frac{\mathbf{a}^T\Sigma\mathbf{b}}{\sqrt{(\mathbf{a}^T\Sigma\mathbf{a}) \cdot (\mathbf{b}^T\Sigma\mathbf{b})}}.$$

6.8. Results

This section presents meteorological examples of 2D and 3D contour variability plots and of 2D correlation visualizations, using weather forecast data from the European Centre for Medium-Range Weather Forecasts (ECMWF) Ensemble Prediction System (ENS). The forecasts include 50 perturbed members and a control run. For details, we refer to e.g. [LP08]. For the present study, we use the forecast from 00:00 UTC 15 October 2012 that has been used previously by Rautenhaus et al. [RKSW15]. The example region encompasses 101 × 41 × 62 grid points. For the 2D examples, the data were interpolated to vertical levels of constant pressure.

For 2D interactive demonstrations, our implementation uses the MATLAB implementations of PCA and AHC. The publicly available FORTRAN implementation of the method by Genz et al. [Gen04] was used for the calculation of the 2D rectangular, normal probabilities. A custom implementation was used for computing signed distance transforms in which, firstly, we compute the unsigned distance field to an iso-contour of a given scalar field using fast marching [Set99] and, secondly, determine the sign of the resulting distance value at every grid point by comparing its scalar value against the specified iso-value. Since the conditional probabilities $f(\mathbf{z})$ typically are smooth, we compute them in a texture using half the grid spacing of the original grid.

All results presented in this chapter were generated on a standard desktop PC (Intel Xeon X5675 processor with 6 × 3.0 GHz, 8 GB RAM and an NVIDIA Geforce GTX 680). On this machine, the generation of a contour variability plot for an ensemble of 2D forecast fields (101 × 41 grid) takes on average 270 ms. The timing includes 66 ms for the distance transformation, 31 ms for the PCA of the distance functions, and 31 ms for hierarchical clustering. The time increases by 80 ms if a correlation visualization for a single cluster is computed. For the 3D example, the contour variability plot (i.e., the SDFs for median contours and the outer boundaries of the "bands") was precomputed and rendered via iso-surface raycasting in the open-source visualization tool Met.3D [RKSW15]. The

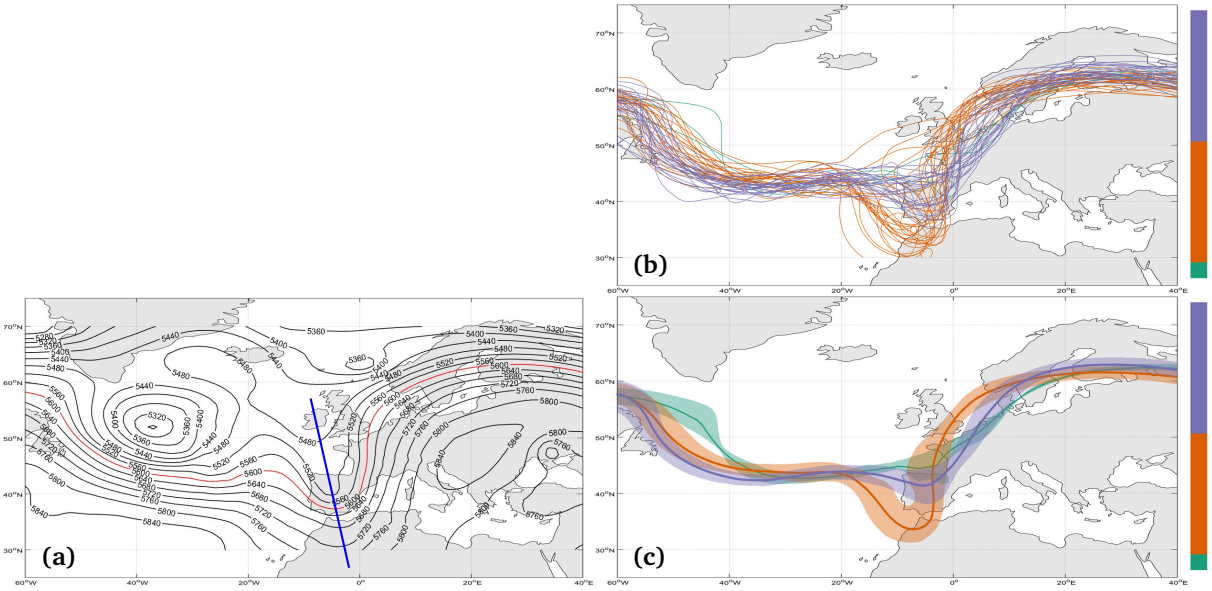


Figure 6.8.: Spatial variability of an ensemble of 500 hPa geopotential height contour lines. The ECMWF ENS forecast from 00:00 UTC, 15 October 2012, valid at 00:00 UTC, 20 October 2012, is shown. **(a)** Geopotential height contours (m) of the ensemble control forecast (5600 m highlighted in red). Note the distinct trough over Spain (blue axis). **(b)** Spaghetti plot of the 5600 m contour lines of all 51 ensemble members, colored by cluster membership. **(c)** Contour variability plot.

precomputation required 11.8s, including 8.1s for the distance transform, 2.6s for the PCA and 40ms for hierarchical clustering.

Contour Variability Plots

To demonstrate the proposed contour variability plots, we discuss two meteorological examples. Fig. 6.8 shows 2D contours of the forecast geopotential height field at 500 hPa. Spaghetti plots of such fields are frequently encountered in meteorology (e.g., [Wil11]). The geopotential height field of the control forecast (Fig. 6.8 (a)) shows a distinct trough extending from Ireland over Spain (blue line). We focus on the 5600 m contour. The spaghetti plot of all members' 5600 m contours (Fig. 6.8 (b)) reveals variability in the line ensemble, in particular in the trough region. Without cluster coloring, it is impossible to discern differences and trends (not shown). The coloring in Fig. 6.8 (b) increases the information content; the variability plot in Fig. 6.8 (c) clearly shows the possible scenarios. The purple and orange clusters contain the majority of the members. The bands of both clusters are largely similar, except for the region around and east of the trough. The lines in the orange cluster represent a much stronger trough. In contrast, contours in the third cluster (green) show notable differences both south of Greenland and in the trough region. We note that the clustering algorithm takes into account the entire domain. If the user's interest is centered on a particular feature in a particular

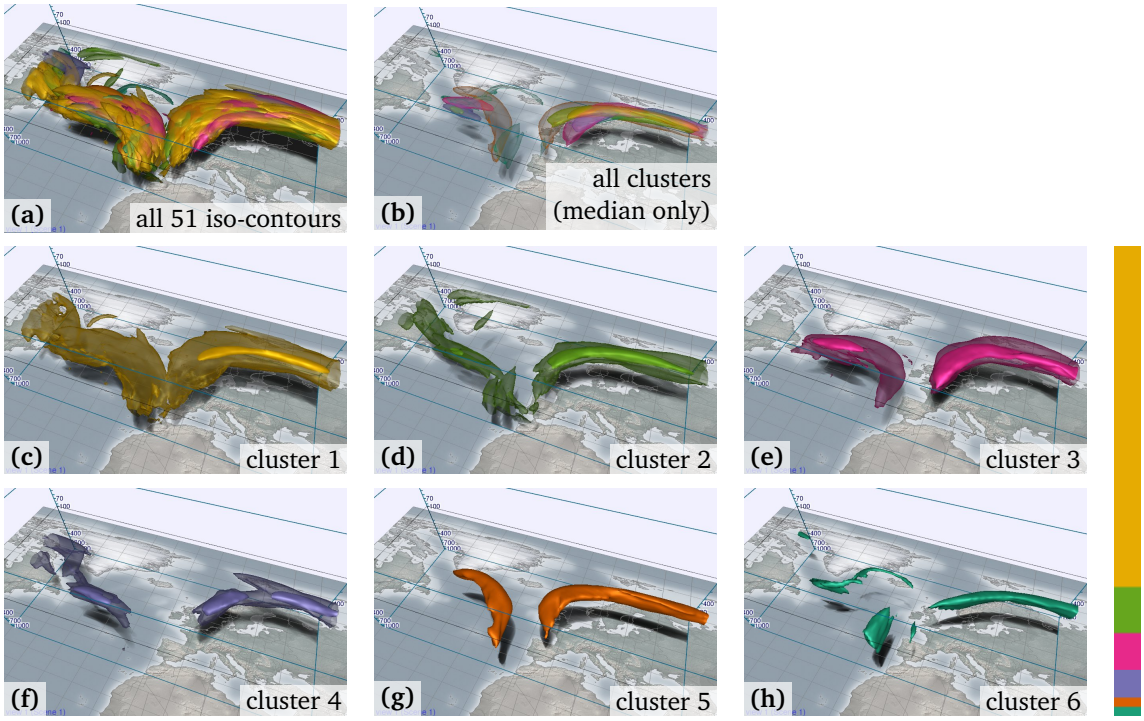


Figure 6.9.: Contour variability plot for an ensemble of 3D wind speed contours representing the jet stream. 50 ms^{-1} iso-surfaces of the ECMWF ENS forecast from 00:00 UTC, 15 October 2012, valid at 18:00 UTC, 19 October 2012 are shown. For comparison, the same scene as in Fig. 6 in [RKSW15] has been chosen. The images show **(a)** a "3D spaghetti" plot of all 51 iso-surfaces (colored by cluster membership), **(b)** the median contours of six clusters identified by the algorithm, and **(c-h)** the median contour (opaque) and the outer boundary of the "band" (transparent) for the six clusters. Note that clusters 5 and 6 contain one outlier each.

region, it may be useful to let the user select a subregion.

In Fig. 6.9, 3D contour clustering is applied to an example from the work by Rautenhaus et al. [RKSW15]. 50 ms^{-1} wind speed iso-surfaces are used as representative features of the jet stream, strong winds (with core wind speeds often exceeding 50 ms^{-1}) within narrow bands typically encountered around 10 km height (e.g., [Ahr08]). Rautenhaus et al. [RKSW15] (their Fig. 6) show iso-surfaces of selected single members and of ensemble statistical quantities, rendered by the Met.3D system. To discern different scenarios of jet feature characteristics, the user of Met.3D can animate over the ensemble members, i.e., cycle through the display of single members. Our clustering method largely enhances this simple approach. Fig. 6.9 (a) shows the iso-surfaces of all members in a "3D spaghetti" plot. In 3D, differences between the members are even more difficult to discern than in the 2D case; there is little more information in the plot that in the plot of the ensemble maximum in Fig. 6f of [RKSW15]. In 3D, plots of all cluster medians and their bands (as in Fig. 6.8 (c)) also suffer from cluttering. We hence visualize a composite of the cluster medians only to show the major scenarios of the jet stream feature (Fig. 6.9 (b)), as well as individual medians and bands of six clusters that

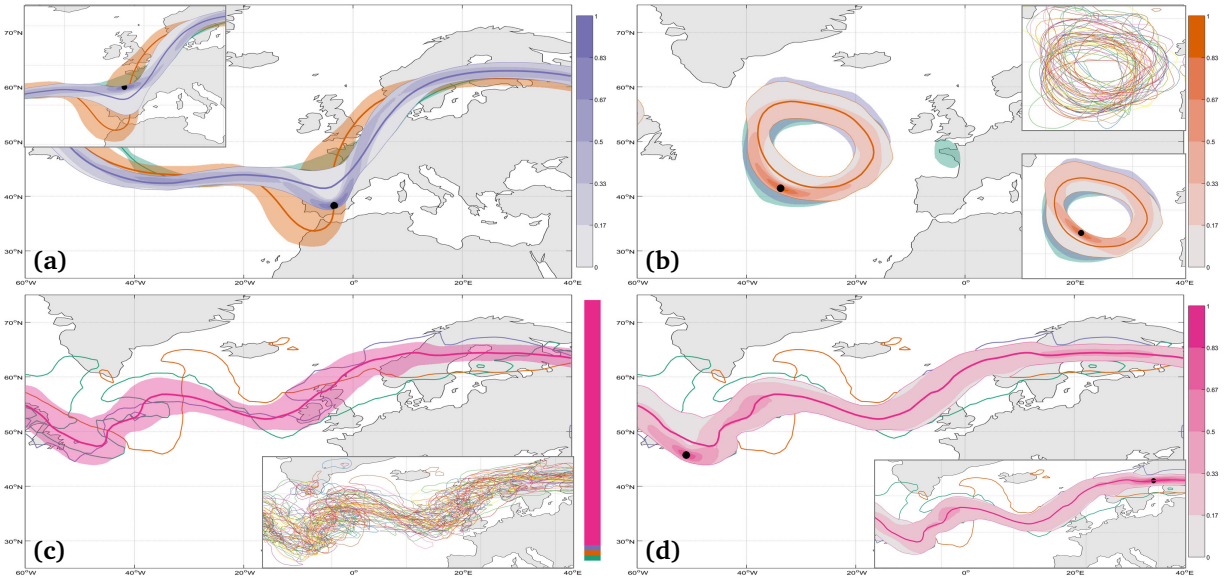


Figure 6.10.: Spatial correlations. **(a)** Correlations in the purple cluster of Fig. 6.8 (c). **(b)** Correlations of a cluster of 680 m contour lines of geopotential height at 925 hPa show a behavior similar to the idealized case in Fig. 6.7 (b) (forecast from 00:00 UTC, 17 October 2012, valid at 12:00 UTC, 21 October 2012). **(c)** Contour lines of 0°C of the temperature field at 850 hPa are clustered into a single cluster and three outliers (same forecast as in Fig. 6.8). **(d)** Except for the westernmost part of the analyzed region, these lines are almost entirely uncorrelated.

our method has detected (ordered from the largest to the smallest cluster in Fig. 6.9 (c-h)). While with this approach, the user still needs to animate over the clusters to get information on possible jet characteristics, the number of animation steps is reduced from 51 members to 6 clusters.

Correlation Visualization

Examples of plots with the proposed visualization of global correlations are depicted in Fig. 6.10. Fig. 6.10 (a) shows correlations of iso-contours passing through a user-positioned circle within the purple cluster of the example in Fig. 6.8 (c). Over western Europe, strong correlations are visible; iso-contours that describe a more pronounced trough (circle positioned over southern Spain) tend to proceed on the western edge of the cluster band over the North Sea (i.e., deeper troughs are more narrow). In other regions, however, the position of the lines is largely uncorrelated to the strength of the trough. Closer to the surface (925 hPa and 36 hours later; Fig. 6.10 (b)), the low-pressure system visible south of Greenland in Fig. 6.8 (a) can be well identified in the geopotential height field. Here, contour lines in the selected cluster resemble a situation similar to the one in Fig. 6.7 (a+b). We conclude that the system has similar size in the individual ensemble members within the cluster, but is shifted in space.

Our third example (Fig. 6.10 (c+d)) shows 0 °C temperature iso-contours at 850 hPa. Here, the contour lines contain many high frequency details; in the spaghetti plot, structures are impossible to discern (Fig. 6.10 (c)). Our clustering method yields one large cluster and three outliers. Positioning the circle in the cluster band south of Greenland (Fig. 6.10 (d)) reveals correlations in the westernmost part of the region, where the low pressure system pushes cold air from the northwest towards the southeast. Hence, if air below 0 °C has been pushed as far south as the circle location at 50 °W, the cold air has also been pushed to the southern side of the variability band further west. East of the circle, however, the contour lines passing through the circle are distributed almost uniformly within the band, indicating that the deviations of the contours from the cluster median are of random nature and not globally correlated. In this case, we cannot make any conclusion, e.g., about the 0 °C boundary being located further south over Great Britain if it is located further south at 50 °W. The uniform distribution is confirmed by placing the circle at different locations (one of which is shown in the inset).

6.9. Conclusion

In this chapter we have shown that the use of distance functions enables us to generate contour variability plots which allow for a characterization of the uncertainty that is represented by an ensemble of iso-contours. In contrast to spaghetti plots, the visual abstraction that is provided by our method effectively conveys the major trends and outliers in a given set of iso-contours. While our method is naturally limited in 3D due to overlaps and occlusion problems, our clustering and visual abstraction nevertheless provide an improvement over having to animate through single ensemble members. In addition, the proposed color coding of confidence bands can assist a user in detecting global correlations between occurrences of contours at different locations. It enables the interactive exploration of interior structures in clusters. Thus, it can help to improve an initial clustering and to decide whether a local or a global analysis of a set of contours is appropriate.

7

Time-hierarchical Clustering and Visualization of Weather Forecast Ensembles

In this chapter, we present the third application of variability plots by investigating the visualization of iso-contours in weather forecast ensembles which are started from perturbed but similar initial conditions. We specifically target a common use case, in which the variability in the ensemble during—and leading up to—a short time window of interest has to be analyzed. For this scenario, we can build upon the ideas of contour variability plots as introduced in the previous chapter, but we have to take into account that the underlying data is now time-dependent. If we were to create individual contour variability plots for each timestep, the corresponding clusterings of consecutive timesteps would often be significantly different and temporal relationships would be hard to identify. In order to solve this problem, we propose a special *time-hierarchical clustering*, which is in accordance with the (assumed) slowly diverging nature of the ensemble. We first perform an initial clustering of the ensemble members in the time window of interest, and then merge clusters in time-reversed order at the points where they start to diverge. This ensures a temporal consistency between the clusterings of consecutive timesteps, and thus allows us to meaningfully combine visualizations of multiple timesteps. On the one hand, we do this by generating space-time surfaces of single cluster medians, and, on the other hand, we create stacked visualizations of contour variability plots. We demonstrate the effectiveness of the resulting visual encodings with forecast examples of the European Centre for Medium-Range Weather Forecasts, for which we are able to convey—in a single view—the development of trends and variability in the ensemble towards the selected time window of interest.

This chapter is based on material that has been originally published in F. FERSTL, M. KANZLER, M. RAUTENHAUS, R. WESTERMANN: Time-hierarchical Clustering and Visualization of Weather Forecast Ensembles. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization)*, 2017, *to appear*. ©2017 IEEE.

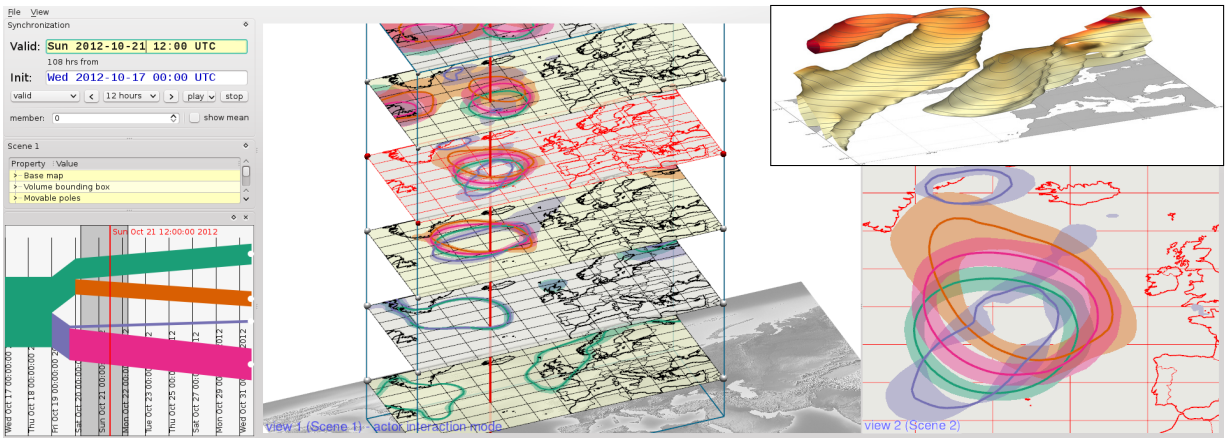


Figure 7.1.: Interactive exploration of time-varying iso-contours in a weather forecast ensemble: Based on our time-hierarchical clustering (bottom left) the user can animate through visualizations of selected time steps (bottom right). An overview of all time steps is shown as a stacked plot (middle), and a representative space-time surface can be shown for a selected cluster (inset, top right).

7.1. Introduction

Ensemble weather forecasts have been well established in meteorology as a tool to provide estimates of the uncertainty inherent in numerical weather predictions [BTB15]. Based on perturbed initial conditions or different forecast model formulations, ensemble methods provide a representative sample of possible future states of the atmosphere [LP08]. Analysis of the temporal evolution and variability of an ensemble forecast is an important task. For example, a forecaster needs to judge whether a prediction can be trusted in a particular region and forecast time window. Also, information on possible future atmospheric scenarios the ensemble predicts and their likelihood are of interest [FC11]. Furthermore, spatio-temporal analysis of an ensemble forecast can provide information on where and when ensemble members start to diverge, allowing inference on atmospheric phenomena that cause members to evolve in different ways.

Spaghetti plots have been established as a common visualization tool to analyze the variability of iso-contours in meteorological scalar field ensembles (e.g., [Wil11]). Typically, these plots simultaneously show contours of all ensemble members at a single timestep. For analysis of the temporal evolution and variability of the displayed scalar fields, animation of sequences of spaghetti plots is frequently used (see, e.g., the website of the National Oceanic and Atmospheric Administration's Storm Prediction Center[1]). When viewing such animations, however, connections between contours in consecutive timesteps are difficult to establish, due to the huge amount of visual information that needs to be memorized by the viewer. As pointed out by Gleicher et al. [GAW*11, p. 294], animation "requires

[1] www.spc.noaa.gov/exper/sref/

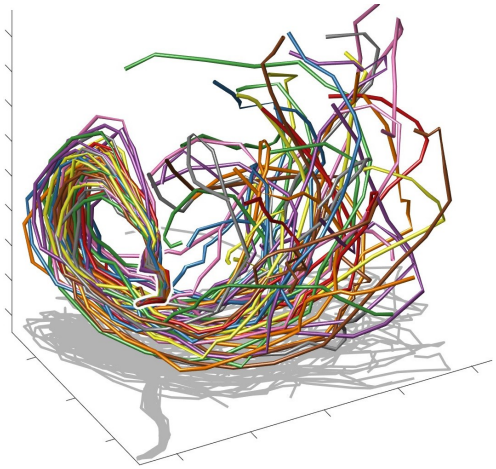


Figure 7.2.: Multi-run plot [FML16] of an ensemble of 51 iso-contours to a fixed iso-value in a geopotential height field, over the first 8 days of a weather forecast in timesteps of 6 hours. Each line represents the evolution trajectory of one ensemble member. Each line vertex corresponds to an iso-contour in a specific timestep. The data is projected to a 3D space using a multi-dimensional scaling projection, so that large vertex-to-vertex distances correspond to large dissimilarities between the corresponding contours.

the use of the viewer's memory and attention shifts to make connections between objects", making it problematic in general for spatio-temporal analysis. In addition to variability plots, a number of other visual abstractions of spaghetti plots have been proposed in recent years [SZD*10, WMK13]. All these techniques, however, represent single timesteps and rely on animation to visualize the temporal development.

In this chapter, we develop an alternative approach to analyze the spatio-temporal evolution of iso-contours obtained from ensemble weather predictions. We target a setting in which the ensemble members are started from perturbed but (relatively) similar initial conditions. This is true, e.g., for the operational ensemble forecast produced by the European Centre for Medium-Range Weather Forecasts (ECMWF) Ensemble Prediction System (ENS; e.g., [LP08]). Due to the chaotic nature of the atmosphere and the techniques used to select initial ensemble members, the variability, or spread, of such forecast ensembles on average increases over time [LP08]. As demonstrated in Fig. 7.2, this diverging nature can be visualized with "multi-run plots" as proposed by Fofonov et al. [FML16]. They compute a similarity matrix of all contours and visualize their trajectories using a multidimensional scaling projection. Note, however, that besides restricting the visualization to the three most significant axis in the multidimensional feature domain, the method cannot show where in the spatial domain the spread is large or low.

To analyze the spatio-temporal evolution of such ensemble predictions, we target the following objectives:

- Temporal evolution similarity: Group ensemble members with similar temporal evolution in a

specified region and time window (following operational but static practice at ECMWF [FC11], we assume that the user is interested in grouping ensemble members according to similar forecast scenarios in a user-specified region and time windows, and in analyzing the temporal development of the obtained groups prior to and after the selected window).

- Time-hierarchical variability: Develop approaches that show the hierarchical temporal evolution of such groups of members and that help to determine the spatial and temporal locations at which differences between the ensemble members start to occur.
- Space-time visualization: Develop visual encodings to analyze the temporal evolution of spatial variability in an effective way.

Contribution To achieve the aforementioned goals, we propose new approaches for clustering weather forecast ensembles with respect to their temporal evolution, and for visualizing the resulting cluster hierarchies in a spatio-temporal context. Our specific contributions are with respect to the following aspects:

- Clustering of iso-contours with similar time evolution: We consider sets of sequences of iso-contours and cluster these sets to find iso-contours with similar dynamics. Each sequence is comprised of iso-contours in a given ensemble member over an arbitrary given time window.
- Time-hierarchical clustering of iso-contours: By taking into account the characteristic behavior of weather forecast ensembles, we propose merging of clusters from a specified time window in the forecast range in time-reversed order. This creates a cluster tree, with the root node being the initial conditions from which the temporal integration of the ensemble simulations has started.
- Space-time cluster visualization: We develop visualization techniques to assess in a single view where variations in the predicted atmospheric states occur and when they start occurring.

We believe that our findings can significantly help to improve and accelerate the analysis of ensembles of iso-contours extracted from meteorological simulation output starting from perturbed but similar initial conditions. In particular, we demonstrate that our approach has significant advantages over the animation of spaghetti plots, since it can reveal information which is difficult to grasp from such plots. We demonstrate the effectiveness of our approach with real-world forecast examples obtained from ECMWF.

7.2. Related Work

Ensemble data is typically spatio-temporal, multivariate, and multivalued [KH13, LPK05], making the analysis and visualization processes difficult. Typical methods to simplify the data evaluate summary statistics and visualize these using color maps, contours, surface deformation, opacity, boxplots, or glyphs [LPK05, PW13, PKRJ10].

For vector fields, Wittenbrink et al. [WPL96] propose uncertainty vector glyphs to show the magnitude and angular uncertainty. For time-varying uncertain vector fields, Hlawatsch et al. [HLNW11] introduce flow radar glyphs. To consider the transport uncertainty, Otto et al. [OGHT10] use particle density functions to obtain an uncertain topological segmentation for Gaussian-distributed steady vector fields and Hummel et al. [HOGJ13] compare the material transport in time-varying flow ensembles by computing individual and joint vector field variances.

Other visualization techniques for dealing with the complexity of ensemble data include coordinated multiple views, which are commonly used to study multivariate relations via linking and brushing [KH13] and parallel coordinates [HW13]. Nocke et al. [NFB07] use coordinated multiple views for climate ensembles. Piringer et al. [PPBT12] propose a design on three levels of details for a comparative visual analysis of 2D function ensembles.

Standard approaches for dealing with large and complex data are to restrict to subsets of the data using, e.g., feature based querying [KHP12], or to use clustering [Jai10]. The clustering of iso-contours in 2D is related to the clustering of parametrized curves. This can be achieved using a variety of different geometric distance measures, e.g., as compared by Zhang et al. [ZHT06] for 2D trajectories in surveillance videos, or by Oeltze et al. [OLK*14] for 3D streamlines of blood flow simulations. As shown in the previous chapter, the clustering of iso-contours based on geometric representations is more involved than the clustering of curves, because iso-contours do not have a natural parametrization and are often composed of multiple, fragmented parts. Several alternative representations have been used to analyze iso-contours like, e.g., rotation and scale invariant features [TN14] or histograms and iso-surface statistics [CBB06]. In particular, signed distance functions are a popular choice for representing iso-contours because each contour can be directly mapped to a point in a high-dimensional Euclidean space by sampling the corresponding signed distance function on a fixed grid. Bruckner and Möller [BM10] use signed distance functions to analyze different iso-contours of the same scalar field, Rathi et al. [RDT06] use such functions for shape analysis.

The clustering of time series data has been shown to be effective in several domains like computer vision, data mining and visualization. Previous approaches have focused mainly on the clustering of uni- or multivariate time series data, i.e., sequences of scalar values which can be interpreted as vectors and clustered using, for instance, Euclidean distances [PG00] or root-mean-square distances [WS99]. Alternative approaches perform clustering based on derived features instead of the

raw data [GTR*99]. For a thorough overview of approaches for the clustering of time series let us refer to the articles by Denton [Den05] and Liao [Lia05]. While several works consider the clustering of time-series of data points which are more complex than simple scalars or vectors, to the best of our knowledge, no method yet has considered time series data exhibiting the slowly diverging nature like weather forecast ensembles, which gives rise to the time-hierarchical clustering we propose in this work.

The visualization of variability in ensembles of 2D scalar fields is often performed using spaghetti plots of iso-contours, especially when applied to weather forecast data [Wil11, QM16]. Because the resulting visualizations often suffer from visual clutter, several simplifications and visual abstractions have been introduced in recent years. Sanyal et al. [SZD*10] use glyphs and graduated ribbons to convey the uncertainty along iso-contours. Different kinds of confidence bands—regions representing the Euclidean spread of a set of iso-contours—are introduced by Whitaker et al. [WMK13], which build upon the concept of statistical band depth, and by our contour variability plots (see Chapter 6), which build upon the concept of standard deviation of signed distance functions.

The visualization techniques for time-varying 2D scalar ensembles we propose in this chapter build upon the concept of space-time cubes. Space-time cubes define a three-dimensional coordinate system, where two dimensions refer to space and the third dimension refers to time [Hö70, Kra03]. Many variants of space-time cubes have been proposed, which we cannot attempt to review here, however, a thorough overview of space-time cubes and their application-specific variations is given in the state-of-the-art report by Bach and co-workers [BDA*14]. Andrienko et al. [AAG03] use space-time cubes for the visualization of temporal events at specific locations in geographic information systems. Tominski and Schulz [TS12] propose the Great Wall of Space-Time, which extrudes a piece-wise linear spatial trajectory along the time dimension.

7.3. Method Overview

We start with an ensemble $\{\mathbf{s}_1^{[1;m]}, ..., \mathbf{s}_n^{[1;m]}\}$ of n 2D time-dependent scalar forecast fields, where each forecast predicts a physical quantity such as geopotential height at consecutive timestep $t_i, i = 1, ..., m$, over a certain period. Subscripts and superscripts, respectively, denote the ensemble member and timesteps over which the dynamics are considered. For a prescribed forecast interval $[t_a; t_b]$, the sub-sequences $\mathbf{s}_1^{[a;b]}, ..., \mathbf{s}_n^{[a;b]}$ are treated as n single elements. For a given value v, the iso-contours where the forecast fields of these sub-sequences take on v are clustered into k clusters. k, a and b are given by the user. The iso-contours in the selected sub-sequences are clustered so that those having similar geometry *and* similar motion trajectories fall into the same groups.

Starting with the initial k clusters, we proceed backward in time and reconsider this clustering at every timestep $t_i, i = (a-1), ..., 1$ with respect to the iso-contours of ensembles $\mathbf{s}_1^{[i,i]}, ..., \mathbf{s}_n^{[i,i]}$. If the

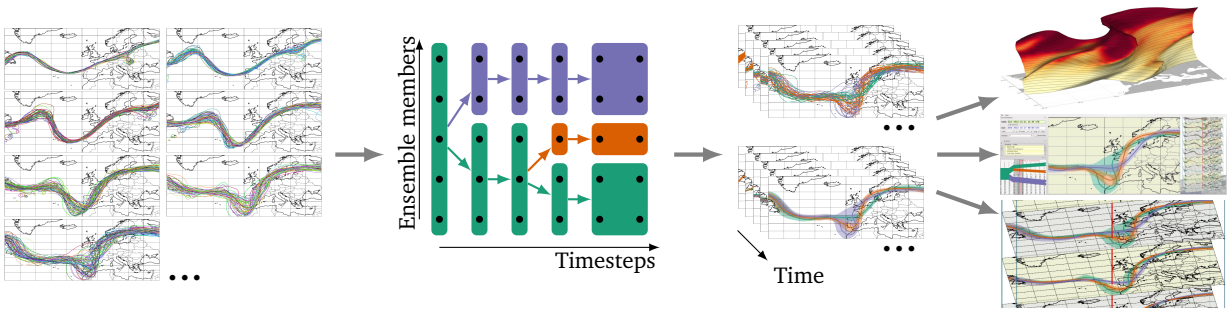


Figure 7.3.: Overview: **(a)** Ensemble of time-varying iso-contours. **(b)** Time-hierarchical clustering of the ensemble members. **(c)** Spaghetti plots of clustered contours & variability plots. **(d)** Our visualizations: Space-time cluster surface, interactive cluster selection and visualization, stacked time-cuts.

dissimilarity of two clusters falls below a certain threshold at timestep i, the two clusters are merged and in the next iteration the reduced set of clusters is considered.

Even though many different clustering algorithms can be be used for creating the cluster hierarchy, we decided to adapt the contour clustering method of contour variability plots (see Chapter 6). I.e., we use agglomerative hierarchical clustering (AHC), in combination with a similarity metric for iso-contours that is based on a principal component transformation of signed distance fields in which these contours are the level-0 sets. The advantage of this approach is twofold: Firstly, it comes with an abstract cluster representation showing the variability of contours per cluster. Secondly, it can be used efficiently to determine a median contour per cluster. Both issues are important in our approach for encoding the cluster hierarchy visually and, thus, to overcome the mentioned limitations of spaghetti plots. A detailed description of the extension to the time-hierarchical clustering approach follows in the next section.

By using the information encoded in the hierarchical cluster tree, in combination with abstract visual cluster representations, we provide three different options for analyzing the contour data: a) The user can interactively animate through the timesteps and let the contours or clusters at every timestep be visualized. b) From the cluster medians a continuous space-time surface can be computed and visualized for each of the k initial clusters. c) A plot of stacked time-cuts showing the abstract cluster representation on a subset of timesteps can be selected. The work flow and visualization options provided by our approach are illustrated in Fig. 7.3.

7.4. Time-hierarchical Clustering

We start with an ensemble of n scalar forecast fields evolving over a sequence of m timesteps. Our general goal is to compute a clustering of the iso-contours in these fields to a fixed iso-value ν. This

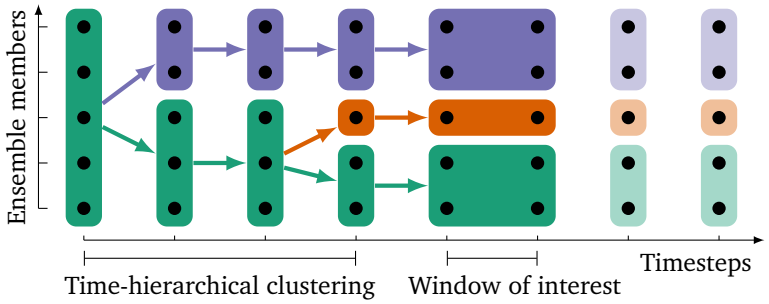


Figure 7.4.: Time-hierarchical clustering: Based on an initial clustering of the ensemble members in a user-selected window of interest, we build a hierarchy of clusters by merging them in time-reversed order. All later timesteps get assigned the clustering from the window of interest.

clustering should reflect the diverging nature of the weather forecast ensemble.

In general it is not possible to find a single clustering that is valid at all timesteps. To account for this, we let the user select a time-window of interest comprising all timesteps in the interval $[t_a, t_b]$ ($t_a = t_b$ is allowed). We then perform an initial clustering of the iso-contours according to a similarity of the contours "averaged" over this time-window. Based on the initial clustering, our basic approach is to go through the sequence of timesteps in reversed order, starting at time $t_a - 1$, and recursively merge pairs of clusters as soon as their similarity exceeds a certain threshold. Because we do not allow clusters to split or individual members to change their cluster, this yields a tree-like hierarchy of clusters, where each merge is associated to a specific timestep. This is illustrated in Fig. 7.4.

Initial Clustering and Contour Variability Plots

Since the time-hierarchical clustering is based on the contour clustering of contour variability plots, and since we make use of contour variability plots to generate an abstract cluster representation that can be used in our time-hierarchical visualization approach, we will provide a brief summary of the relevant parts of Chapter 6.

We represent the contours by signed distance functions (SDFs). Let $\mathbf{s}_1, ..., \mathbf{s}_n \in \mathbb{R}^M$ be a set of scalar fields (defined on a grid with M vertices) in which the iso-contours of interest are given implicitly via the iso-value v. For each $\mathbf{s}_i$, a signed distance transform according to the given iso-value is performed which yields corresponding SDFs $\mathbf{d}_i \in \mathbb{R}^M$. Each input contour i is now represented by a SDF $\mathbf{d}_i$, which can be interpreted as a high-dimensional point in $\mathbb{R}^M$, and, hence, the input set of iso-contours can be treated as an M-dimensional point cloud. In this representation, standard clustering methods can be applied to find clusters of iso-contours. For instance, AHC based on Euclidean distances and average linking, followed by an automated guess for an optimal number of clusters using the L-Method (see [SC04] and Section 4.3.3). Furthermore, a principal component analysis of the point cloud

can be performed first, to reduce the dimensionality of the space in which clustering is performed from M to $N << M$, yet in the case of the time-hierarchical clustering we omit this step because it does not improve performance (in Chapter 6, the dimensionality reduction is mostly required for the computation of co-occurrence probabilities).

Based on a given clustering of SDFs, a median contour can be computed very efficiently. Let $C \subseteq \{1,\dots,n\}$ denote a cluster of SDFs, given as a point cloud in M-dimensional space. The so-called *geometric median* is then computed as the point which has the least sum of squared distances to all other points in the cluster (and generally does not coincide with an existing point). Given this median point, respectively the corresponding SDF, the *median contour* is implicitly given as its zero-contour.

Furthermore, a full *contour variability plot* is obtained by adding co-called *confidence bands*, which indicate the spatial standard deviation of the contours in a cluster. The band to a given number $\alpha > 0$ of standard deviations is obtained as the region enclosed between the zero-contours of the two artificial SDFs

$$\mathbf{d}_{1,2} := \underset{i\in C}{\text{mean}}\{\mathbf{d}_i\} \pm \alpha \cdot \underset{i\in C}{\text{std}}\{\mathbf{d}_i\}, \tag{7.1}$$

where "mean" and "std" are operators which compute the component-wise (i.e., grid-point-wise) mean values and standard deviations, respectively. Intuitively, since each component of a SDF $\mathbf{d}_i$ corresponds to a grid point, the band contains all grid points at which the value zero is within α standard deviations of all occurring values, i.e., grid points which are "likely" to be close to one of the given contours. We use $\alpha = 1$ throughout this chapter.

Agglomerative Hierarchical Clustering

AHC is one of the most commonly used clustering methods (a comprehensive overview is given in Section 4.3.2; for a general overview of clustering techniques let us also refer to the overview article by Jain [Jai10]). Given n observations $\mathbf{x}_1,\dots,\mathbf{x}_n$, AHC builds a hierarchy of clusters (an unbalanced binary tree) from which clusterings with different numbers of clusters can be derived. First, each observation is put into a separate cluster of cardinality one, and the hierarchy is then built by repeatedly merging pairs of similar clusters until all points are contained in a single cluster. To decide which clusters are merged in every iteration, AHC uses a distance metric and a so-called linkage criterion.

The distance metric determines the similarity between individual observations and can be generically specified through a distance matrix $D \in \mathbb{R}^{n\times n}$, where each entry $(D)_{ij}$ indicates the distance, or dissimilarity, between observations i and j. Throughout this chapter, we use the Euclidean distances $(D^t)_{ij} = \left|\left|\mathbf{s}_i^{[t;t]} - \mathbf{s}_j^{[t;t]}\right|\right|$ between the SDFs of corresponding iso-contours as distance metric. If an interval $[t_a;t_b]$ consisting of multiple timesteps is considered, we use the "mixed" Euclidean

distances

$$\left(D^{[a;b]}\right)_{ij} = \sqrt{\sum_{t=a}^{b} \left\| \mathbf{s}_i^{[t;t]} - \mathbf{s}_j^{[t;t]} \right\|^2}.$$

The linkage criterion, on the other hand, specifies how distances between clusters are determined from the distances $(D)_{ij}$, such that in every iteration the pair of clusters with the smallest distance can be determined for merging. More precisely, given a clustering of $\{1,...,n\}$ into k disjoint subsets, the linkage criterion defines inter-cluster distances as a linkage matrix $L \in \mathbb{R}^{k\times k}$ which is a function of the point-wise distances given in D. For practical uses, linkage criteria are typically expressed through a recursive update rule. Starting with $L := D$, this update rule specifies how the distances in L have to be adapted when two clusters are merged. When two disjoint clusters $I, J \subset \{1,...,n\}$ are combined into a new cluster $I \cup J$, the corresponding two rows and two columns in L are collapsed and the new entries—the distances of the new cluster to other clusters $K \subset \{1,...,n\}$— are determined from the distances between I, J and K:

$$(L)_{I\cup J,K} = f\Big((L)_{I,K}, (L)_{J,K}, (L)_{I,J}\Big).$$

For the time-hierarchical clustering, we use linking according to Ward's method [Jr.63] because it produces more equally sized clusters than average linking as used for streamline and contour variability plots in Chapters 5+6. This linkage criterion seeks to minimize the sum of within-cluster variances, and the corresponding update rule is:

$$(L)_{I\cup J,K} = \frac{(|I|+|K|)\cdot(L)_{I,K} + (|J|+|K|)\cdot(L)_{J,K} - |K|\cdot(L)_{I,J}}{|I|+|J|+|K|}.$$

Note that the choice of linkage criterion is application dependent and has a strong influence on the resulting clusterings. Hence, different criteria might be preferable in other cases (e.g., average, single or even complete linking, which are described in more detail in Section 4.3.2).

Generation of the Time-hierarchy

To generate the required time-hierarchical clustering, we follow the idea of AHC and use a linkage criterion to determine when clusters should be merged. The major difference is that now we are considering multiple timesteps and that the distances between the observations are time-dependent, i.e., there is a different distance matrix $D^t \in \mathbb{R}^{n\times n}$ in every timestep.

We start by defining an inter-cluster distance threshold τ, which is used to decide when clusters will be merged. Since different distance metrics and inputs can result in distance values at vastly

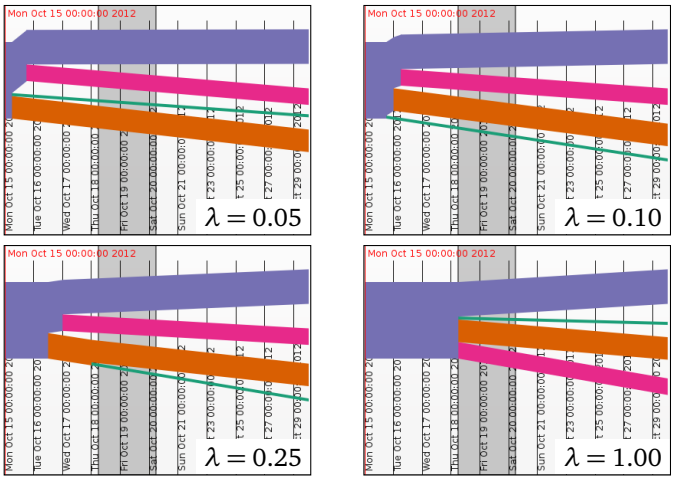


Figure 7.5.: Effect of different thresholds $\tau = \lambda \cdot \delta$ on the time-hierarchy of clusters for the scenario shown in Fig. 7.10 (b). The horizontal axis corresponds to time, the gray region shows the window of interest ($= [t_a; t_b]$) over which the initial clustering is performed and the width of the colored bars corresponds to the cardinality of the clusters.

different scales, we make this threshold depend on the initial clustering in our time-window of interest. Given the corresponding distance matrix $D^{[a;b]}$ that was used to generate the initial clustering with k clusters, we compute the corresponding linkage matrix $L^{[a;b]} \in \mathbb{R}^{k \times k}$. Let δ be the smallest non-zero entry of $L^{[a;b]}$. Then, in the sense of AHC, δ is the minimum distance between the clusters in the initial clustering, and we choose the inter-cluster distance threshold relative to δ as $\tau := \lambda \cdot \delta$ with $\lambda \in]0; 1]$. The factor λ can be used to control how fast, or aggressive, our time-hierarchical clustering merges clusters. Larger λ will cause clusters to be merged more aggressively, while smaller λ will defer the merging of clusters to earlier timesteps.

Once τ has been selected, we iterate through the sequence of timesteps in reversed order, starting at timestep $a-1$, and try to merge clusters as soon as their distance according to the linkage criterion becomes smaller than τ. In every timestep t, similar to before, we use the current clustering with k clusters and the current distance matrix D^t to compute the corresponding linkage matrix $L^t \in \mathbb{R}^{k \times k}$. If any non-zero entry in L^t is smaller than τ, we merge the two corresponding clusters and update L^t (analogously to regular AHC). We repeat this procedure in timestep t until no more merges can be performed with a distance smaller than τ, and then proceed to timestep $t-1$. After processing the first timestep, our time-hierarchical clustering is finished. Note that, in general, the resulting hierarchy is not a binary tree because more than one merge can occur in one timestep, and that there is no guarantee that all clusters are merged into a single cluster after processing the first timestep. The number of clusters in the first timestep, however, can be influenced by changing λ. The effect of different choices for λ on the time-hierarchy of clusters is illustrated in Fig. 7.5.

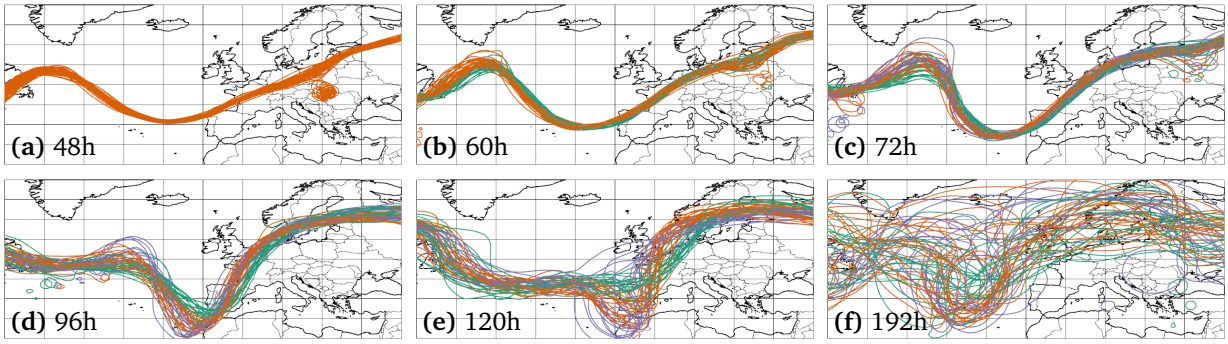


Figure 7.6.: Spaghetti plots of an ensemble of 5600 m geopotential height contour lines at 500 hPa. Each plot shows the lines of all 51 members of the ECMWF ENS forecast, colored by cluster membership. The forecast from 00:00 UTC 15 October 2012, valid at the indicated lead times, is shown.

7.5. Space-time Visualization

Given the time-hierarchical clustering of iso-contours, the corresponding cluster tree has to be visualized so that both aspects, the time evolution of single clusters and the splitting of clusters over time, can be conveyed effectively. This is difficult to achieve, since it requires to show when and where variations occur that cause clusters to change. In principle, common techniques including juxtaposition and animation of spaghetti plots of the contours in each timestep can be applied, further enhanced by coloring the contours according to their cluster membership.

Fig. 7.6 shows juxtaposes, i.e., side-by-side placement in one view, of spaghetti plots of the 51 iso-contours in a weather forecast at six different time steps, colored according to the proposed time-hierarchical clustering. One can see immediately that from this kind of visualization it is difficult to grasp the major spatial shape and variation of the clusters, their spatial changes over time, and the temporal split events. Animation, on the other hand, besides its known limitations, fails especially when a sequence of static spaghetti plots is displayed. This is due to the large number of contour lines that need to be memorized from image to image and the often rather chaotic appearance of the plots.

In a first attempt to improve the visualization, one can replace the spaghetti plots in each timestep by an abstract spatial cluster representation, for instance, via contour boxplots [WMK13], variability plots, or any other suitable abstraction. As seen in Fig. 7.7, this helps to better convey the clusters' shapes, variations, and splits. Yet it is still difficult to perceive the relationships between the structures in timesteps which are not placed close to each other, a common problem of juxtaposition as pointed out by Gleicher et al. [GAW*11].

To overcome the limitations of juxtaposition and animation, we propose to visualize the time evolution of clusters via new techniques that built upon the concept of *space-time cubes* [Ḧ70, Kra03]. Space-time cubes define a 3D orthogonal coordinate system, where two dimensions refer to space and

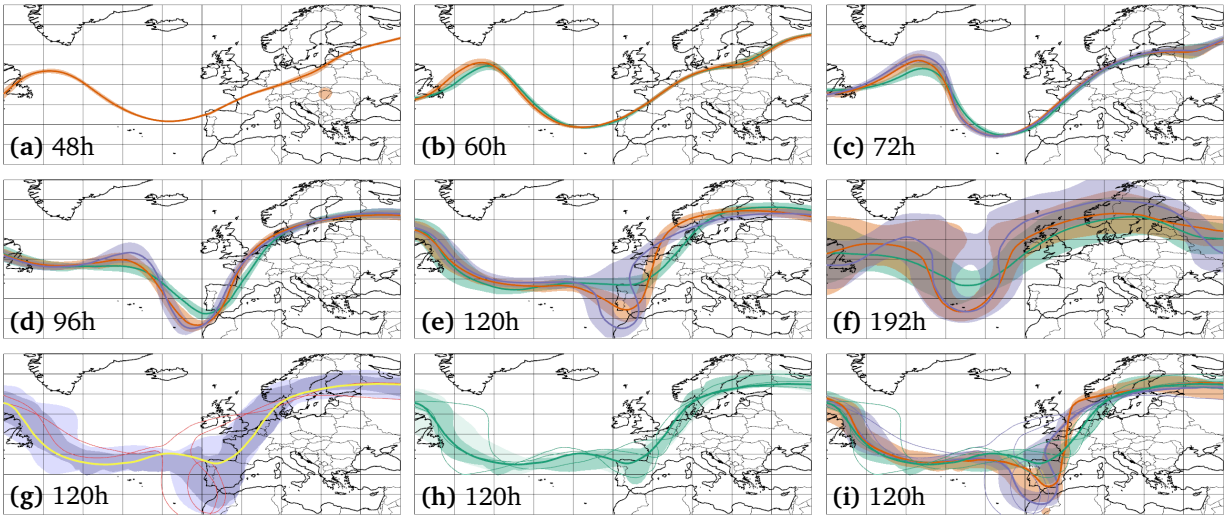


Figure 7.7.: Analoguous to Fig. 7.6. **(a-f)** Contour variability plots (see Chapter 6). **(g-i)** Contour boxplots as proposed by Whitaker et al. [WMK13]: **(g)** Contour boxplot computed from all 51 ensemble members, **(h)** from a single cluster, and **(i)** one contour boxplot computed from each cluster (100% data depth bands omitted for simplicity).

the third dimension refers to time. Spatio-temporal data points are displayed in this coordinate system to reveal spatio-temporal patterns in the data. In an between-subjects experiment [KDA*09] it was demonstrated that users, when using space-time cubes, needed only half of the time that was required when using alternative techniques, to answer complex questions requiring an overall understanding of the spatio-temporal structure of the data.

7.5.1. Space-time Cluster Surfaces

Firstly, we propose the visualization of the temporal cluster evolution by connecting iso-contours across time to form a *space-time cluster surface*. The approach has similarities to the Great Wall of Space-Time by Tominski and Schulz [TS12], yet it does not extrude a piecewise linear base contour along constant time trajectories, but forms a smooth surface from arbitrarily given iso-contours at every timestep. To build the surface, we first compute in every timestep and for every cluster a representative median contour (see Section 7.4). Other representatives, like the mean contour, are possible, yet since a Gaussian distribution assumption is not always justified for the spread of contours belonging to one cluster, we decided to use solely median contours for this purpose.

Instead of first extracting polygonal representations of the median contours and constructing the cluster surfaces from them, we propose a different approach: For every cluster, we take the signed distance fields of the median contours and stack them together consecutively with respect to time to form a space-time distance volume. By drawing the level-0 sets in these volumes using volumetric

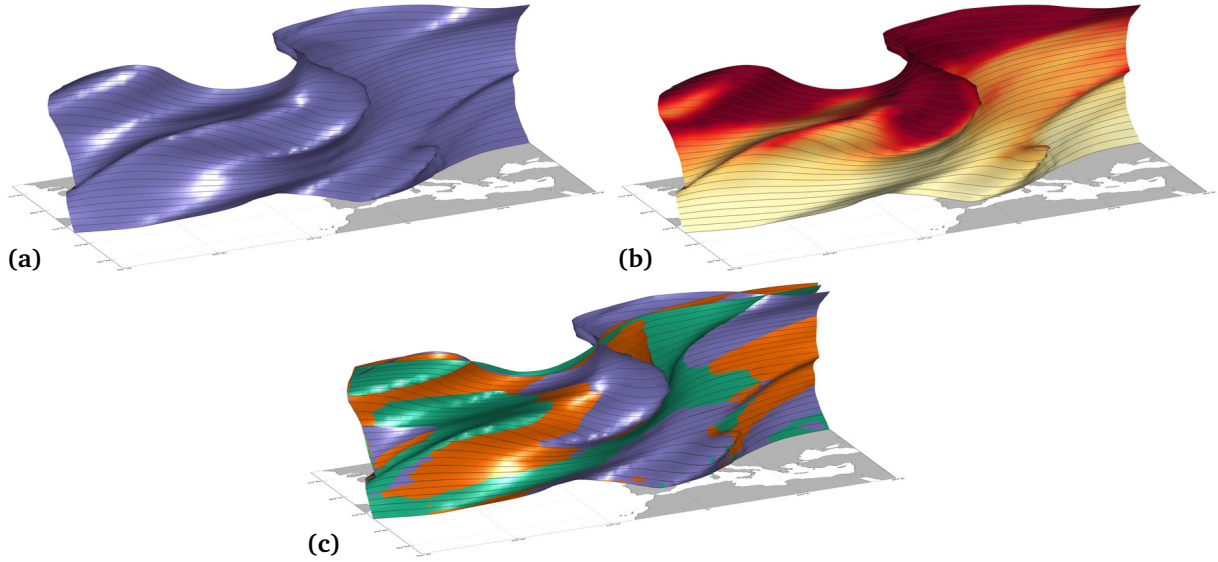


Figure 7.8.: Space-time cluster surfaces: **(a)** Single cluster median. **(b)** Same as (a) but with color coded standard deviation. **(c)** Simultaneous visualization of all time-hierarchical clusters.

ray-casting and trilinear interpolation, temporal interpolation between contours is performed and a smoothly varying cluster surface is rendered. A drawback of this approach is that it can result in an inconsistent surface, for instance, when a contour splits into two disconnected contours from one timestep to the next. However, if the time step of the numerical weather prediction output is small enough for the scale of the considered atmospheric phenomenon, such inconsistencies are rare. As an example, Fig. 7.8 (a) shows a space-time cluster surface for a single cluster evolving over time.

To also reveal the spread of the contours belonging to a cluster, the points of the space-time cluster surfaces are colored in the following way: At every surface point, we compute the standard deviation of the signed distance values of all contours, i.e., their distance fields, belonging to the cluster. The standard deviation is then mapped to color, as shown in Fig. 7.8 (b). In the shown example, the coloring shows clearly the increasing spatial spread of the cluster members with advancing time. Also, the plot shows increased spread along the trough which is moving eastwards.

An apparent limitation of cluster surfaces are occlusions which are introduced especially when a highly curved cluster surface consisting of multiple layers is visualized. It is clear that in such a case it is not always possible to perceive from the surface the overall shape of the cluster. On the other hand, this problem can at least partly be alleviated by rendering the surface on the GPU. Even for high resolution fields, GPU volume ray-casting can achieve very high frame rates, enabling an interactive navigation and a detailed exploration of the cluster surface from different perspectives. In combination with clipping planes and by using transparency and shading effects to emphasize specific features, a powerful visual encoding of a time-varying cluster of iso-contours is given.

In principle, the entire cluster tree can be rendered in one single image containing all individual cluster surfaces, via multi-parameter volume rendering, i.e., by testing at every sample point along the rays against the stacked signed distance volumes of all clusters, and rendering simultaneously the implicitly given cluster surface in each volume. The resulting visualization is shown in Fig. 7.8 (c), and it immediately shows the expected problem of this kind of visualization: Since cluster surfaces overlap and occlude each other, it is hardly possible to grasp the relevant spatial and temporal changes in one single view. It can nevertheless be perceived clearly, how the clusters diverge from each other over time. This is indicated by the fan-like appearance of the surfaces tree, indicating the fanning out of the cluster medians with increasing time.

7.5.2. Stacked Time-cuts

To overcome the restriction of space-time cluster surfaces to only one single cluster, we propose an alternative visualization which also builds upon space-time cubes. The major problem of cluster surfaces as proposed is that relevant geometric features can be occluded by other surface parts and spatial reference is increasingly lost with advancing time, i.e., increasing distance of geometric features to the base plane. To alleviate these problems, we first restrict the visualization of the cluster information to a discrete set of timesteps, so-called time-cuts, by rendering these cuts as a set of slices stacked on top of each other (see Figs. 7.10 (a+b)). The advantage of such a slice based visualization is that it enables an almost un-occluded view on the clusters in each slice, at the same time exploiting the humans's perceptional capabilities to form a mental image of the information in-between. Similar to space-time cluster surfaces, however, care needs to be taken to choose the time step between two slices small enough for the scale of the considered phenomenon.

With space-time slices, however, it is still difficult to identify the spatial relationships between features at different times, because no common points of orientation are given in the slices. Therefore, we slightly modify and enhance the plot by a) visualizing in each slice a map showing the ground over which the space-time cube is built, and b) letting the user place and move interactively so called vertical drop lines at reference locations. In combination with rendering the cube's edges, drawing silhouettes around the slices, and color coding cluster and ground information differently, a significantly enhanced localization of features and their spatial relationships is obtained.

7.6. Results

To demonstrate the practical application of the proposed methods, we discuss two real-world weather forecasting cases that occurred during an atmospheric research campaign ("T-NAWDEX-Falcon", cf. [SBG*14]) in 2012 and analyze the corresponding time-hierarchical clusterings. The cases were also

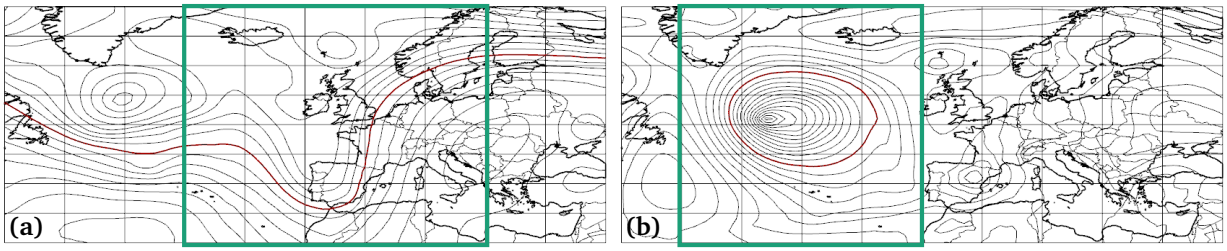


Figure 7.9.: Context for the two real-world examples. **(a)** Geopotential height contour lines of the ECMWF ensemble control forecast at 500 hPa. The forecast from 00:00 UTC 15 October 2012, valid at 12:00 UTC 19 October 2012, is shown. The 5600 m contour line is highlighted in red. The green box shows the selected region of interest. **(b)** The same as (a) but at 925 hPa, showing the forecast from 00:00 UTC 17 October 2012, valid at 00:00 UTC 21 October 2012. The 680 m contour line is highlighted in red.

previously used in Chapter 6 and in the work by Rautenhaus et al. [RKSW15]. Forecast data are obtained from the ECMWF Ensemble Prediction System (ENS). The ensemble comprises an unperturbed control run (i.e., started from the "best" initial conditions) and 50 perturbed members (e.g., [LP08]). Our example region covers the North Atlantic and Europe and encompasses $101 \times 41 \times 62$ grid points. Data are available on vertical levels of constant pressure from the TIGGE archive [SKB*15]. For the present study, we use forecasts from 00:00 UTC 15 October 2012 and from 00:00 UTC 17 October 2012.

Visualization methods including contour variability plots, time-hierarchical cluster tree view and stacked time-cuts have been implemented in the open-source ensemble visualization tool Met.3D [RKSW15]. Fig. 7.1 shows a screenshot of the system in use. The user is able to interactively navigate through time and within the displayed scenes in real-time. Cluster correspondence of the individual ensemble members is precomputed. For the computation of the initial clustering, our implementation uses the MATLAB implementation of AHC. The computation of SDFs via fast marching [Set99] and the time-hierarchical clustering use a custom implementation. All results presented in this chapter were generated on a standard desktop PC (Intel Xeon X5675 processor with 6×3.0 GHz, 8 GB RAM and an NVIDIA Geforce GTX 680). The time required to compute a time-hierarchical clustering is dictated by the computation of SDFs and takes about 1.5 s for both shown cases (the generation of the initial clustering and the time-hierarchy only take about 30ms). Rendering times for the horizontal sections in Met.3D are on the order of a few ms (cf. Table 2 in [RKSW15]).

Example 1: Trough

Our first example examines the temporal development of an upper-level low-pressure trough over the North Atlantic and Europe. We assume the following scenario: On Monday, 15 October 2012, the user (forecaster) is interested in the large-scale atmospheric development towards the end of the week (cf. the scenario described in [SBG*14]). The geopotential height field at 500 hPa is a common

product to judge such developments (e.g., [Ahr08]). Fig. 7.9 (a) shows the forecast geopotential height field of the control forecast, valid at 12:00 UTC 19 October 2012. A distinct trough approaches Europe, extending from Ireland over Spain to Morocco (cf. Fig. 2 in [SBG*14]). We select the 5600 m contour as a representative line for the situation to judge the uncertainty in the ensemble (red line in Fig. 7.9 (a)). To apply the time-hierarchical clustering algorithm, we define a region covering 30° W to 20° E and 30° N to 70° N (green box in Fig. 7.9 (a)) and a 24-hour time window starting at 12:00 UTC 19 October 2012 as the region of interest.

Fig. 7.6 shows spaghetti plots of a number of time steps of the forecast, colored by cluster membership. The plots up to a forecast lead time of 120 h clearly show the approximate development of the 5600 m contour, at 120 h an increased uncertainty in the region of interest can be observed. At longer lead times, the plot quickly becomes illegible. Also, as noted in Section 7.5, the spatial shape of the clusters, their temporal development, and, in particular, the split events are hard to discern.

Our proposed alternative visualizations, the space-time cluster surfaces and the stacked time-cuts with time-hierarchical clustering, are shown in Figs. 7.8+7.10 (a). If the user is interested in a particular cluster, Fig. 7.8 (a) shows the space-time cluster surface. The depiction is similar to a 2D Hovmöller diagram [GDJ*11] and at a glance provides a qualitative depiction of the trough's eastward movement with time. However, in contrast to a 2D Hovmöller diagram, the 3D display frees the color channel to display the corresponding cluster's standard deviation (Fig. 7.8 (b)). As discussed for the spaghetti plots, this depiction as well highlights that uncertainty first increases around the trough.

The cluster tree (Fig. 7.10 (a)) provides a compact summary of the clustering computed from the selected region of interest (highlighted in gray). In the selected time window, the algorithm distinguishes three clusters of roughly similar size (with the purples cluster slightly smaller). In the first 60 h of the forecast, the ensemble members are similar to each other (all in the orange cluster), at 60 h and 66 h lead time, the members split into the final clusters. In the stacked time-cuts, the temporal development of the clusters is immediately visible. A time step of 12 h has been chosen between consecutive slices, small enough to capture the development of the considered large-scale phenomenon. As a spatial reference, we have placed a vertical pole west of Portugal (the pole can be interactively moved by the user). The visualization clearly shows how and when the forecast splits into three scenarios that predict troughs of different strength in the region of interest (purple cluster – strong trough; green cluster – weak trough; orange cluster – in between). Hence, the information of interest (the possible scenarios for the temporal development of the trough) can very quickly be obtained from a single image.

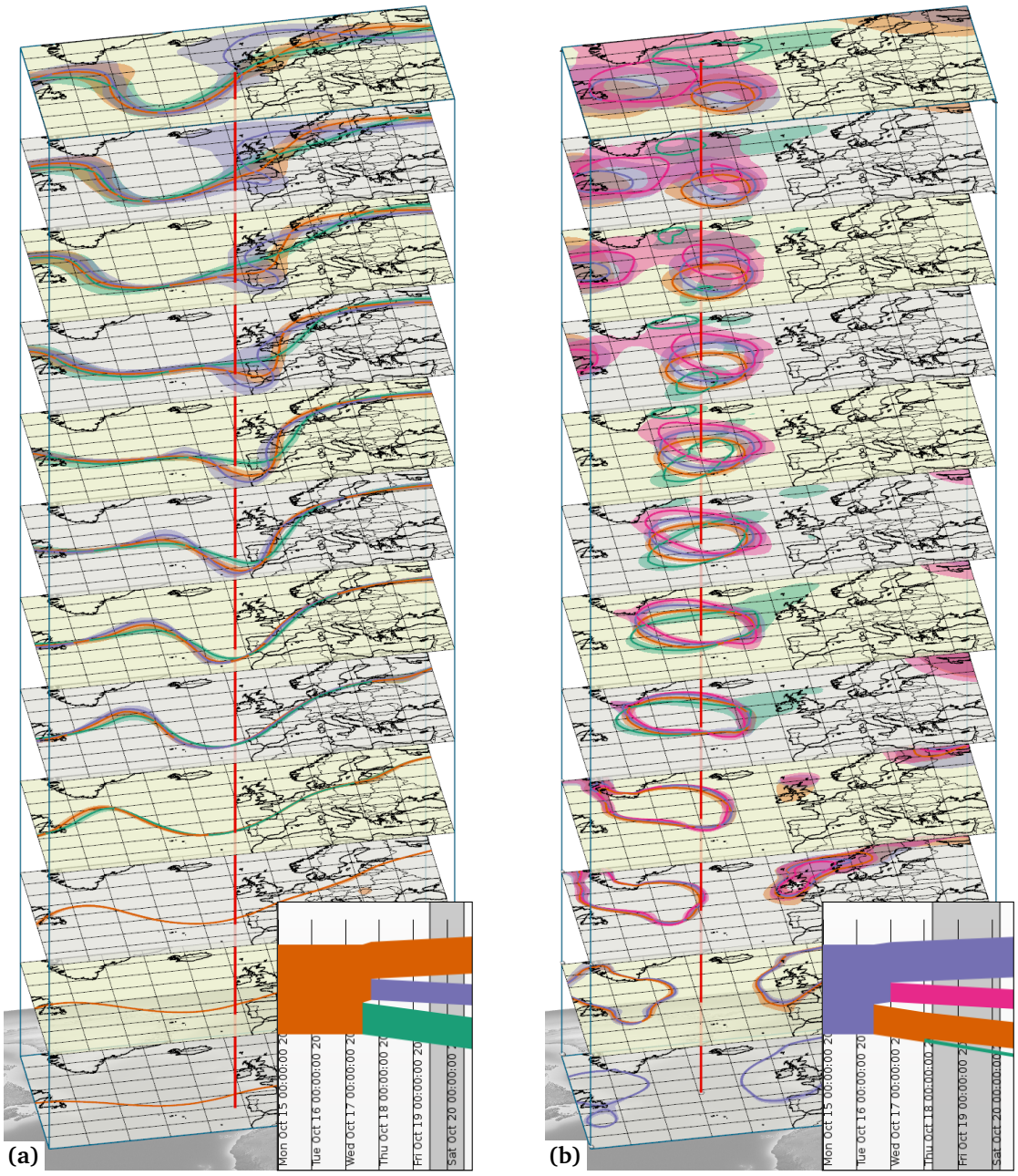


Figure 7.10.: Stacked time-cuts with variability plots. **(a)** Ensemble of 5600 m geopotential height contour lines at an elevation of 500 hPa. The ECMWF ENS forecast from 00:00 UTC 15 October 2012, valid at 00:00 UTC 16 October 2012 (lowest slice, 24 h forecast lead time) to 12:00 UTC 21 October 2012 (uppermost slice, 168 h forecast lead time), is shown. Slices are plotted at 12 h intervals. Note how the ensemble members cluster into forecasts that predict a trough of varying strength. A vertical pole (red) provides a spatial reference for the temporal development of the trough features. The inset shows the corresponding time-hierarchical cluster tree (the temporal region of interest is highlighted in gray). Compare to Fig. 6.8. **(b)** Ensemble of 680 m geopotential height contour lines at an elevation of 925 hPa. The ECMWF ENS forecast from 00:00 UTC 17 October 2012, valid at 00:00 UTC 18 October 2012 (lowest slice, 24 h forecast lead time) to 12:00 UTC 23 October 2012 (uppermost slice, 168 h forecast lead time), is shown. Slices are plotted at 12 h intervals. Compare to Fig. 6.10 (b).

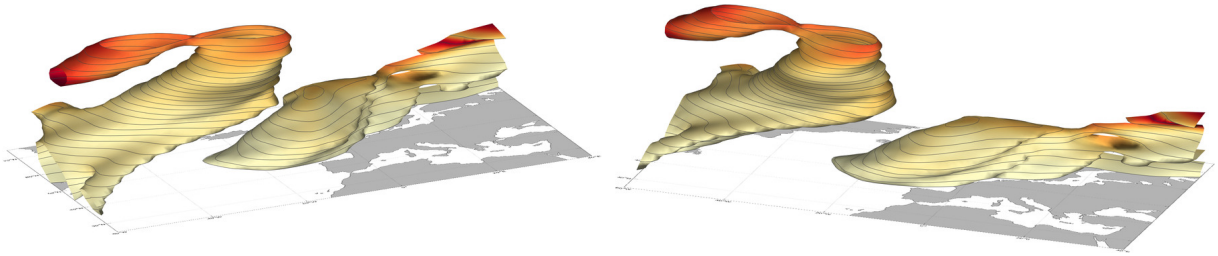


Figure 7.11.: Two different views of the space-time cluster surface of the purple cluster in Fig. 7.10 (b), colored by standard deviation of signed distance values.

Example 2: Low Pressure System Contours

Our second example focuses on the strong low-pressure system which is visible south of Greenland in Fig. 7.9 (a) (former Hurricane Raphael; cf. [SBG*14] and Fig. 6.10 (b)). Closer to the surface at 925 hPa and in the forecast initialized 48 h later (00:00 UTC 17 October 2012), the system can be represented by the 680 m contour line. Fig. 7.9 (b) shows the control forecast valid at 00:00 UTC 20 October 2012, along with a spatial region of interest centered around the system (50° W to 10° W and 30° N to 70° N). As the time window of interest, we define a 48-hour interval starting at 06:00 UTC 20 October 2012.

Fig. 7.1 shows a screenshot of our system during the analysis of the forecast case, Fig. 7.10 (b) shows stacked time-cuts as in Fig. 7.10 (a). In this example, the clustering algorithm distinguishes four forecast scenarios for the region of interest, this time with differing cluster sizes. In particular, note that the green cluster contains only very few members, while the majority of members (i.e., the most likely scenario) is represented by the purple cluster. Again, the stacked time-cuts provide a compact visual summary of the temporal development of the forecasts in a single image. We see that the four clusters mainly differ in shape and orientation in the temporal region of interest, at later time steps they are also shifted in space and differences become larger. The space-time cluster surface of the purple cluster is shown in Fig. 7.11. In this view, the continuous time evolution of the cluster median it is easier to see. In particular, the approach of a second depression towards the end of the forecast period is highlighted. The red color, however, indicates the high associated uncertainty.

Characteristics of Time-hierarchical Clustering

To analyze characteristics of the proposed time-hierarchical clustering technique, we computed average inter-cluster and intra-cluster distances for each timestep t of both example cases. The average inter-cluster distance is computed as the average of all entries in D^t which correspond to a distance between members of two different clusters. The average intra-cluster distance for each cluster is

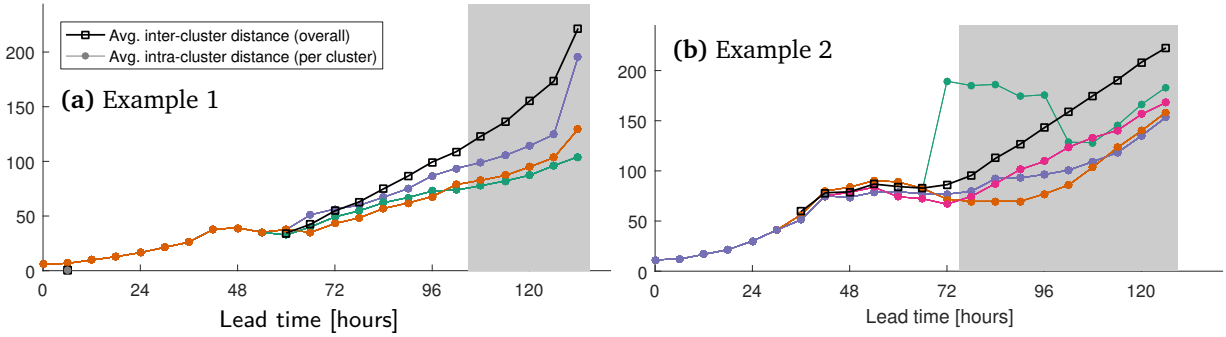


Figure 7.12.: Average inter- and intra-cluster distances for the examples shown in **(a)** Fig. 7.10 (a) and **(b)** Fig. 7.10 (b). Average inter-cluster distances can only be computed in timesteps with more than one cluster.

computed as the average of all non-zero entries in D^t which correspond to a distance between two members of the respective cluster.

Fig. 7.12 shows the temporal evolution of both measures. As expected from the assumption of increasing ensemble spread with time, the average inter-cluster distance increases as well. The only exception can be observed after 54 h lead time in example 2, where the clusters slightly approach each other before starting to diverge again.

The intra-cluster distances also increase with time over most parts of the forecasts, however, distances slightly decreasing with time can be observed as well. If both splits and merges were allowed in our algorithm, decreasing distances (with time) could cause clusters to merge again. This could happen, e.g., in Fig. 7.12 (a), where a "temporary" split might occur starting from 36 to 42 h lead time and ending from 48 h to 54 h lead time. We note that since our algorithm moves backward in time, the identified splits (in forward time direction) are always the last possible splits when compared to the case that re-merging clusters are allowed. The algorithm could also be modified to always find the first split with increasing time.

For our examples, we observe in all timesteps that the average inter-cluster distance is greater or equal than (most of) the average intra-cluster distances. However, the strength of this characteristic decreases with increasing time difference to the time window of interest.

Limitations

Underlying our approach is the assumption that there is a monotonic increase in the number of clusters over time and that merge events do not occur. Thus, given the initial clustering regarding the selected time window, the clusters which are derived via our hierarchical merging backwards in time cannot always represent the contour distribution at earlier time steps equally well. The time-hierarchy rather shows how the initial clusters develop prior to the selected time window. Because of this, misleading results can be generated for ensembles not showing the specific diverging nature which we assume.

In the future we plan to extend our approach with respect to this limitation, by considering split and merge events like the ones discussed in the previous section, and by providing options to analyze the stability of the cluster evolution over time.

7.7. Conclusion

In this chapter we have presented a new approach for visualizing the spatial and temporal evolution of iso-contours in ensembles of 2D time-varying scalar fields. We have achieved this via a specific time-hierarchical clustering of the initial set of contours, and the conversion of the contours per cluster into an abstract representation showing the major trend and spread of the cluster. We have introduced novel visualization approaches for analyzing the evolution of single clusters over time, via space-time cluster surfaces, and the simultaneous depiction of all clusters in a spatio-temporal context, via stacked time-cuts. In combination with additional geo-references, an intuitive understanding of the evolution of the clusters over time is enabled.

8

Conclusion and Future Work

In this thesis, we have presented new techniques for fluid simulation and ensemble visualization. In this last chapter, we summarize our contributions and conclusions, and we present some directions for future work.

Fluid Simulation In the first part of this thesis, we have introduced an efficient method for the simulation of incompressible fluids. To the best of our knowledge, this is the first method to combine a hexahedral finite element formulation on a spatially adaptive octree grid with a multigrid solver. We have specifically considered the simulation of liquids with free surfaces, for which we have extended the basic finite element discretization to a second-order accurate treatment of the fluid/air boundary using a Nitsche type method. The octree grid is adapted to the fluid's (potentially evolving) domain such that its boundary is always represented at the highest available resolution. Towards the interior of the fluid, we use ever coarser cells to effectively make the number of simulation elements proportional to the area of the fluid's boundary. This results in huge memory savings compared to non-adaptive methods, allowing us to simulate—on a single desktop computer—large liquid bodies at extreme effective resolutions. Moreover, in contrast to other adaptive approaches such as unstructured grids, our particular combination of techniques allows us to realize the majority of computations on the octree grid in much the same regular manner as on Cartesian grids. I.e., due to the finite element discretization and the hexahedral grid structure, we can parallelize most steps through the use of a cell based approach with analytically precomputed numerical stencils, and we can handle arbitrary configurations of hanging vertices at level transitions in the octree grid. Further, the hierarchical, semi-regular structure of the octree grid and the algebraic construction of the coarse grid operators allow us to generate a consistent multigrid hierarchy in a generic way. The resulting multigrid solver exhibits excellent, near-optimal convergence rates even in the presence of thin walls and fragmented fluid parts. This is also owed to the multigrid-friendly discretization of the pressure Poisson equation

due to our specific choice of finite elements, and to our cell duplication strategy which we use for the creation of the coarse levels of the multigrid hierarchy.

Altogether, our method is able to simulate large-scale fluid and liquid effects in complex domains and pushes the boundaries of efficient numerical fluid simulations that are subject to memory constraints. Due to the linear time-complexity of multigrid, the time spent on solving the pressure Poisson equation—the typical performance bottleneck—is reduced to a minimum. However, in order to make conclusive statements about the performance of the method, a detailed comparison to other methods is still missing. This will be interesting to do, since the octree significantly reduces the number of simulation elements compared to, e.g., a Cartesian grid. On the other hand, the octree data structure requires dedicated time for maintenance, adds indirections to memory accesses and reduces cache coherency, which already indicates that the results of such a comparison will strongly depend on the underlying scenario. For the purposes of visualization, we have already started to use our simulation method as a tool for generating ensemble datasets of fluid flow (see, e.g., Fig. 1.1). In the future, this could be a great help to investigate, in particular, the visualization of ensembles of *unsteady* flows, which still remains a poorly researched area due to the complexity of the information contained in the data. Lastly, a further next step will be to integrate flow visualization techniques directly into the simulation, which will allow us to explore the possibilities of coupling simulation and visualization in the context of ensemble simulations.

Ensemble Visualization In the second part of this thesis, we have introduced a new visual metaphor—called variability plots—as an alternative to spaghetti plots in ensemble visualization. Based on a simple, high-dimensional feature space representation of an input ensemble of lines, we use principal component analysis, agglomerative hierarchical clustering and multivariate normal distributions to find a continuous statistical model for the line distribution. By transforming geometric medians and confidence ellipses of individual clusters back to the original domain space, we generate artificial median lines and confidence regions, which illustrate the spatial standard deviation of the lines in a cluster. Together, the median lines and confidence regions of all clusters yield a variability plot, which provides an abstract visualization of the major trends and outliers in the input ensemble of lines.

We have demonstrated the application of this general principle to streamlines—as an example for parameterized curves in general—as well as iso-contours. While the application to streamlines requires an expensive random sampling and kernel density estimate to compute confidence regions from confidence ellipses, we have shown that this transformation can be performed analytically for iso-contours. Moreover, since we use signed distance functions for the initial representation of iso-contours, we were able to derive a mathematical foundation for computing probabilities of co-occurrence of iso-contours. We have used these probabilities to complement contour variability plots

with a special color-coding, which, for the first time, offers a way of revealing global correlations between occurrences of iso-contours at different locations in the domain.

In a third application to weather forecast ensembles, we have extended the contour clustering of contour variability plots to provide consistent results over a sequence of timesteps in which the uncertainty in the ensemble can be assumed to grow continuously. For this, we have developed a time-hierarchical clustering, which first performs an initial clustering of the ensemble members with respect to a given time window of interest, and then merges clusters in time-reversed order. The resulting hierarchical sequence of clusterings can be used to reduce the complexity in visualizations of individual timesteps in a temporally consistent way, and thus to create meaningful composite visualizations of multiple timesteps. We have shown two examples for such composite visualizations by creating space-time cluster surfaces and stacked contour variability plots.

In conclusion, our proposed variability plots provide an effective abstraction layer for spaghetti plots, which reduces visual clutter independent of the number of ensemble members. As a major contrast to existing alternative methods for spaghetti plots, we use clustering as a first step in order to extract trends and outliers from the ensemble. It should be noted, however, that our clustering could also be integrated into other approaches. Compared to contour and curve boxplots [WMK13, MWK14], which use the notion of statistical data depth, we use the notion of standard deviation. This is similar to the difference between error plots and boxplots in 1D, and does not favor one of both approaches, but rather means that we have to choose depending on the concrete application at hand. However, this characteristic has the consequence that the median lines and confidence regions in variability plots are artificial, smooth shapes, whereas the median lines and the boundaries of confidence regions in contour/curve boxplots are (parts of) the original input lines.

In the future, we plan to continue our work on variability plots in the following ways: Firstly, it will be interesting to apply the concept of variability plots to other types of line and geometry based features of ensembles, and to investigate whether existing realizations of variability plots can be further improved. For instance, we have shown that three-dimensional variability plots of iso-surfaces can be constructed in the same way as two-dimensional variability plots of iso-contours, yet they suffer from heavy visual clutter and mutual occlusion because multiple transparent surfaces have to be shown simultaneously. Secondly, we aim at improving the detection of outliers in the data. So far, outliers are detected if they show up in a separate cluster of cardinality one, yet no specific mechanism is used to explicitly separate them, e.g., in a preprocess. Lastly, in close collaboration with meteorologists, we intend to perform a thorough comparison of variability plots to other existing alternatives for spaghetti plots. This will not only help to identify the distinct advantages of the different approaches, but may also help to bring abstract visualizations such as variability plots from research into operational practice. There, as of today, simple summary statistics and spaghetti plots are still the dominant means for visualizing the variability in ensemble simulations.

Bibliography

[AAG03] ANDRIENKO N., ANDRIENKO G., GATALSKY P.: Visual data exploration using space-time cube. In *Proceedings of the 21st International Cartographic Conference* (2003), pp. 1981–1983. 134

[ABA00] AFTOSMIS M. J., BERGER M. J., ADOMAVICIUS G.: A parallel multilevel method for adaptively refined Cartesian grids with embedded boundaries, AIAA 2000-0808. In *Proc. 38th AIAA Aerospace Sciences Meeting and Exhibit* (2000). 56

[ABS96] ALMGREN A. S., BELL J. B., SZYMCZAK W. G.: A numerical method for the incompressible Navier-Stokes equations based on an approximate projection. *SIAM Journal on Scientific Computing 17*, 2 (1996), 358–369. 23, 28

[Ahr08] AHRENS C. D.: *Meteorology Today*, 9 ed. Brooks Cole, 2008. 126, 145

[ATW13] ANDO R., THÜREY N., WOJTAN C.: Highly adaptive liquid simulations on tetrahedral meshes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 32*, 4 (2013), 103:1–103:10. 44, 45, 47

[BBB07] BATTY C., BERTAILS F., BRIDSON R.: A fast variational framework for accurate solid-fluid coupling. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 26*, 3 (2007), 100:1–100:7. 52

[BBB10] BROCHU T., BATTY C., BRIDSON R.: Matching fluid simulation elements to surface geometry and topology. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 29*, 4 (2010), 47:1–47:9. 46

[BBC04] BANSAL N., BLUM A., CHAWLA S.: Correlation clustering. *Machine Learning 56*, 1 (2004), 89–113. 112

[BBW*06] BUIZZA R., BIDLOT J.-R., WEDI N., FUENTES M., HAMRUD M., HOLT G., PALMER T., VITART F.: The ECMWF variable resolution ensemble prediction system (VarEPS). *ECMWF Newsletter 108* (2006), 14–20. 70

[BCM01] BROWN D. L., CORTEZ R., MINION M. L.: Accurate projection methods for the incompressible Navier-Stokes equations. *Journal of Computational Physics 168*, 2 (2001), 464 – 499. 50

[BDA*14] BACH B., DRAGICEVIC P., ARCHAMBAULT D., HURTER C., CARPENDALE S.: A review of temporal data visualizations based on space-time cube operations. In *EuroVis - State of the Art Reports* (2014). 134

[BF91] BREZZI F., FORTIN M.: *Mixed and hybrid finite element methods*. Springer-Verlag New York, Inc., 1991. 27

[BFMW12] BÜRGER K., FRAEDRICH R., MERHOF D., WESTERMANN R.: Instant visitation maps for interactive visualization of uncertain particle trajectories. In *Proceedings SPIE* (2012), vol. 8294, pp. 82940P–82940P–12. 102

[BGOS06] BARGTEIL A. W., GOKTEKIN T. G., O'BRIEN J. F., STRAIN J. A.: A semi-Lagrangian contouring method for fluid simulation. *ACM Transactions on Graphics 25*, 1 (2006), 19–38. 46

[BHJ*14] BONNEAU G.-P., HEGE H.-C., JOHNSON C., OLIVEIRA M., POTTER K., RHEINGANS P., SCHULTZ T.: Overview and state-of-the-art of uncertainty visualization. In *Scientific Visualization*. Springer, 2014, pp. 3–27. 1, 91, 111

[BHM00] BRIGGS W. L., HENSON V. E., MCCORMICK S. F.: *A Multigrid Tutorial*, 2 ed. SIAM, 2000. 58

[BJB*11] BHATIA H., JADHAV S., BREMER P.-T., CHEN G., LEVINE J. A., NONATO L. G., PASCUCCI V.: Edge maps: Representing flow with bounded error. In *IEEE Pacific Visualization Symposium* (2011), pp. 75–82. 91

[BKS03] BASHIR F., KHOKHAR A., SCHONFELD D.: Segmented trajectory based indexing and retrieval of video data. In *Proceedings of the International Conference on Image Processing* (2003), pp. 623–629. 93

[BM10] BRUCKNER S., MÖLLER T.: Isosurface similarity maps. *Computer Graphics Forum (Proceedings of EuroVis) 29*, 3 (2010), 773–782. 111, 112, 133

[BPKW03] BRUN A., PARK H.-J., KNUTSSON H., WESTIN C.-F.: Coloring of DT-MRI fiber traces using Laplacian eigenmaps. In *Computer Aided Systems Theory-EUROCAST* (2003), pp. 518–529. 92

[BPMS12] BORN S., PFEIFLE M., MARKL M., SCHEUERMANN G.: Visual 4D MRI blood flow analysis with line predicates. In *IEEE Pacific Visualization Symposium* (2012), pp. 105–112. 92

[BR86] BRACKBILL J. U., RUPPEL H. M.: FLIP: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. *Journal of Computational Physics 65*, 2 (1986), 314–343. 47

[Bra07] BRAESS D.: *Finite Elements: Theory, Fast Solvers, and Applications in Solid Mechanics*, 3 ed. Cambridge University Press, 2007. 16

[Bri08] BRIDSON R.: *Fluid Simulation for Computer Graphics*. A K Peters, 2008. 14, 27, 28, 46

[Bro04] BROWN R.: Animated visual vibrations as an uncertainty visualisation technique. In *Proceedings of GRAPHITE* (2004), pp. 84–89. 91, 111

[BTB15] BAUER P., THORPE A., BRUNET G.: The quiet revolution of numerical weather prediction. *Nature 525*, 7567 (2015), 47–55. 130

[CBB06] CARR H., BRIAN D., BRIAN D.: On histograms and isosurface statistics. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization) 12*, 5 (2006), 1259–1266. 112, 133

[CCG11] CROCKETT R., COLELLA P., GRAVES D.: A Cartesian grid embedded boundary method for solving the Poisson and heat equations with discontinuous coefficients in three dimensions. *Journal of Computational Physics 230*, 7 (2011), 2451–2469. 56

[CCK07] CHEN Y., COHEN J., KROLIK J.: Similarity-guided streamline placement with error evaluation. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization) 13*, 6 (2007), 1448–1455. 92

[CFL28] COURANT R., FRIEDRICHS K., LEWY H.: Über die partiellen Differenzengleichungen der mathematischen Physik. *Mathematische Annalen 100*, 1 (1928), 32–74. 30

[CFL*07] CHENTANEZ N., FELDMAN B. E., LABELLE F., O'BRIEN J. F., SHEWCHUK J. R.: Liquid simulation on lattice-based tetrahedral meshes. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2007), pp. 219–228. 45, 47

[CGG04] COROUGE I., GOUTTARD S., GERIG G.: Towards a shape model of white matter fiber bundles using diffusion tensor MRI. In *IEEE International Symposium on Biomedical Imaging: Nano to Macro* (2004), vol. 1, pp. 344–347. 100, 101

[CHL13] COX J., HOUSE D., LINDELL M.: Visualizing uncertainty in predicted hurricane tracks. *International Journal for Uncertainty Quantification 3*, 2 (2013), 143–156. 107

[Cho68] CHORIN A. J.: Numerical solution of the Navier-Stokes equations. *Mathematics of Computation 22* (1968), 745–762. 12

[CM11a] CHENTANEZ N., MÜLLER M.: A multigrid fluid pressure solver handling separating solid boundary conditions. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2011), pp. 83–90. 46

[CM11b] CHENTANEZ N., MÜLLER M.: Real-time Eulerian water simulation using a restricted tall cell grid. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 30*, 4 (2011), 82:1–82:10. 46, 47

[CWSO13] CLAUSEN P., WICKE M., SHEWCHUK J. R., O'BRIEN J. F.: Simulating liquids and solid-liquid interactions with Lagrangian meshes. *ACM Transactions on Graphics 32*, 2 (2013), 17:1–17:15. 45, 47

[CYY*11] CHEN C.-K., YAN S., YU H., MAX N., MA K.-L.: An illustrative visualization framework for 3D vector fields. *Computer Graphics Forum (Proceedings of Eurographics) 30*, 7 (2011), 1941–1951. 92

[Den05] DENTON A.: Clustering of time series data. *Encyclopedia of Data Warehousing and Mining* (2005), 258–263. 134

[DGW11] DICK C., GEORGII J., WESTERMANN R.: A real-time multigrid finite hexahedra method for elasticity simulation using CUDA. *Simulation Modelling Practice and Theory 19*, 2 (2011), 801–816. 38

[DH04] DONEA J., HUERTA A.: *Finite Element Methods for Flow Problems*. Wiley, 2004. 10

[DH09] DOLBOW J., HARARI I.: An efficient finite element method for embedded interface problems. *International Journal for Numerical Methods in Engineering 78*, 2 (2009), 229–252. 53

[Dic12] DICK C.: *Computational Steering for Implant Planning in Orthopedics*. Dissertation, Technische Universität München, 2012. 38

[DW89] DOUGLAS J., WANG J. P.: An absolutely stabilized finite element method for the Stokes problem. *Mathematics of Computation 52*, 186 (1989), 495–508. 28

[EFFM02] ENRIGHT D., FEDKIW R., FERZIGER J., MITCHELL I.: A hybrid particle level set method for improved interface capturing. *Journal of Computational Physics 183*, 1 (2002), 83–116. 46

[ELF05] ENRIGHT D., LOSASSO F., FEDKIW R.: A fast and accurate semi-Lagrangian particle level set method. *Computers and Structures 83*, 6-7 (2005), 479–490. 45, 46, 50

[ESW14] ELMAN H. C., SILVESTER D. J., WATHEN A. J.: *Finite Elements and Fast Iterative Solvers with Applications in Incompressible Fluid Dynamics*. Oxford Univ. Press, 2014. 10

[FC11] FERRANTI L., CORTI S.: New clustering products. *ECMWF Newsletter 127* (2011), 6–11. 130, 132

[FF01] FOSTER N., FEDKIW R.: Practical animation of liquids. In *Proceedings of ACM SIGGRAPH* (2001), pp. 23–30. 46

[FKSS13] FERREIRA N., KLOSOWSKI J. T., SCHEIDEGGER C. E., SILVA C. T.: Vector field k-means: Clustering trajectories by fitting multiple vector fields. *Computer Graphics Forum (Proceedings of EuroVis) 32*, 3pt2 (2013), 201–210. 93

[FM96] FOSTER N., METAXAS D.: Realistic animation of liquids. *Graphical Models and Image Processing 58*, 5 (1996), 471–483. 9, 46

[FML16] FOFONOV A., MOLCHANOV V., LINSEN L.: Visual analysis of multi-run spatio-temporal simulations using isocontour similarity for projected views. *IEEE Transactions on Visualization and Computer Graphics 22*, 8 (2016), 2037–2050. 112, 131

[FOKG05] FELDMAN B. E., O'BRIEN J. F., KLINGNER B. M., GOKTEKIN T. G.: Fluids in deforming meshes. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2005), pp. 255–259. 47

[Gaf04] GAFFNEY S.: *Probabilistic Curve-Aligned Clustering and Prediction with Mixture Models*. Ph.d. dissertation, University of California, Irvine, 2004. 112

[GAW*11] GLEICHER M., ALBERS D., WALKER R., JUSUFI I., HANSEN C., ROBERTS J.: Visual comparison for information visualization. *Information Visualization 10*, 4 (2011), 289–309. 130, 140

[GCCH95] GRESHO P. M., CHAN S. T., CHRISTON M. A., HINDMARSH A. C.: A little more on stabilized Q1Q1 for transient viscous incompressible flow. *International Journal for Numerical Methods in Fluids 21*, 10 (1995), 837–856. 23, 28

[GDJ*11] GLATT I., DÖRNBRACK A., JONES S., KELLER J., MARTIUS O., MÜLLER A., PETERS D. H. W., WIRTH V.: Utility of Hovmöller diagrams to diagnose Rossby wave trains. *Tellus A 63*, 5 (2011), 991–1006. 145

[Gen04] GENZ A.: Numerical computation of rectangular bivariate and trivariate normal and t probabilities. *Statistics and Computing 14*, 3 (2004), 251–260. 119, 124

[GF05] GUY R. D., FOGELSON A. L.: Stability of approximate projection methods on cell-centered grids. *Journal of Computational Physics 203*, 2 (2005), 517–538. 23, 27, 28

[GFCK02] GIBOU F., FEDKIW R. P., CHENG L.-T., KANG M.: A second-order-accurate symmetric discretization of the poisson equation on irregular domains. *Journal of Computational Physics 176*, 1 (2002), 205–227. 52

[GPR*00] GARCKE H., PREUSSER T., RUMPF M., TELEA A., WEIKARD U., VAN WIJK J. J.: A continuous clustering method for vector fields. In *Proceedings of IEEE Visualization* (2000), pp. 351–358. 92

[GR04] GRIGORYAN G., RHEINGANS P.: Point-based probabilistic surfaces to show surface uncertainty. *IEEE Transactions on Visualization and Computer Graphics 10*, 5 (2004), 564–573. 91, 111

[GS00] GRESHO P. M., SANI R. L.: *Incompressible Flow and the Finite Element Method: Isothermal Laminar Flow*. Wiley, 2000. 10, 14, 18, 23, 26, 27, 28, 51

[GTR*99] GOUTTE C., TOFT P., ROSTRUP E., NIELSEN F. Å., HANSEN L. K.: On clustering fMRI time series. *NeuroImage 9*, 3 (1999), 298 – 310. 134

[H$\ddot{7}$0] HÄGERSTRAAND T.: What about people in regional science? *Papers in Regional Science 24*, 1 (1970), 7–24. 134, 140

[Hac85] HACKBUSCH W.: *Multi-Grid Methods and Applications*. Springer, 1985. 38, 46, 55

[HAC97] HIRT C. W., AMSDEN A. A., COOK J. L.: An arbitrary Lagrangian-Eulerian computing method for all flow speeds. *Journal of Computational Physics 135*, 2 (1997), 203–216. 47

[HD10] HARARI I., DOLBOW J.: Analysis of an efficient finite element method for embedded interface problems. *Computational Mechanics 46*, 1 (2010), 205–211. 53

[HFB86] HUGHES T. J., FRANCA L. P., BALESTRA M.: A new finite element formulation for computational fluid dynamics: V. Circumventing the Babuška-Brezzi condition: A stable Petrov-Galerkin formulation of the Stokes problem accommodating equal-order interpolations. *Computer Methods in Applied Mechanics and Engineering 59*, 1 (1986), 85–99. 28

[HJWST12] HELLRUNG JR. J. L., WANG L., SIFAKIS E., TERAN J. M.: A second order virtual node method for elliptic problems with interfaces and irregular domains in three dimensions. *Journal of Computational Physics 231*, 4 (2012), 2015–2048. 53

[HLNW11] HLAWATSCH M., LEUBE P., NOWAK W., WEISKOPF D.: Flow radar glyphs—static visualization of unsteady flow with uncertainty. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization) 17*, 12 (2011), 1949–1958. 133

[HNB*06] HOUSTON B., NIELSEN M. B., BATTY C., NILSSON O., MUSETH K.: Hierarchical RLE level set: A compact and versatile deformable surface representation. *ACM Transactions on Graphics 25*, 1 (2006), 151–175. 49

[HOGJ13] HUMMEL M., OBERMAIER H., GARTH C., JOY K.: Comparative visual analysis of Lagrangian transport in CFD ensembles. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization) 19*, 12 (2013), 2743–2752. 92, 112, 133

[HW13] HEINRICH J., WEISKOPF D.: State of the art of parallel coordinates. In *Eurographics - State of the Art Reports* (2013). 133

[HWHJ99] HECKEL B., WEBER G., HAMANN B., JOY K. I.: Construction of vector field hierarchies. In *Proceedings of IEEE Visualization* (1999), pp. 19–25. 92

[Jai10] JAIN A. K.: Data clustering: 50 years beyond k-means. *Pattern Recognition Letters 31*, 8 (2010), 651–666. 92, 133, 137

[JPR*04] JEN D., PARENTE P., ROBBINS J., WEIGLE C., TAYLOR R., BURETTE A., WEINBERG R.: ImageSurfer: A tool for visualizing correlations between two volume scalar fields. In *Proceedings of IEEE Visualization* (2004), pp. 529–536. 112

[Jr.63] JR. J. H. W.: Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association 58*, 301 (1963), 236–244. 79, 99, 138

[JS03] JOHNSON C., SANDERSON A.: A next step: Visualizing errors and uncertainty. *IEEE Computer Graphics and Applications 23*, 5 (2003), 6–10. 1, 111

[KDA*09] KRISTENSSON P. O., DAHLBACK N., ANUNDI D., BJORNSTAD M., GILLBERG H., HARALDSSON J., MARTENSSON I., NORDVALL M., STAHL J.: An evaluation of space time cube representation of spatiotemporal patterns. *IEEE Transactions on Visualization and Computer Graphics 15*, 4 (2009), 696–702. 141

[KFCO06] KLINGNER B. M., FELDMAN B. E., CHENTANEZ N., O'BRIEN J. F.: Fluid animation with dynamic meshes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 25*, 3 (2006), 820–825. 47

[KH13] KEHRER J., HAUSER H.: Visualization and visual analysis of multifaceted scientific data: A survey. *IEEE Transactions on Visualization and Computer Graphics 19*, 3 (2013), 495–513. 133

[KHP12] KENDALL W., HUANG J., PETERKA T.: Geometric quantification of features in large flow fields. *IEEE Computer Graphics and Applications 32*, 4 (2012), 46–54. 133

[KM90] KASS M., MILLER G.: Rapid, stable fluid dynamics for computer graphics. In *Proceedings of ACM SIGGRAPH* (1990), pp. 49–57. 46

[Kra03] KRAAK M.: The space-time cube revisited from a geovisualization perspective. In *Proceedings of the 21st International Cartographic Conference* (2003), pp. 1988–1996. 134, 140

[KWL*04] KINDLMANN G., WEINSTEIN D., LEE A., TOGA A., THOMPSON P.: Visualization of anatomic covariance tensor fields. In *IEEE Engineering in Medicine and Biology Society* (2004), pp. 1842–1845. 112

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings of ACM SIGGRAPH* (1987), pp. 163–169. 53

[LGF00] LEVENTON M., GRIMSON W., FAUGERAS O.: Statistical shape influence in geodesic active contours. In *Proceedings of IEEE Computer Vision and Pattern Recognition* (2000), pp. 316–323. 112

[LGF04] LOSASSO F., GIBOU F., FEDKIW R.: Simulating water and smoke with an octree data structure. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 23*, 3 (2004), 457–462. 14, 44, 46

[Lia05] LIAO T. W.: Clustering of time series data—a survey. *Pattern Recognition 38*, 11 (2005), 1857 – 1874. 134

[LLPY07] LUNDSTROM C., LJUNG P., PERSSON A., YNNERMAN A.: Uncertainty visualization in medical volume rendering using probabilistic animation. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization) 13*, 6 (2007), 1648–1655. 91

[LP08] LEUTBECHER M., PALMER T.: Ensemble forecasting. *Journal of Computational Physics 227*, 7 (2008), 3515–3539. 124, 130, 131, 144

[LPK05] LOVE A. L., PANG A., KAO D. L.: Visualizing spatial multivalue data. *IEEE Computer Graphics and Applications 25*, 3 (2005), 69–79. 133

[LPR*09] LIEHR F., PREUSSER T., RUMPF M., SAUTER S., SCHWEN L. O.: Composite finite elements for 3D image based computing. *Computing and Visualization in Science 12*, 4 (2009), 171–188. 58

[Lum67] LUMLEY J. L.: The structure of inhomogeneous turbulent flows. In *Atmospheric turbulence and radio wave propagation* (1967), Nauka, Moscow, pp. 166–178. 95

[LZF10] LENTINE M., ZHENG W., FEDKIW R.: A novel algorithm for incompressible flow using only a coarse grid projection. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 29*, 4 (2010), 114:1–114:9. 46

[MEB*12] MISZTAL M. K., ERLEBEN K., BARGTEIL A., FURSUND J., CHRISTENSEN B. B., BÆRENTZEN J. A., BRIDSON R.: Multiphase flow of immiscible fluids on unstructured moving meshes. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2012), pp. 97–106. 45, 47

[MEH12] MULD T. W., EFRAIMSSON G., HENNINGSON D. S.: Flow structures around a high-speed train extracted using proper orthogonal decomposition and dynamic mode decomposition. *Computers & Fluids 57*, 0 (2012), 87 – 97. 78, 95

[MG07] MIN C., GIBOU F.: Geometric integration over irregular domains with application to level-set methods. *Journal of Computational Physics 226*, 2 (2007), 1432–1443. 53

[MJL*13] MCLOUGHLIN T., JONES M. W., LARAMEE R. S., MALKI R., MASTERS I., HANSEN C. D.: Similarity measures for enhancing interactive streamline seeding. *IEEE Transactions on Visualization and Computer Graphics 19*, 8 (2013), 1342–1353. 92, 112

[MLJ*13] MUSETH K., LAIT J., JOHANSON J., BUDSBERG J., HENDERSON R., ALDEN M., CUCKA P., HILL D., PEARCE A.: OpenVDB: an open-source data structure and toolkit for high-resolution volumes. In *ACM SIGGRAPH Courses* (2013), pp. 19:1–19:1. 44

[MPO07] MEYER K. E., PEDERSEN J. M., ÖZCAN O.: A turbulent jet in crossflow analysed with proper orthogonal decomposition. *Journal of Fluid Mechanics 583* (2007), 199–227. 95

[MST10] MCADAMS A., SIFAKIS E., TERAN J.: A parallel multigrid Poisson solver for fluids simulation on large grids. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2010), pp. 65–74. 44, 46

[MVvW05] MOBERTS B., VILANOVA A., VAN WIJK J. J.: Evaluation of fiber clustering methods for diffusion tensor imaging. In *Proceedings of IEEE Visualization* (2005), pp. 65–72. 92

[MWK14] MIRZARGAR M., WHITAKER R., KIRBY R. M.: Curve boxplot: Generalization of boxplot for ensembles of curves. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization) 20*, 12 (2014), 2654–2663. 73, 89, 90, 92, 107, 110, 112, 153

[NFB07] NOCKE T., FLECHSIG M., BOHM U.: Visual exploration and evaluation of climate-related simulation data. In *Winter Simulation Conference* (2007), pp. 703–711. 133

[Nit71] NITSCHE J.: Über ein Variationsprinzip zur Lösung von Dirichlet-Problemen bei Verwendung von Teilräumen, die keinen Randbedingungen unterworfen sind. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg 36*, 1 (1971), 9–15. 53

[NKJF09] NESME M., KRY P. G., JEŘÁBKOVÁ L., FAURE F.: Preserving topology and elasticity for embedded deformable models. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 28*, 3 (2009), 52:1–52:9. 56

[OF03] OSHER S., FEDKIW R.: *Level set methods and dynamic implicit surfaces*. Springer, 2003. 50, 52, 61

[OGHT10] OTTO M., GERMER T., HEGE H.-C., THEISEL H.: Uncertain 2D vector field topology. *Computer Graphics Forum (Proceedings of Eurographics) 29*, 2 (2010), 347–356. 91, 133

[OJ14] OBERMAIER H., JOY K.: Future challenges for ensemble visualization. *IEEE Computer Graphics and Applications 34* (2014), 8–11. 91, 111

[OLK*14] OELTZE S., LEHMANN D. J., KUHN A., JANIGA G., THEISEL H., PREIM B.: Blood flow clustering and applications in virtual stenting of intracranial aneurysms. *IEEE Transactions on Visualization and Computer Graphics 20*, 5 (2014), 686–701. 92, 110, 112, 133

[PG00] POLICKER S., GEVA A. B.: Nonstationary time series analysis by temporal clustering. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 30*, 2 (2000), 339–343. 133

[PH11] PÖTHKOW K., HEGE H.-C.: Positional uncertainty of isocontours: Condition analysis and probabilistic measures. *IEEE Transactions on Visualization and Computer Graphics 17*, 10 (2011), 1393–1406. 91, 111

[PH13] PÖTHKOW K., HEGE H.-C.: Nonparametric models for uncertainty visualization. *Computer Graphics Forum (Proceedings of EuroVis) 32*, 3pt2 (2013), 131–140. 91

[PKRJ10] POTTER K., KNISS J., RIESENFELD R., JOHNSON C. R.: Visualizing summary statistics and uncertainty. *Computer Graphics Forum (Proceedings of EuroVis) 29*, 3 (2010), 823–831. 133

[Pop03] POPINET S.: Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries. *Journal of Computational Physics 190*, 2 (2003), 572–600. 48

[PPBT12] PIRINGER H., PAJER S., BERGER W., TEICHMANN H.: Comparative visual analysis of 2D function ensembles. *Computer Graphics Forum (Proceedings of EuroVis) 31*, 3pt3 (2012), 1195–1204. 133

[PPH12] PETZ C., PÖTHKOW K., HEGE H.-C.: Probabilistic local features in uncertain vector fields with spatial correlation. *Computer Graphics Forum (Proceedings of EuroVis) 31*, 3pt2 (2012), 1045–1054. 91

[PRW11] PFAFFELMOSER T., REITINGER M., WESTERMANN R.: Visualizing the positional and geometrical variability of isosurfaces in uncertain scalar fields. *Computer Graphics Forum (Proceedings of EuroVis) 30*, 3 (2011), 951–960. 91, 111

[PW12] PFAFFELMOSER T., WESTERMANN R.: Visualization of global correlation structures in uncertain 2D scalar fields. *Computer Graphics Forum (Proceedings of EuroVis) 31* (2012), 1025–1034. 112

[PW13] PFAFFELMOSER T., WESTERMANN R.: Visualizing contour distributions in 2D ensemble data. In *EuroVis - Short Papers* (2013). 71, 72, 133

[PWB*09] POTTER K., WILSON A., BREMER P.-T., WILLIAMS D., DOUTRIAUX C., PASCUCCI V., JOHNSON C. R.: Ensemble-Vis: A framework for the statistical visualization of ensemble data. In *Proceedings of IEEE International Conference on Data Mining Workshops* (2009), pp. 233–240. 71, 91, 112

[PWH11] PÖTHKOW K., WEBER B., HEGE H.-C.: Probabilistic marching cubes. *Computer Graphics Forum (Proceedings of EuroVis) 30*, 3 (2011), 931–940. 91, 111

[PWL97] PANG A. T., WITTENBRINK C. M., LODHA S. K.: Approaches to uncertainty visualization. *The Visual Computer 13*, 8 (1997), 370–390. 1, 91, 111

[QM16] QUINAN P. S., MEYER M.: Visually comparing weather features in forecasts. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization) 22*, 1 (2016), 389–398. 112, 134

[RDT06] RATHI Y., DAMBREVILLE S., TANNENBAUM A.: Statistical shape analysis using kernel PCA. In *Proceedings SPIE* (2006), vol. 6064, p. 60641B. 112, 133

[RKSW15] RAUTENHAUS M., KERN M., SCHÄFLER A., WESTERMANN R.: Three-dimensional visualization of ensemble weather forecasts – Part 1: The visualization tool Met.3D (version 1.0). *Geoscientific Model Development 8*, 7 (2015), 2329–2353. 124, 126, 144

[RT12] RÖSSL C., THEISEL H.: Streamline embedding for 3D vector field exploration. *IEEE Transactions on Visualization and Computer Graphics 18*, 3 (2012), 407–420. 92

[SBG*14] SCHÄFLER A., BOETTCHER M., GRAMS C. M., RAUTENHAUS M., SODEMANN H., WERNLI H.: Planning aircraft measurements within a warm conveyor belt. *Weather 69*, 6 (2014), 161–166. 143, 144, 145, 147

[SC04] SALVADOR S., CHAN P.: Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence* (2004), pp. 576–584. 79, 82, 83, 100, 121, 136

[SCG02] STEVENS D. E., CHAN S. T., GRESHO P.: An approximate projection method for incompressible flow. *International Journal for Numerical Methods in Fluids 40*, 10 (2002), 1303–1325. 23, 28

[Set99] SETHIAN J. A.: *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge Univ. Press, 1999. 124, 144

[SJWS08] SALZBRUNN T., JÄNICKE H., WISCHGOLL T., SCHEUERMANN G.: The state of the art in flow visualization: Partition-based techniques. In *Proceedings of the Simulation and Visualization Conference* (2008), pp. 77–92. 93

[SKB*15] SWINBANK R., KYOUDA M., BUCHANAN P., FROUDE L., HAMILL T. M., HEWSON T. D., KELLER J. H., MATSUEDA M., METHVEN J., PAPPENBERGER F., SCHEUERER M., TITLEY H. A., WILSON L., YAMAGUCHI M.: The TIGGE project and its achievements. *Bulletin of the American Meteorology Society 97*, 1 (2015), 49–67. 144

[Sok58] SOKAL R.: A statistical method for evaluating systematic relationships. *University Kansas Scientific Bulletin 38* (1958), 1409–1438. 99

[SS06] SALZBRUNN T., SCHEUERMANN G.: Streamline predicates. *IEEE Transactions on Visualization and Computer Graphics 12*, 6 (2006), 1601–1612. 92

[Sta99] STAM J.: Stable fluids. In *Proceedings of ACM SIGGRAPH* (1999), pp. 121–128. 9, 28, 29, 46, 51

[Str71] STROUD A.: *Approximate calculation of multiple integrals*. Prentice-Hall, 1971. 54

[STS06] SAUBER N., THEISEL H., SEIDEL H.: Multifield-graphs: An approach to visualizing correlations in multifield scalar data. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization) 12*, 5 (2006), 917–924. 112

[SW15] SPRENGER M., WERNLI H.: The Lagrangian analysis tool LAGRANTO - version 2.0. *Geoscientific Model Development Discussions 8*, 2 (2015), 1893–1943. 104

[SZD*10] SANYAL J., ZHANG S., DYER J., MERCER A., AMBURN P., MOORHEAD R. J.: Noodles: A tool for visualization of numerical weather model ensemble uncertainty. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization) 16*, 6 (2010), 1421–1430. 72, 92, 112, 131, 134

[TH73] TAYLOR C., HOOD P.: A numerical solution of the Navier-Stokes equations using the finite element technique. *Computers & Fluids 1*, 1 (1973), 73–100. 26

[TN14] THOMAS D., NATARAJAN V.: Multiscale symmetry detection in scalar fields by clustering contours. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization) 20*, 12 (2014), 2427–2436. 111, 112, 133

[TOS01] TROTTENBERG U., OOSTERLEE C., SCHÜLLER A.: *Multigrid*. Academic Press, 2001. 58

[TS12] TOMINSKI C., SCHULZ H.-J.: The great wall of space-time. In *Vision, Modeling and Visualization* (2012). 134, 141

[TSB*05] TERAN J., SIFAKIS E., BLEMKER S., NG-THOW-HING V., LAU C., FEDKIW R.: Creating and simulating skeletal muscle from the visible human data set. *IEEE Transactions on Visualization and Computer Graphics 11*, 3 (2005), 317–328. 56

[TvW99] TELEA A., VAN WIJK J. J.: Simplified representation of vector fields. In *Proceedings of IEEE Visualization* (1999), pp. 35–42. 92

[TW98] THOMPSON D. W. J., WALLACE J. M.: The Arctic oscillation signature in the wintertime geopotential height and temperature fields. *Geophysical Research Letters 25*, 9 (1998), 1297–1300. 95

[TWGT10] THÜREY N., WOJTAN C., GROSS M., TURK G.: A multiscale approach to mesh-based surface tension flows. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 29*, 4 (2010), 48:1–48:10. 46

[Wal99] WALL W. A.: *Fluid-Struktur-Interaktion mit stabilisierten Finiten Elementen*. PhD thesis, Universität Stuttgart, 1999. 28

[Wan00] WANG W.: Special bilinear quadrilateral elements for locally refined finite element grids. *SIAM Journal on Scientific Computing 22*, 6 (2000), 2029–2050. 60

[Wil11] WILKS D. S.: *Statistical Methods in the Atmospheric Sciences*. Academic Press, 2011. 70, 91, 95, 112, 125, 130, 134

[WMFB11] WOJTAN C., MÜLLER-FISCHER M., BROCHU T.: Liquid simulation with mesh-based surface tracking. In *ACM SIGGRAPH Courses* (2011), pp. 8:1–8:84. 46

[WMK13] WHITAKER R. T., MIRZARGAR M., KIRBY R. M.: Contour boxplots: A method for characterizing uncertainty in feature sets from simulation ensembles. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization) 19*, 12 (2013), 2713–2722. 73, 90, 91, 92, 110, 112, 131, 134, 140, 141, 153

[WPL96] WITTENBRINK C. M., PANG A. T., LODHA S. K.: Glyphs for visualizing uncertainty in vector fields. *IEEE Transactions on Visualization and Computer Graphics 2*, 3 (1996), 266–279. 133

[WS99] WIJK J. J. V., SELOW E. R. V.: Cluster and calendar based visualization of time series data. In *Proc. of the IEEE Symposium on Information Visualization* (1999), pp. 4–9, 140. 133

[WTGT09] WOJTAN C., THÜREY N., GROSS M., TURK G.: Deforming meshes that split and merge. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 28*, 3 (2009), 76:1–76:10. 46

[YWSC12] YU H., WANG C., SHENE C.-K., CHEN J. H.: Hierarchical streamline bundles. *IEEE Transactions on Visualization and Computer Graphics 18*, 8 (2012), 1353–1367. 92

[ZB05] ZHU Y., BRIDSON R.: Animating sand as a fluid. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 24*, 3 (2005), 965–972. 47

[ZCL08] ZHANG S., CORREIA S., LAIDLAW D. H.: Identifying white-matter fiber bundles in DTI data using an automated proximity-based fiber-clustering method. *IEEE Transactions on Visualization and Computer Graphics 14*, 5 (2008), 1044–1053. 92

[ZHT06] ZHANG Z., HUANG K., TAN T.: Comparison of similarity measures for trajectory clustering in outdoor surveillance scenes. In *Proceedings of the 18th International Conference on Pattern Recognition* (2006), pp. 1135–1138. 92, 93, 112, 133

[ZLC*13] ZHU B., LU W., CONG M., KIM B., FEDKIW R.: A new grid structure for domain extension. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 32*, 4 (2013), 63:1–63:12. 14, 44, 47

[ZWK10] ZEHNER B., WATANABE N., KOLDITZ O.: Visualization of gridded scalar data with uncertainty in geosciences. *Computers & Geosciences 36*, 10 (2010), 1268–1275. 91