TECHNISCHE UNIVERSITÄT MÜNCHEN

FAKULTÄT FÜR INFORMATIK

LEHRSTUHL FÜR COMPUTER GRAPHIK UND VISUALISIERUNG

Visual Abstractions for Analyzing Uncertain Multidimensional Data

Ismail Demir

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.

TECHNISCHE UNIVERSITÄT MÜNCHEN

FAKULTÄT FÜR INFORMATIK

LEHRSTUHL FÜR COMPUTER GRAPHIK UND VISUALISIERUNG

Visual Abstractions for Analyzing Uncertain Multidimensional Data

Ismail Demir

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. J. Krüger, Universität Duisburg-Essen

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

Visualizing multidimensional data plays an important role in many areas of science and engineering. Typically, such data sets are either generated by computer simulations or are acquired from experimental measurements. Throughout this thesis, data from various domains of applications are considered, such as weather forecasting, automotive engineering, and computational fluid dynamics.

In the process of data generation or acquisition, uncertainty of different kinds is introduced. During a simulation run, uncertainty arises mainly because of the discrepancy between the computational model and the underlying true physics. Numerical errors due to approximations of analytical solutions constitute another source of uncertainty. Moreover, uncertainty is always introduced by measurement errors and from interpolation techniques that are used to compensate the lack of collected data.

Visualizing uncertain data is a challenging task, particularly when multidimensional data has to be conveyed to the user. The reason behind this is the great amount of information inherent in such data sets, whereas the user is typically limited to a two-dimensional screen. Hence, methods of visual abstraction are necessary to reduce the degree of complexity, while avoiding artifacts caused by occlusion or visual cluttering. Due to the advances in modern graphics hardware, new opportunities arise to develop such methods.

This thesis provides substantial contributions to this field of research. First, we present a method that enables an interactive visual exploration of multidimensional scalar fields. We estimate progressive surfaces via Kriging interpolation by adding scattered samples. For this purpose, we propose a GPU-accelerated updating scheme for the Kriging interpolation, which enables the incremental construction of surfaces when new sample points are added. Hence, results are quickly available, which allows the user to interactively steer the location, where response surfaces are visualized, and to discover relevant dependency relations.

Secondly, we discuss visualizing 3D scalar field ensembles. Ensembles are usually generated by running repeated simulations using different parameter settings, where each run constitutes a single member. Treating ensembles as multivariate data using traditional visualization techniques is unfeasible for larger numbers of members, yet it is important to reveal uncertainties, correlations and trends in such data sets. To address these aspects, we present an alternative visualization technique, which provides an abstract view, based upon a linear layout of 3D space and by using combined bar and line charts.

Thirdly, we introduce a novel approach for visualizing 3D isosurface ensembles based on rendering silhouettes instead of solid surfaces. Thus, occlusion effects are avoided, while spatial coherence and the major shape of the surfaces are preserved. By providing interactive mechanisms, such as picking, clustering, cutting and animation, the user is able to explore the data on a more detailed level.

Fourthly, we present a novel concept to overcome the limitations of traditional voxel-based surface ray-casting. We introduce a hierarchical closest-point octree representation to improve upon the surface approximation quality of voxel grids. We demonstrate that, compared to a standard voxel hierarchy, our approach results in a significantly lower memory consumption and avoids block artifacts at higher zoom levels. Thus, we are able to load models of high resolutions into GPU memory and achieve high-quality rendering results at interactive rates.

Finally, by extending this concept to ensembles of shapes, we use the closest-point grid to quantify region-wise central tendencies and provide means to find the most central shape. In addition, we construct a locally best-matching shape that communicates relevant local features to the user. Our approach is capable of processing arbitrary non-parametric shapes in two and three-dimensional space and neither requires closure nor orientability.

Zusammenfassung

Das Visualisieren mehrdimensionaler Daten spielt eine wesentliche Rolle in vielen Bereichen von Wissenschaft und Technik. Typischerweise werden solche Datensätze entweder durch Computersimulationen erzeugt oder aus experimentellen Messungen gewonnen. In dieser Arbeit werden Daten aus verschiedenen Anwendungsbereichen in Betracht gezogen, wie etwa aus der Wettervorhersage, der Fahrzeugtechnik und der numerischen Strömungsmechanik.

Der Prozess der Erzeugung oder Erfassung von Daten ist mit verschiedenen Arten von Unsicherheit verbunden. Während eines Simulationslaufes entsteht Unsicherheit vor allem wegen der Diskrepanz zwischen dem Rechenmodell und der zugrundeliegenden tatsächlichen Physik. Numerische Fehler aufgrund von Approximationen analytischer Lösungen bilden eine weitere Quelle von Unsicherheit. Darüber hinaus wird Unsicherheit stets durch Messfehler und von Interpolationstechniken, die verwendet werden, um einen Mangel an gesammelten Daten auszugleichen, verursacht.

Unsichere Daten zu visualisieren ist eine schwierige Aufgabe, insbesondere dann, wenn dem Benutzer mehrdimensionale Daten vermittelt werden sollen. Der Grund dafür ist der große Informationsgehalt in solchen Datensätzen, wohingegen der Benutzer normalerweise auf einen zweidimensionalen Bildschirm begrenzt ist. Daher sind Verfahren der visuellen Abstraktion erforderlich, um den Grad an Komplexität zu reduzieren und Artefakte, die durch Verdeckung oder visuelle Störungen verursacht werden, zu vermeiden. Dank der Fortschritte in der Entwicklung moderner Graphikhardware entstehen neue Möglichkeiten, um solche Methoden zu entwickeln.

Diese Arbeit leistet wesentliche Beiträge auf diesem Forschungsgebiet. Erstens stellen wir eine Methode vor, die interaktive visuelle Exploration von mehrdimensionalen Skalarfeldern ermöglicht. In einem Verfahren, bei dem schrittweise gestreut liegende Datenpunkte hinzugefügt werden, nähern wir Flächen mithilfe der Kriging-Interpolation an. Zu diesem Zweck schlagen wir ein GPU-beschleunigtes Aktualisierungsschema für die Kriging-Interpolation vor, um Flächen schrittweise konstruieren zu können. Daher stehen Ergebnisse schnell zur Verfügung, die den Benutzer in die Lage versetzen, relevante Abhängigkeiten zu entdecken und interaktiv zu steuern, wo neue Flächen erzeugt werden.

Zweitens behandeln wir die Visualisierung von 3D Skalarfeld-Ensembles. Ensembles werden in der Regel durch Ausführen von wiederholten Simulationen mit unterschiedlichen Parametereinstellungen erzeugt, wobei jede Ausführung ein einzelnes Element des Ensembles darstellt. Ensembles unter Verwendung traditioneller Visualisierungstechniken als multivariate Daten aufzufassen, ist für eine größere Anzahl von Elementen impraktikabel. Dennoch ist es wichtig, Unsicherheiten, Korrelationen und Trends in solchen Datensätzen aufzuzeigen. Hierfür präsentieren wir eine alternative Visualisierungstechnik, die eine abstrakte Ansicht basierend auf einer Linearisierung des 3D-Raums liefert, und sich durch die Verwendung von kombinierten Balken- und Liniendiagrammen auszeichnet.

Drittens stellen wir einen neuartigen Ansatz vor, um Ensembles aus 3D-Isoflächen basierend auf Silhouetten anstelle von vollständigen Oberflächen zu visualisieren. Somit werden Verdeckungseffekte deutlich vermindert, während räumliche Kohärenz und die eigentliche Form der Flächen erhalten bleiben. Interaktive Mechanismen, wie Selektion, Clusterbildung, Schnittflächen und Animation versetzen den Benutzer in der Lage, die Daten im Detail zu erkunden.

Viertens präsentieren wir ein neues Konzept, um die Grenzen von traditionellem Voxel-basierten Ray-Casting zu überwinden. Wir stellen eine hierarchische Closest-Point Octree Repräsentation vor, die die Approximationsqualität von Polygonflächen durch Voxelgitter verbessert. Wir zeigen, dass unser Ansatz im Vergleich zu einer herkömmlichen Voxel-Hierarchie zu einem deutlich geringeren Speicherverbrauch führt und Blockartefakte bei höheren Zoomstufen vermeidet. Das ermöglicht uns, Modelle von hohen Auflösungen in den GPU-Speicher zu laden, und qualitativ hochwertige Rendering-Ergebnisse bei interaktiven Bildraten zu erzielen.

Schließlich erstrecken wir dieses Konzept auf Ensembles von Formen, wobei wir das Closest-Point Gitter benutzen, um regionsweise zentrale Tendenzen zu quantifizieren und Mittel zur Verfügung stellen, um die zentralste Form zu finden. Darüber hinaus erzeugen wir eine jeweils lokal am besten passende Form, um dem Benutzer lokale Features zu kommunizieren. Unser Verfahren ist in der Lage, beliebige nichtparametrische Formen im zwei- und dreidimensionalen Raum zu verarbeiten, und stellt keinerlei Anforderungen an Abgeschlossenheit oder Orientierbarkeit.

Acknowledgments

I gratefully acknowledge the support of all of the people who made this thesis possible. First and foremost, I would like to thank my advisor, Prof. Dr. Rüdiger Westermann, for offering me the great opportunity to pursue research in the fields of uncertainty visualization and real-time rendering. I am truly grateful for his guidance, for his dedication to my work, and for the numerous motivating discussions.

I would also like to thank the co-authors of my papers, Dr. Christian Dick, Mihaela Jarema and Dr. Johannes Kehrer for supporting my work and helping me to obtain these results by contributing their suggestions and ideas. It was always a pleasure to work with them.

Furthermore, I would like to thank my current and former colleagues, Dr. Stefan Auer, Boris Bonev, Dr. Kai Bürger, Shunting Cao, Dr. Matthäus Chajdas, Rachel Chu, Dr. Christian Dick, Sebastian Eberhardt, Marie-Lena Eckert, Martin Ender, Florian Ferstl, Dr. Roland Fraedrich, Dr. Tiffany Inglis, Mihaela Jarema, Mathias Kanzler, Michael Kern, Dr. Andreas Klein, Alexander Kumpf, Henrik Masbruch, Dr. Tobias Pfaffelmoser, Vitaly Polisky, Dr. Marc Rautenhaus, Dr. Florian Reichl, Prof. Dr. Nils Thuerey, Dr. Marc Treib, Dr. Kiwon Um, Dr. Mikael Vaaraniemi, and Prof. Dr. Jun Wu, who have always been open for discussions.

I would like to thank my colleagues Florian Ferstl, Mihaela Jarema, Dr. Marc Rautenhaus, and furthermore, Dr. Torsten Enßlin and Dr. Jens Jasche from the Max-Planck-Institute for Astrophysics for providing data sets.

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

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

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Contribution . . . 2
1.3 Outline . . . 4
1.4 List of Publications . . . 5

2 Fundamentals **7**
2.1 Sources of Uncertainty . . . 7
2.1.1 Data Acquisition . . . 8
2.1.2 Data Transformation . . . 9
2.1.3 Visualization . . . 10
2.2 Classification of Uncertainty . . . 11
2.3 Mathematical Background . . . 12
2.3.1 Probability Space . . . 13
2.3.2 Random Variables . . . 14
2.3.3 Probability Density Functions . . . 15
2.3.4 Statistical Measurements for Uncertainty Quantification . . . 17
2.3.5 Multidimensional and Multivariate Data Representations . . . 22
2.3.6 Sample-Based Probability Analysis . . . 23
2.3.7 Numerical Methods . . . 25
2.4 Visualizing Uncertain Data Sets . . . 28
2.4.1 Data Dimensionality . . . 29
2.4.2 Uncertainty Dimension . . . 32
2.4.3 Approach . . . 34

2.4.4 Composition . . . 36
2.4.5 Context . . . 38
2.5 GPU Acceleration . . . 38
2.5.1 Background . . . 39
2.5.2 Real-Time Rendering . . . 40
2.5.3 GPGPU Computing . . . 42

3 Progressive Visualization of Multidimensional Scattered Data 45
3.1 Introduction . . . 45
3.2 Related Work . . . 48
3.3 Kriging Interpolation . . . 50
3.3.1 Main Principles . . . 51
3.3.2 Ordinary Kriging . . . 51
3.4 Minimizing Uncertainty . . . 53
3.4.1 Largest Empty Circle Algorithm . . . 54
3.4.2 Extension to More Than Two Dimensions . . . 55
3.5 Progressive GPU Kriging . . . 55
3.5.1 Computational Complexity . . . 56
3.5.2 CUDA Parallelization . . . 57
3.6 Visualizing Response Surfaces . . . 58
3.6.1 Response Surface Selection . . . 58
3.6.2 Slice-based Interpolation . . . 61
3.7 Results . . . 62
3.7.1 Multidimensional Real-World Data Set . . . 62
3.7.2 Performance Analysis . . . 63
3.8 Limitations . . . 65
3.9 Conclusion and Future Work . . . 66

4 Comparative Visualization of 3D Scalar Field Ensembles 69
4.1 Introduction . . . 69
4.2 Related Work . . . 71
4.3 Overview . . . 73
4.4 Multi-Charts . . . 75
4.4.1 Linearizing the 3D Grid . . . 75
4.4.2 Chart Layout . . . 76
4.5 Multiresolution Ensemble Summaries . . . 77
4.5.1 Analyzing the Quantitative Distribution of Uncertainty . . . 77
4.5.2 Analyzing Distribution and Relationship Among Members . . . 80

4.6 Brushing and Querying . . . 82
4.7 Spatial Clustering and Correlations . . . 85
4.7.1 Histogram Clustering . . . 85
4.7.2 Correlations . . . 87
4.8 Further Results . . . 90
4.8.1 Incompressible Fluid Flow Simulation . . . 90
4.8.2 Cosmic Density Field . . . 91
4.9 Conclusion and Evaluation . . . 93

5 Silhouette-Based Visualization of 3D Isosurface Ensembles 97
5.1 Introduction . . . 97
5.2 Related Work . . . 99
5.3 Preprocess . . . 100
5.3.1 Mesh-Generation . . . 100
5.3.2 Clustering . . . 101
5.4 Rendering Shaded Silhouettes . . . 102
5.4.1 Drawing Silhouettes in the Fragment Stage . . . 103
5.4.2 Density-Based Removal . . . 104
5.5 Visualizing Details on Demand . . . 105
5.5.1 Cutting Planes . . . 105
5.5.2 Clustering . . . 106
5.5.3 Picking and Brushing . . . 106
5.5.4 Animation . . . 107
5.6 Results . . . 107
5.7 Conclusion and Future Work . . . 108

6 Ray-Casting Based on Vector-to-Closest-Point Octrees 111
6.1 Introduction and Related Work . . . 111
6.2 VCP Representation . . . 114
6.2.1 Ray-Casting . . . 116
6.2.2 VCP Interpolation . . . 117
6.3 VCP Octree . . . 118
6.3.1 Mesh-based Generation . . . 119
6.4 GPU Implementation Issues . . . 121
6.4.1 VCP Leaf Data . . . 121
6.4.2 Reducing Run-time Memory Traffic . . . 122
6.4.3 Dynamic Memory Management . . . 123
6.4.4 GPU-CPU Upstreaming . . . 124

6.4.5 Interruption-Free Loading 125
6.5 Results 125
6.6 Conclusion and Future Work 126

7 Visualization of Shape-Based Ensembles 129
7.1 Introduction 129
7.2 Related Work 131
7.3 Method Overview 133
7.4 Generating the Vector-to-Closest-Point Ensemble 135
7.5 Modeling Vectors to Closest Points 135
7.5.1 Quantifying the Local Centrality 136
7.5.2 Finding the Median 139
7.5.3 Spatial Attributes 141
7.6 Clustering 142
7.7 Ensemble Visualization 143
7.7.1 Local Best Median 143
7.7.2 Fuzzy Regions 144
7.7.3 2D Ensembles 144
7.7.4 Color 144
7.7.5 Transparency 145
7.7.6 Silhouettes 145
7.8 Results 146
7.9 Conclusion and Future Work 148

8 Conclusion and Future Work 151

Bibliography 155

1 Introduction

"What is not surrounded by uncertainty cannot be the truth."

– Richard Feynman

1.1 Motivation

Advancements in new technology, research and science bring out an ever-growing supply of data that has the potential of broadening the human knowledge. Traditionally, scientific data are acquired from experimental measurements for the purpose of getting reliable information about unknown quantities. Due to the advances in modern computer hardware, running computer simulations has been proven as a means to efficiently generate large quantities of data based on underlying physical models. Often, computer simulations are used to predict prospective events based on measured data, e.g., in the disciplines of weather forecasting, risk management or traffic engineering. Simulations are also useful as a substitute for cases, where experimental measurements are too costly or impractical, for instance, in modeling car crashes such that the design of the vehicle protects the passengers or in simulating dam breaks.

However, grasping relevant information is very hard or even infeasible for human cognition, when only raw data are presented to the user or the domain expert. Obviously, the difficulty of this task increases further, when complexity and size of available data are growing due to technological advances. Therefore, the demand of sophisticated visualization techniques rises that enable the user to gain insight into vast quantities of data. Such techniques are typically based on graphical representations of scientific data sets, which exploit the human skills of visual perception. Their objective is to assist scientists in analyzing complex data sets, to recognize spatial or temporal structures and to extract characteristic features of particular relevance.

This thesis focuses in particular on multidimensional data sets, meaning that raw data values produced by the acquisition method live in multiple dimensions. Such data sets are encountered in various fields of research and engineering, e.g., in astrophysics and computational fluid dynamics, where three-dimensional distributions of density or particles are analyzed. Data sets of higher dimensions often come about in the context of sensitivity or regression analysis, where a target value depends on many independent variables. Yet visualizing multidimensional data is a challenging task, due to the inherent occlusion and attenuation effects as well as the loss of depth perception on a two-dimensional screen. Hence, developing advanced visualization techniques for multidimensional data sets, especially for more than two dimensions, is a vital area in research and a great amount of work is devoted to that purpose [WB97].

So far, we have touched on the need for visualization techniques, but we did not take the fact into account that real-world data can never be exact, as the opening quote by Richard Feynman makes clear. In fact, we encounter uncertainty everywhere in reality and the concept of error analysis is universally adopted in engineering, yet it remains a common practice in the area of visualization to simply assume uncertainty away [BOL12]. This is problematic, as it often gives the false illusion of precision or data quality that does not exist in reality. Moreover, due to the progress in numerical methods and the ever-increasing amount of computational power, information about uncertainty, such as probability distributions, variability data and confidence intervals, is covered by many of today's data sets. In the context of this thesis, ensemble simulation is of particular relevance, a process where repeated simulation runs are carried out for slightly modified parameter settings, thus yielding a distribution of values at each sample point. In recent years, the awareness of uncertainty in the field of visualization has grown and manifold techniques have been developed focusing particularly on that issue [PRJ12]. The purpose of these techniques is to aid experts in analyzing data sets, where the visualization conveys the underlying uncertainty faithfully to the user. Yet there still remain many unresolved issues, as integrating uncertainty into traditional visualization approaches can often not be achieved in a straightforward way. In particular, visual cluttering and occlusions effects are causes to this problem, which we address in this thesis.

1.2 Contribution

This thesis provides substantial contributions to the research domain of visualizing and analyzing uncertain multidimensional data. We propose several approaches predicated on displaying visual abstractions derived from given data sets. In particular, we provide means to explore higher dimensional scalar fields and ensembles of shapes and scalar fields in two or three dimensions. By utilizing GPU acceleration, our methods are computationally efficient and allow the user to discover data sets interactively. Our specific contributions are:

Progressive Response Surfaces. We present a novel method, which allows the user to interactively explorate scalar-valued data sets via high quality and progressive response surface prediction from multidimensional input samples in an efficient manner. We utilize Kriging interpolation to estimate a response surface that minimizes the expectation value and variance of the prediction error. To this end, we propose a progressive updating scheme for the Kriging weights, built upon incremental matrix inversion. Our method requires only minor computational effort when new sample points are added and high efficiency is achieved by employing parallelized operations on modern GPUs. This allows us to cope with even large data sets and give immediate feedback to the user, as single sample points are added continuously to the visualized response surfaces. In particular, this enables the user to monitor the surface, and thus provides means to cancel the computation early, if no more significant changes occur. By steering the generation process interactively to different points of interest, the user is able to quickly discern relevant dependencies. This method is presented in Chapter 3.

Multi-Charts. We propose bidirectional linking between extended bar charts (multi-charts) and volume visualization as a means to visually analyze three-dimensional scalar field ensembles. Multi-charts linearize 3D data points along a space-filling curve, which is then drawn to the same area of the screen. Statistical information on ensemble members is encoded into the bars and line charts are then overlaid to compare individual members to rest of the ensemble. Alternative linearizations based on statistical similarities allow for clustering of spatial locations based on the underlying data distribution. Data are arranged at different scales to quickly provide an overview and enable the user to interactively focus on regions of interest. By brushing or querying value intervals and specific distributions, the corresponding spatial points are simultaneously highlighted in the volumetric view. Additionally, we provide a picking mechanism in the volumetric view, where the corresponding locations are tagged in the multi-chart view, which allows the user can go back and forth between the abstract and the 3D view to focus the analysis. This technique is discussed in Chapter 4.

Screen-space Silhouettes for Isosurface Visualization. We introduce a novel visualization technique for ensembles of isosurfaces in three-dimensional space. Visualizing multiple isosurfaces at once is a challenging task due to inherent occlusion effects, yet analyzing uncertainty represented by such an ensemble is an important task. By using screen-space silhouettes, we reduce the displayed information, while maintaining the major shape of the surfaces. This approach retains spatial coherence without making any assumption about the underlying surface distribution. We provide means to interactively explore the ensemble on a finer level by additional mechanisms, namely, picking, clustering, cutting and animation. This approach is introduced in Chapter 5.

Vector-to-Closest-Point Octrees. We propose a novel improvement of voxel-based ray-casting to overcome the limitation of rendering block artifacts at high zoom levels. Traditional voxel-based techniques handle this problem by using very high resolutions, thus wasting a great amount of memory. By using a hierarchical vector-to-closest-point representation, we can inherit the advantages of a voxel-based approach and achieve a significantly smoother depiction of the surface. We show that, although the VCP grid consumes more memory per cell, it requires less memory overall, because it relies on a shallower tree hierarchy. We demonstrate the potential of our method for rendering high-quality surface models at interactive rates. This method is discussed in Chapter 6.

Visualizing the Centrality of Shapes. We propose a novel approach for analyzing the centrality, i.e., the central tendency, of an ensemble of shapes in two- and three-dimensional space based on the Vector-to-Closest-Point representation. This technique assists the user in determining the most central shape, to quantify the region-wise centrality, and to construct a locally best-matching shape. Unlike previous approaches, which build upon binary functions or signed distance fields relying on orientable shapes, we make use of the closest point representation. In doing so, our approach can handle arbitrary non-parametric shapes regardless of dimension and orientability. Directional distributions based on the closest point vectors are used to perform region-wise classifications. We demonstrate the effectiveness of our technique by visualizing various synthetic data sets as well as a real-world weather forecast ensemble. This technique is presented in Chapter 7.

1.3 Outline

The remainder of this thesis is organized as follows. In Chapter 2, we discuss fundamentals of visualizing and analyzing uncertain data. After presenting sources of uncertainty and a classification scheme, we introduce the mathematical background behind of probability theory in the context of uncertainty visualization. We then give an overview of popular visualization techniques and establish a scheme that allows us to categorize our contributions. We end this chapter with a discussion of GPU acceleration, since efficient rendering and computation methods serve as the basis of our visualization techniques. Next, in Chapter 3, we introduce our method for exploring multidimensional scalar-valued data by progressive response surface prediction. In Chapter 4, we present our technique for visualizing and analyzing 3D scalar field ensembles via multi-charts and volume visualization. Chapter 5 provides our approach for visualizing ensembles of 3D isosurfaces by rendering screen-space silhouettes. Then, in

Chapter 6, our method to improve voxel-based ray-casting is introduced, which avoids typical block artifacts by relying on a Vector-to-Closest-Point representation. Chapter 7 addresses the analysis of shape-based ensembles with regard to their central tendencies, by building a upon the Vector-to-Closest-Point representation. Finally, we conclude this thesis in Chapter 8 with a summary of our work and an outlook on future research directions.

1.4 List of Publications

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

1. Demir I., Westermann R.: Progressive High-Quality Response Surfaces for Visually Guided Sensitivity Analysis. *Computer Graphics Forum (Proceedings of EuroVis 2013) 32*, 3 (2013), 21–30. `doi:10.1111/cgf.12089`. [DW13].
2. Demir I., Dick C., Westermann R.: Multi-Charts for Comparative 3D Ensemble Visualization. *IEEE Transactions on Visualization and Computer Graphics 20*, 12 (Dec. 2014). `doi:10.1109/TVCG.2014.2346448`. [DDW14].
3. Demir I., Westermann R.: Vector-to-Closest-Point Octree for Surface Ray-Casting. In *Vision, Modeling & Visualization* (2015), Bommes D., Ritschel T., Schultz T., (Eds.), The Eurographics Association. `doi:10.2312/vmv.20151259`. [DW15].
4. Demir I., Kehrer J., Westermann R.: Screen-space Silhouettes for Visualizing Ensembles of 3D Isosurfaces. In *Proc. IEEE Pacific Visualization Symp. (Visualization Notes)* (2016). `doi:10.1109/PACIFICVIS.2016.7465271`. [DKW16].
5. Demir I., Jarema M., Westermann R.: Visualizing the Central Tendency of Ensembles of Shapes. In *Proc. SIGGRAPH Asia 2016 Symposium on Visualization* (2016). `doi:10.1145/3002151.3002165`. [DJW16].

2 Fundamentals

"You cannot be certain about uncertainty."

– Frank Knight

2.1 Sources of Uncertainty

Generally speaking, uncertainty can be defined as the lack of certainty that is the lack of reliability about information or knowledge. In the process of visualization, uncertainty is introduced at various stages. In Figure 2.1, the visualization pipeline based on the model by Haber and McNabb [HM90] is depicted. First, in the data acquisition stage, raw data are collected by means of measuring or simulating. Next, in the data transformation stage, the raw data is processed by filtering and mapping and thereby prepared for being presented to the user. The filtering step comprises picking a subset of the initial data set and interpolating missing values, usually in accordance with the user-selected focus. Optional transformations altering the data in other ways, like normalizing or offsetting are also performed in this step. Based on that output, geometry together with attributes is generated in the mapping step. Finally, by rendering geometric data in the visualization stage, a graphical representation is produced that is presented to the user. As shown in Figure 2.1, uncertainty emerges at every step. It is worth noting that uncertainty also propagates to subsequent steps. Visualization of uncertainty, i.e., uncertainty visualization, addresses the issue of showcasing the uncertainty included in the data set to the domain expert. In contrast, uncertainty of visualization occurs during the transformation and visualization stage and does not reflect imprecisions of the original data [BOL12]. Therefore, this kind of uncertainty is undesirable, yet it cannot be completely avoided, as we will see in the next sections.

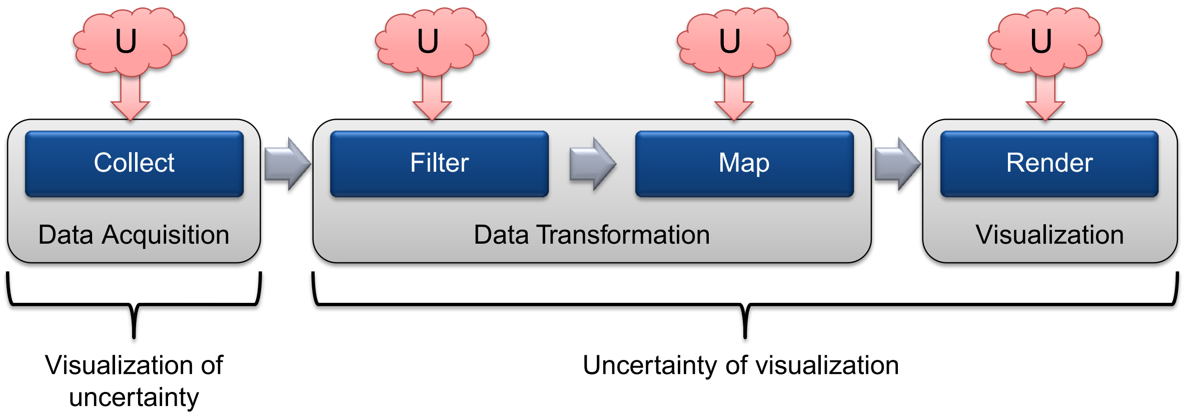


Figure 2.1: Visualization pipeline extended by uncertainty according to Pang et al. and Brodlie et al. [BOL12, PWL97], based on the model originally proposed by Haber and McNabb [HM90]. Uncertainty is introduced at every stage in the pipeline.

2.1.1 Data Acquisition

It is generally known that empirical data, i.e., data acquired through experiments or observations, can never be free from uncertainty [Cha83]. Consequently, data generated by simulations suffers from the same restriction, as both the input data and the underlying physical principles were ultimately derived from empirical measurements. According to Kennedy and O'Hagan [KO01], uncertainty accrues from various sources in the process of acquiring data:

- **Parameter uncertainty.** This kind of uncertainty represents the fact that values gathered by experimentation or observation, which are used as inputs to mathematical models, cannot be exact. An example is the viscosity of a fluid that has to be determined through measurements.
- **Model inadequacy.** Since no mathematical model can perfectly reflect the real world, there exists a discrepancy between the output of the simulation—assuming even input parameters of perfect certainty—and the result of the real process. Typical examples are simulations that (partially) ignore frictional forces.
- **Residual variability.** This defines a state of uncertainty due to unrecognized conditions. That is, even if the initial conditions are completely specified, the real-world process can produce diverging outcomes, as a consequence of two reasons. First, not all conditions relevant for the outcome might be known and second, true randomness might occur during the process.
- **Parametric variability.** When an experiment is performed, it is impossible to control the initial conditions with perfect certainty. Yet it is desirable to predict the outcome

of that experiment. Consequently, an additional extent of uncertainty must be included in the simulation's result, which is defined as parametric variability.

- **Observation error.** When comparing the real-world outcome of an experiment with the simulation's result, we must account for the fact that measuring the real outcome is prone to errors. Hence, uncertainty increases accordingly, although it should be noticed that this effect can be alleviated by repeatedly running the experiment.
- **Code uncertainty.** We cannot assume that computer simulations in practice produce results of perfect certainty, due to the inherent effects of uncertainty in every real-world scenario. Although considering such effects might be impractical, strictly speaking, they cannot be neglected.

2.1.2 Data Transformation

In the data transformation stage, data is processed by filtering and mapping. In both steps, uncertainty is introduced and propagated. During filtering, uncertainty occurs, because altering raw data has the potential of amplifying their extent of uncertainty [PWL97], and interpolating suffers from the inherent constraint that recovering unknown data points is always prone to errors. In particular, even if we can assume that initial data are given with perfect certainty, interpolated values at points not included in the data set must be considered as uncertain due to the inevitable interpolation error. Different strategies have been proposed to address this issue [BOL12]:

- We can iterate over all points in the initial data set, and use the information afflicted by uncertainty to compute a single value and estimate the extent of uncertainty. A commonly used example is the combination of mean (or expectancy value) and standard deviation. Interpolation is then carried out on these values by pretending that they are certain and the extent of uncertainty is propagated by a suitable interpolation method. Since in this process, a distribution of uncertain values is bundled into a single value, a new degree of uncertainty is introduced. In particular, when using mean and standard deviation, the newly introduced uncertainty grows the more, as the original values can be approximated by a Gaussian distribution to a lesser degree.
- Alternatively, we can replace the original values with a probability density function (PDF) and utilize an interpolation method based on PDFs. However, this procedure generates also uncertainty, as substituting discrete values by a continuous PDF is apparently an approximation prone to errors.

- Suppose, we have knowledge about the underlying physical model of the data set. In this case, we can design an interpolation scheme that incorporates the parametrization of this model. Thus, the interpolation error can be reduced to a certain extent. For instance, consider the case, where we know that our model can be described by a quadratic term. Here, we can expect better outcomes by using spline instead of linear interpolation.

The mapping step can also induce uncertainty, since geometry is usually restricted by certain specifications into which the filtered data must fit. As a typical example, consider triangle meshes that are often used to render 3D images. Here, fitting smooth surfaces causes uncertainty at every non-vertex point of the mesh. Hence, it is crucial to ensure that the geometry exhibits a resolution fine enough to avoid this unwanted effect to the best degree possible in accordance with hardware capabilities.

2.1.3 Visualization

Another source of uncertainty appears in the visualization stage, where the geometric data are rendered to an image and presented to the user. As it distracts the expert from the uncertainty intrinsic in the data to be analyzed, again, we consider this an undesired source of uncertainty and aim at minimizing such effects. A brief overview on how this kind of uncertainty appears is presented below, although this list should not be considered exhaustive.

- In the rendering step, geometry has to be discretized in order to be drawn to a pixel raster. This process, i.e., rasterization, causes some extent of uncertainty, especially when the image resolution is too low. To remedy this problem, interactive focus and zooming techniques integrated into the visualization process can be utilized [BOL12].
- When rendering three-dimensional objects, shading and lighting techniques are used to simulate their characteristics and various effects. In this procedure, another layer of uncertainty is introduced caused by rendering algorithms, which involve inevitably a trade-off between quality and framerate, and computational precision. Consequently, by using different algorithms, striving for different framerates, and resorting to different hardware, distinct images can be produced. In this situation, each image can possibly have a different meaning to the user [PWL97].
- In many cases, rendering the output image depends on a vast number of parameters that are either selected automatically or by user intervention. In any case, these parameters can influence the outcome in a way not obvious to the expert. For instance, consider an application visualizing the behavior of a fluid by drawing streamlines. These streamlines have to be seeded at different domain points and slightly different starting points can

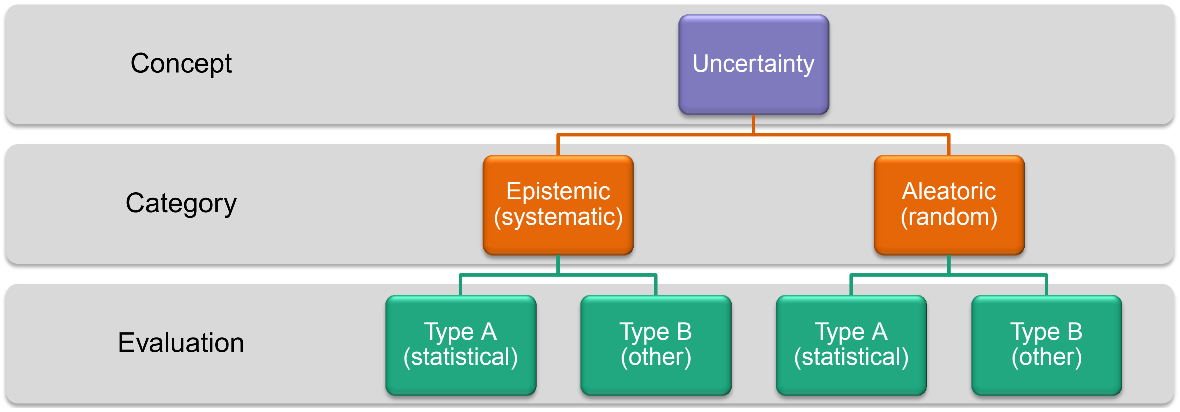


Figure 2.2: Classification of uncertainty according to the NIST guidelines on evaluating and expressing uncertainty of measurements [TK09].

yield significantly diverging results. Likewise, bringing in animation to show time-dependent features constitutes a source of uncertainty. Here, the key is that some sort of interpolation has to be used to compensate for different time spans between given data points on the one hand and the framerate achieved by the hardware on the other hand [PWL97].

- Lastly, uncertainty is involved at the level of the user's perception, as the same image can have other meanings to different users. Such effects can be caused by the user's visual and mental abilities or his scientific or cultural background [BHJ*14]. As a typical example of this situation, consider the differences of human's cognition and interpretation of colors [Lev97].

In conclusion, uncertainty emerges at all stages of the visualization pipeline, and it is essential to be aware of these effects and to handle them in a proper fashion in order to be faithful to reality when presenting data to the user.

2.2 Classification of Uncertainty

After discussing the sources of uncertainty, we will now focus on understanding how uncertain data sets are classified. Figure 2.2 provides an overview on how uncertainty is classified according to the NIST guidelines [TK09]. Uncertainty is commonly distinguished into two categories, epistemic and aleatoric uncertainty [PRJ12, DKD09].

- **Epistemic uncertainty** refers to a systematic lack of information that could be known in principle but remains unknown in practice. Low measurement accuracy, inadequate models and purposefully hidden data are the causes of this kind of uncertainty. Among other things, scientific or economic reasons can be behind that, e.g., when better measurement instruments are too costly or when the latest models are lacking certain physical properties. As an example of this, consider a reference device used for calibration, which itself exhibits a certain degree of uncertainty.

- **Aleatoric uncertainty** can be discovered by statistical means, that is, by running an experiment multiple times and witnessing slightly different outcomes. It contains the randomness inherent to the experiment and thus cannot be diminished by using better measurement equipment or models. The unavoidable discrepancy between the outcomes of repeated measurements is an example for this category.

In line with the NIST report, evaluating uncertainty, i.e., estimating its numerical value, can be classified into two methodologies [TK09].

- **Type A** refers to uncertainty evaluated by statistical methods. By using probability distributions, we can characterize this sort of uncertainty and then design visualization schemes built upon that information. In particular, mean and standard deviation are commonly used to address this issue, although this approach is problematic, if the distribution cannot assumed to be Gaussian (see Section 2.3.4).

- **Type B** refers to uncertainty evaluated by other means. Scientific judgment, experience and general knowledge about properties of materials and instruments are typical methods to estimate this quantity.

Uncertainty visualization mainly focuses on aleatoric uncertainties of type A [PRJ12]. By visually conveying data in combination with their incorporated uncertainty to the user, it ultimately aims at supporting the experts in analyzing and decision making [PWL97].

2.3 Mathematical Background

At the beginning of this chapter, a qualitative definition of uncertainty was presented. However, for embedding uncertainty information into tools for visualization, quantitative measurements are necessary. To this end, we rely on probability theory—the branch of mathematics addressing the study of random phenomena. In the following, we will describe, how uncertainty can be expressed by mathematical models in particular in the context of visualization.

2.3.1 Probability Space

The probability space is a fundamental construct in probability theory. It is a mathematical construct modeling a random experiment that consists of three components. We will first introduce each component and then come back to the definition of a probability space [Fri97]. As a running example, let us consider an experiment where two (perfectly fair or unbiased) dice are thrown.

The *sample space* Ω is a set of all possible outcomes from a random experiment. In our example, it is given as

$$\Omega = \{1, 2, 3, 4, 5, 6\}^2 . \tag{2.3.1}$$

The *σ-algebra* $\mathcal{F} \subset 2^\Omega$ is a set of all events of interest such that the conditions below are met. An *event* is a subset of Ω that *occurs* if the outcome of the experiment is an element of the event.

- $\mathcal{F}$ contains the empty set:
$$\emptyset \in \mathcal{F} \tag{2.3.2}$$

- $\mathcal{F}$ is closed under complementation:
$$A \in \mathcal{F} \Rightarrow \Omega \setminus A \in \mathcal{F}. \tag{2.3.3}$$

- $\mathcal{F}$ is closed under countable unions:
$$A_i \in \mathcal{F}\,(i \in \mathbb{N}) \Rightarrow \bigcup_{i=1}^{\infty} A_i \in \mathcal{F}. \tag{2.3.4}$$

Notice that, due to Eq. (2.3.2) and Eq. (2.3.3), $\mathcal{F}$ also contains the sample space Ω. Moreover, Eq. (2.3.3) and Eq. (2.3.4) imply that $\mathcal{F}$ is also closed under countable intersections:

$$A_i \in \mathcal{F}\,(i \in \mathbb{N}) \Rightarrow \bigcap_{i=1}^{\infty} A_i \in \mathcal{F}. \tag{2.3.5}$$

We now give some justifications for these restrictions. First, we want to be able to speak of no event occurring, Eq. (2.3.2). Second, if we can speak of one event occurring, we also want to speak of this event not occurring, Eq. (2.3.3). Lastly, we would like to speak of at least one of multiple individual events occurring, Eq. (2.3.4).

In our running example, let us consider each possible event, namely $\mathcal{F} = 2^\Omega$.

The *probability measure* on $\mathcal{F}$ is a function $P : \mathcal{F} \to [0,1]$ such that $P(\Omega) = 1$ and

$$P\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} P(A_i) \tag{2.3.6}$$

for each pairwise disjoint sequence $A_i \in \mathcal{F}\,(i \in \mathbb{N})$. Because of the latter condition, P is said to be countable additive.

In our scenario, we have

$$P(\omega) = \frac{1}{36} \text{ for all } \omega \in \Omega. \tag{2.3.7}$$

We now define the *probability space* as the triple $(\Omega, \mathcal{F}, P)$.

Note that often Ω is a topological space, e.g., $\Omega = \mathbb{R}$. In this case, it is implicitly assumed that $\mathcal{F}$ is generated by the collection of open sets in Ω. $\mathcal{F}$ is then called a *Borel algebra*.

2.3.2 Random Variables

In real applications, random variables are typically used to represent the actual observation [Fri97]. We begin with some basic definitions.

A tuple $(\Psi, \mathcal{G})$ is a *measurable space*, if $\Psi \neq \emptyset$ and $\mathcal{G}$ is a σ-algebra on Ψ.

Let $(\Omega, \mathcal{F})$ and $(\Psi, \mathcal{G})$ be measurable spaces. A function $X : \Omega \to \Psi$ is said to be *measurable*, if $X^{-1}(B) \in \mathcal{F}$ for every $B \in \mathcal{G}$. That is, the preimage of every $B \in \mathcal{G}$ under X is in $\mathcal{F}$. When $(\Omega, \mathcal{F}, P)$ is a probability space, then X is called a *random variable*.

In our running example, let us consider the sum of both dice. The corresponding random variable is given as

$$X : \Omega \to \mathbb{R}, \quad (\omega_1, \omega_2) \to \omega_1 + \omega_2. \tag{2.3.8}$$

Proposition 1 *Let $(\Omega, \mathcal{F}, P)$ be a probability space, $(\Psi, \mathcal{G})$ a measurable space and $X : \Omega \to \Psi$ a random variable. Let $Q : \mathcal{G} \to [0,1], B \to P(X^{-1}(B))$. Then $(\Psi, \mathcal{G}, Q)$ is a probability space. Q is called the* (probability) distribution *that is* induced *by X.*

Proof. See [Fri97]. □

We denote by $\mathcal{P}(c(X))$ the probability measure of the set of outcomes, or *probability* in short, where X fulfills a certain condition $c(X)$:

$$\mathcal{P}(c(X)) = P(\{\omega \in \Omega : c(X(\omega))\}). \tag{2.3.9}$$

For instance, the probability of the dice sum being equal to 3 can be stated as $\mathcal{P}(X = 3)$. To compute this probability, we use the preceding proposition and Eq. (2.3.6):

$$\begin{aligned}\mathcal{P}(X = 3) &= P(\{\omega \in \Omega : X(\omega) = 3\})\\ &= P(\{(1,2),(2,1)\})\\ &= P(\{(1,2)\}) + P(\{(2,1)\})\\ &= \frac{1}{36} + \frac{1}{36} = \frac{1}{18}\end{aligned} \tag{2.3.10}$$

2.3.3 Probability Density Functions

We now introduce the concept of probability density functions (PDFs). In many real-world scenarios, outcomes of a random experiment are continuously distributed, i.e., $\Omega = \mathbb{R}$. Moreover, they do not take values that can be exactly predicted. As an example, suppose we are interested in the temperature X at a given point in time and space. The probability, for instance, that it will be exactly 0 °C is equal to 0, i.e., $\mathcal{P}(X = 0) = 0$. Rather, we have to ask for the probability that the temperature lies between, e.g., −0.5 °C and 0.5 °C to obtain a useful information, i.e., $\mathcal{P}(-0.5 \le X \le 0.5)$. We begin with a basic definition [Fri97].

A function $F : \mathbb{R} \to \mathbb{R}$ is called a *distribution function* if it is increasing and right-continuous and satisfies the condition

$$\lim_{x\to-\infty} F(x) = 0 \text{ and } \lim_{x\to\infty} F(x) = 1. \tag{2.3.11}$$

Proposition 2 *Let $(\mathbb{R}, \mathcal{B}, P)$ be a probability space. Note that $\mathcal{B}$ denotes the Borel algebra on $\mathbb{R}$. Then, the function $F(x) = P((-\infty, x])$ is a distribution function.*

Proof. See [Fri97]. □

In our example, where two dice are rolled, the distribution function is given by $F(x) = \mathcal{P}(X \le x)$. A jump discontinuity occurs at every integer in the range from 2 to 12. Hence, it is not continuous (but still right-continuous). For instance, we have

$$F(2) = \frac{1}{36},\ F(3) = \frac{1}{36} + \frac{1}{18} = \frac{1}{12}, \ldots,\ F(12) = 1. \tag{2.3.12}$$

Note that every distribution function can have at most countably many jump discontinuities and it is continuous if and only if $P(\{x\}) = 0$ for every $x \in \mathbb{R}$. Moreover, a probability measure can also be constructed based on a distribution function as the following proposition shows.

Proposition 3 *Let $F : \mathbb{R} \to \mathbb{R}$ be a distribution function. Then, $P\left((-\infty, x]\right) = F(x)$ is a probability measure on $(\mathbb{R}, \mathcal{B})$.*

Proof. See [Fri97]. □

Given a distribution function F such that for a random variable X, it holds $F(x) = \mathcal{P}(X \leq x)$, we can now compute the probability for X being within the interval $(a, b]$ as

$$\mathcal{P}(a < X \leq b) = F(b) - F(a). \tag{2.3.13}$$

Let F be a distribution function that can be characterized by

$$F(x) = \int\limits_{-\infty}^{x} f(t)\, dt. \tag{2.3.14}$$

Then we call f the *probability density function (PDF)* of F, or *density* in short.

Consider the following remarks.

- If a PDF $f(x)$ exists, the distribution function is continuous. For a corresponding random variable X, the probability for being within the interval $(a, b]$ can then be computed as
$$\mathcal{P}(a \leq X \leq b) = \int\limits_{a}^{b} f(t)\, dt. \tag{2.3.15}$$
Hence, no PDF exists for the dice-throwing example.

- If a PDF $f(x)$ is continuous at x, then we have
$$f(x) = \frac{d}{dx} F(X). \tag{2.3.16}$$

A frequently used distribution in many applications is the Gaussian distribution (or normal distribution) [Cas02, Pat96]. Its PDF is given by

$$f_{\mathcal{N}(\mu,\sigma^2)}(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \tag{2.3.17}$$

The parameters μ and σ^2 specify mean and variance (see Section 2.3.4). For the Gaussian distribution, this is denoted by $\mathcal{N}(\mu, \sigma^2)$. In Figure 2.3, the PDF and distribution function are shown for two parameter settings.

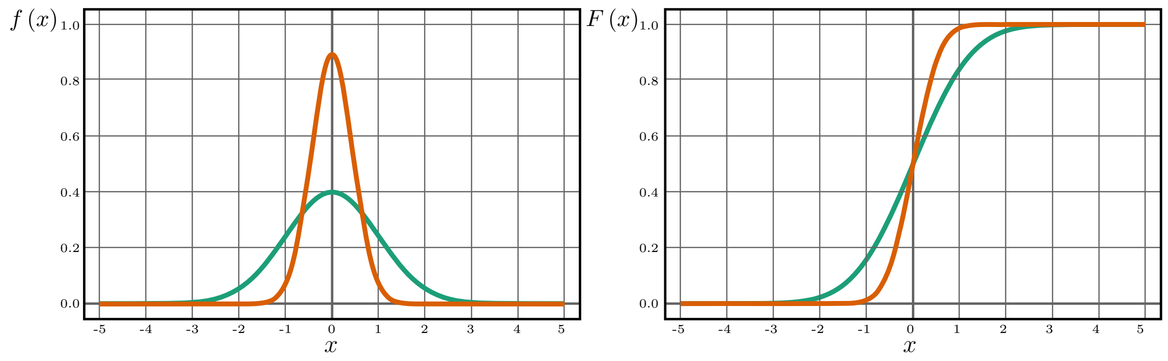


Figure 2.3: The probability density function (left) and the distribution function (right) of two Gaussian distributions with parameters $\mathcal{N}(0,1)$ and $\mathcal{N}(0,0.2)$. These density functions are also referred to as Gaussian bell curves.

The PDF of the normal distribution has a peak at μ and the probability of the outcome being at most one standard deviation, i.e., σ, apart from the mean is roughly 68 %. As an example, let us consider a temperature forecast (for a given point in time and space). Suppose, it can be described by the Gaussian distribution $\mathcal{N}(0,1)$ (see Figure 2.3). To compute the probability that the temperature, X, will be between $-0.5\,°\mathrm{C}$ and $0.5\,°\mathrm{C}$, we proceed as follows.

$$\begin{aligned} \mathcal{P}(-0.5 \leq X \leq 0.5) &= \int\limits_{-0.5}^{0.5} f_{\mathcal{N}(0,1)}(t)\,dt \\ &= \int\limits_{-0.5}^{0.5} \frac{1}{\sqrt{2\pi \cdot 1}}\, e^{-\frac{(t-0)^2}{2\cdot 1}}\,dt \quad \approx 0.3829 \end{aligned} \tag{2.3.18}$$

Note that the last step can only be evaluated by numerical methods [HJ61], which is beyond the scope of this thesis.

2.3.4 Statistical Measurements for Uncertainty Quantification

We now address some statistical measurements that are commonly used in the area of uncertainty visualization. To begin with, we introduce the terms mean, variance and standard deviation [Cas02, Fri97].

Mean

Let X be random variable defined on a probability space $(\Omega, \mathcal{F}, P)$. Then the *mean*, or *expected value*, is defined as the Lebesgue integral

$$\mathbb{E}[X] = \int\limits_{\Omega} X \, dP. \tag{2.3.19}$$

We now consider two special cases, where the mean can be defined in more concrete way.

Let X be discrete random variable defined on a probability space $(\Omega, \mathcal{F}, P)$ such that it only takes the values x_i with probability p_i, $i \in \mathbb{N}$. Then, the mean is given by

$$\mathbb{E}[X] = \sum_{i=1}^{\infty} x_i \cdot p_i = \sum_{i=1}^{\infty} x_i \cdot \mathcal{P}\left(X = x_i\right). \tag{2.3.20}$$

Often, the mean is denoted by μ. Getting back to the dice-throwing example, we can compute the mean as

$$\begin{aligned}\mathbb{E}[X] &= \sum_{x=2}^{12} x \cdot \mathcal{P}\left(X = x\right) \\ &= 2 \cdot \frac{1}{36} + 3 \cdot \frac{1}{18} + \cdots + 12 \cdot \frac{1}{36} \quad = 7\end{aligned} \tag{2.3.21}$$

Let X be random variable defined on a probability space $(\Omega, \mathcal{F}, P)$ such that a PDF $f(x)$ on X exists. Then, the mean is given by integrating

$$\mathbb{E}[X] = \int\limits_{-\infty}^{\infty} x \cdot f(x) \, dx. \tag{2.3.22}$$

For instance, for the Gaussian distribution $\mathcal{N}\left(\mu, \sigma^2\right)$, we have $\mathbb{E}[X] = \mu$ [Pat96].

Intuitively, the expected value is the average outcome of a long series of repeated random experiments. According to the law of large numbers, the probability of the average converging to the expected value is 1 as the number of experiments grows to infinity. Moreover, it can be shown that the mean is linear and monotonic. Let X and Y be random variables and $a, b, c \in \mathbb{R}$. Then:

$$\mathbb{E}[aX + bY + c] = a\mathbb{E}[X] + b\mathbb{E}[Y] + c \tag{2.3.23}$$

$$\mathcal{P}\left(X \leq Y\right) = 1 \Rightarrow \mathbb{E}[X] \leq \mathbb{E}[Y] \tag{2.3.24}$$

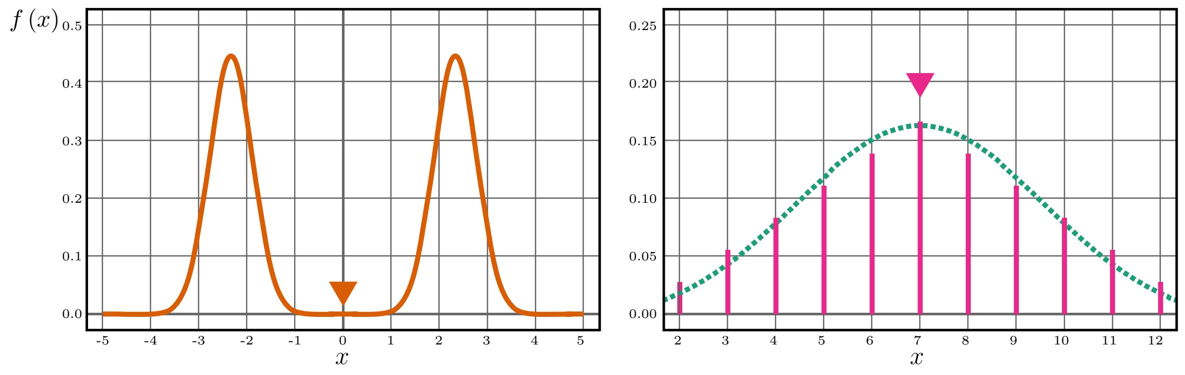


Figure 2.4: Left: The PDF of a bimodal probability distribution. Clearly, the mean—marked by an arrow—misses the two peaks and instead takes a value where almost no outcomes would occur. Right: The probabilities of the dice throwing experiment. Here, the mean points to a useful value and fitting a Gaussian bell curve to that distribution yields as a good approximation (green dotted line).

In many applications of visualization, the mean is used to represent a probability distribution. As we have already mentioned before, this approach is problematic, if the distribution cannot approximately assumed to be Gaussian. This is demonstrated in Figure 2.4 (left). Here, a bimodal probability distribution is given, i.e., a distribution with two peaks (or modes). As an example for such a distribution, consider the traffic density, where peaks often occur during the morning and evening rush hours. Here, visualizing the mean would give no useful information to the user, as it takes a value between the two peaks where almost no outcomes occur.

On the other hand, consider Figure 2.4 (right). Here, the probabilities of the dice throwing experiment are plotted. In this scenario, the mean conveys useful information, since, as a matter of fact, most outcomes of that experiment are close to the mean value of 7. Moreover, this distribution can be approximated by a bell curve to a good degree, which also indicates that the mean is a good representative.

Variance and Standard Deviation

Let X be random variable defined on a probability space $(\Omega, \mathcal{F}, P)$. Then the *variance* is defined as the expected squared deviation from the mean:

$$\mathrm{Var}[X] = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] \tag{2.3.25}$$

Note that this can also be stated as

$$\text{Var}[X] = \mathbb{E}\left[X^2\right] - (\mathbb{E}[X])^2 \tag{2.3.26}$$

The variance is also denoted by σ^2. It is a statistical measurement of the spread between the outcomes of a random experiment. Hence, it is always a non-negative number. In our running example, where two dice are rolled, we can compute the variance as

$$\begin{aligned}\text{Var}[X] &= \sum_{x=2}^{12} (x - \mathbb{E}[X])^2 \cdot \mathcal{P}(X = x) \\ &= (2-7)^2 \cdot \frac{1}{36} + (3-7)^2 \cdot \frac{1}{18} + \cdots + (12-7)^2 \cdot \frac{1}{36} \\ &\approx 5.83\end{aligned} \tag{2.3.27}$$

For the Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, we have $Var[X] = \sigma^2$ [Pat96].

The standard deviation σ is defined as the square root of the variance, i.e.,

$$\sigma = \sqrt{\text{Var}[X]}. \tag{2.3.28}$$

For the Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, the standard deviation is σ and—as mentioned earlier—there is a probability of $\approx 68\,\%$ that an outcome falls within the scope of one standard deviation from the mean. In our example, the standard deviation is $\sigma \approx \sqrt{5.83} \approx 2.41$. Due to the approximation by the bell curve, as shown in Figure 2.4 (right), we can estimate that the probability of the dice sum being within 5 and 9—one standard deviation apart from the mean—equals roughly 68 %. This is a good approximation of the exact probability of $^2/_3$.

Similar to the variance, the standard deviation quantifies the variation. An advantage of using the standard deviation is the fact that it is expressed in terms of the same units as the random variable. Hence, for the user it is usually a more convenient way of characterizing the distribution.

In many applications, the standard deviation serves as a quantification of uncertainty [TK09]. Moreover, in the field of visualization, a combination of mean and standard deviation is often used to convey uncertain data sets to the user, as directly showing the PDF is—except for the 1D case—usually too complex [BHJ*14].

However, the standard deviation is subject to a similar restriction as the mean. Intuitively, the reason behind this is that probability distributions more complex than the Gaussian distribution contain information that cannot be adequately expressed by two scalar values (mean and standard deviation). As a typical example in the context of simulation, consider outliers, i.e., values significantly different from the mean, but produced only by very few simulation runs. Taking outliers into account is of eminent importance in the field of risk management, as rare events of high impact can constitute high risks. For instance, suppose most simulation runs of a weather forecast system predict a moderate amount of snowfall, while a blizzard is projected by few exceptions (outliers). Visualizing mean and standard deviation would conceal that risk leading to a false illusion of certainty. Hence, using more sophisticated visualization techniques is of paramount importance.

Other Quantities

Other statistical quantities that are of importance within the scope of this thesis, include the covariance, the median and quantiles. Intuitively, the covariance specifies to which extent two random variables change together. Formally, it is defined as follows.

Let X, Y be random variables on the probability space $(\Omega, \mathcal{F}, P)$ such that $\mathbb{E}[X], \mathbb{E}[Y]$ are finite. Then the *covariance* is defined as

$$\operatorname{Cov}(X, Y) = \mathbb{E}\left[\left(X - \mathbb{E}[X]\right)\left(Y - \mathbb{E}[Y]\right)\right]. \tag{2.3.29}$$

Notice that $\operatorname{Cov}(X, X) = \operatorname{Var}[X]$.

Let $F(x)$ be a distribution function. Then the *quantile function* is defined as [Par79]

$$Q(u) = \inf\left\{x : F(x) \leq u\right\}. \tag{2.3.30}$$

Hence, for a random variable $X : \Omega \to \mathbb{R}$ with distribution function $F(x)$, the quantile $Q(u)$ specifies the value such that

$$\mathcal{P}\left(X \leq Q(u)\right) \geq u. \tag{2.3.31}$$

For instance, when rolling two dice, we have $Q(0.9) = 10$. Thus, the probability that the sum of the dice will be less than or equal to 10 is at least 90 %.

Notice that the *median* is defined as the quantile $Q(^1/_2)$.

2.3.5 Multidimensional and Multivariate Data Representations

Within the scope of this thesis, multidimensional and multivariate data representations are of particular importance. As an example, consider a data set, where the fuel efficiency—i.e., the dependent variable—of cars was measured. For each car, various parameters are known, like horsepower, weight, cylinders, acceleration—i.e., the independent parameters. Such a data set is multidimensional, because its outcomes depend on multiple input parameters, each spanning one dimension. This gives rise to the following definition [Fri97, Van10].

Let $(\Omega, \mathcal{F}, P)$ be a probability space, $(\Psi, \mathcal{G})$ be measurable space, and T be a topological space. Let

$$\boldsymbol{X} : T \to (\Omega \to \Psi), t \to X(t) \tag{2.3.32}$$

be a collection of random variables. Then $\boldsymbol{X}$ is called a *random field*. If $T = \mathbb{R}^n$ for $n \in \mathbb{N}$, then $\boldsymbol{X}$ is said to be *n-dimensional*.

For instance, we can denote the expected fuel efficiency of a car with parameter settings $t = (t_1, t_2, \dots)$ for horsepower, weight, etc., by $\mathbb{E}[\boldsymbol{X}(t)]$.

Next, consider a weather forecast simulation that predicts the (vector-valued) wind velocity at a given point in time and space. Here, a random variable that models this scenario is multivariate, because it consists of multiple individual variables, namely the x, y, z-components of the wind velocity. This gives rise to the following definition.

Let X be a random variable defined on the probability space $(\Omega, \mathcal{F}, P)$ such that

$$X : \Omega \to \mathbb{R}^n \text{ for } n \in \mathbb{N} \tag{2.3.33}$$

Then X is said to be *multivariate with n components*.

Of course, these two properties can be combined to model multidimensional multivariate data sets. As an example, suppose, a simulation predicts the wind velocity over a geographical region $U \subset \mathbb{R}^3$ and within a time span $T \subset \mathbb{R}$. Then, the collection

$$\boldsymbol{X} : (U, T) \to \left(\Omega \to \mathbb{R}^3\right), (u, \tau) \to X(u, \tau) \tag{2.3.34}$$

represents the according random field. By $\mathbb{E}[\boldsymbol{X}(u, \tau)]$ we can denote the expected wind velocity at point u and time τ. Note that Ω corresponds to the (randomized) input parameters given to the simulation.

Although the terms multidimensional and multivariate have distinct meanings, they are often used vaguely in the literature [WB97]. To avoid cumbersome expressions like "multidimensional multivariate data set", we usually use the term of greater relevance in the respective context.

Visualizing multidimensional or multivariate data is a great challenge, as much information has to be conveyed to the expert in a way that enables him to gain the decisive insight without being distracted by subsidiary details. This holds in particular, when an additional dimension of uncertainty is introduced.

2.3.6 Sample-Based Probability Analysis

Usually, when visualizing uncertain data sets, underlying raw data are not available in the form of probability density functions. The reason behind this is the fact that measurements occur at discrete positions in time and space. Likewise, simulations produce discrete outcomes for discrete settings of input parameters. Consequently, a finite number of discrete values is given and by using an appropriate visualization tool, the data are presented to the domain expert. The difficulty in this endeavor is to extract statistical information without direct knowledge about the underlying distribution. To begin with, we introduce some elementary definitions [Cas02, Fri97].

Let $(\Omega, \mathcal{F}, P)$ be a probability space, $(\Psi_k, \mathcal{G}_k)_{(k=1,2)}$ be a collection of measurable spaces and $X : \Omega \to \Psi_1$, $Y : \Omega \to \Psi_2$ be random variables. Then, X and Y are said to be *independent* if

$$P\left(X^{-1}(B_1) \cap Y^{-1}(B_2)\right) = P\left(X^{-1}(B_1)\right) P\left(Y^{-1}(B_2)\right) \text{ for all } B_1 \in \mathcal{G}_1, B_2 \in \mathcal{G}_2. \quad (2.3.35)$$

Defined by induction, a set of at least 3 random variables is said to be *(mutually) independent*, if any proper finite subset is independent.

Intuitively, this is the case, if the outcome of any random variable has no effect on the probability distributions of the rest.

Let $n \in \mathbb{N}$, $F : \mathbb{R} \to \mathbb{R}$ be a distribution function and X_k be a collection of independent real-valued random variables such that $F(x) = \mathcal{P}(X_k \leq x)$, $k = 1, \ldots, n$. Then, X_k is called a *random sample of length* n.

Notice that this definition can be extended in a straightforward way to samples of random fields and multivariate random samples. For random fields over a topological space T, consider a collection of distribution functions $T \to (\mathbb{R} \to \mathbb{R})$, $t \to F_t$. Now, we can define a *sample of random fields of length* $n \in \mathbb{N}$ as a collection of random variables such that $\boldsymbol{X}_k = X_{k,t}$ $(k = 1, \ldots, n,\ t \in T)$. With regard to multivariate samples of $m \in \mathbb{N}$ components, let us consider a collection of distributions, $\{1, \ldots, m\} \rightarrow (\mathbb{R} \rightarrow \mathbb{R}), l \rightarrow F_l$. Now, we can define a *multivariate random sample of length $n \in \mathbb{N}$ and m components* as a collection of random variables $X_{k,l}$ $(k = 1, \ldots, n,\ l = 1, \ldots, m)$. Of course, both properties can also be combined, although we omit the formal definition.

Given a random sample X_k, $(k = 1, \ldots, n)$ on a probability space $(\Omega, \mathcal{F}, P)$, the realizations of each random variable, that is $X_k(A)$ for a single event $A \in \mathcal{F}$, are called *sample points* or *observations*.

We now look upon two applications that are of particular importance throughout this thesis.

Ensembles

An ensemble consists of a number of possible states of a real system [Gib02]. Let $(\Omega, \mathcal{F}, P)$ be a probability space and T be a topological space. Let $U \subset T$ be a (usually finite) subset of points and $\boldsymbol{X}_k = X_{k,u}$ $(k = 1, \ldots, n,\ u \in U,\ n \in \mathbb{N})$ be a sample of random fields. Moreover, let $A_1, \ldots, A_n \in \mathcal{F}$ be events. Then, each *state* or *member*, indexed by $k \in \{1, \ldots, n\}$, comprises the sample points $X_{k,u}(A_k)$ $(u \in U)$ belonging to a sample of random fields. The collection of all states is called the *ensemble*. Notice that this definition can be directly extended to multivariate ensembles. Again, we refrain from giving the technical definition.

Based on the sample points, the objective is to recover a probability distribution function $f_u : \Omega \rightarrow \mathbb{R}$, or some statistical quantities like mean or standard deviation at every point $u \in U$. When $T = \mathbb{R}^d$ $(d \in \mathbb{N})$, and U is resolved uniformly on a Cartesian grid, the points $u \in U$ are called *grid points*. Moreover, if we are only interested in describing the raw data set, we can use the convenient form $\{1, \ldots, n\} \times U \rightarrow \mathbb{R}$. Ultimately, the goal of ensemble-based visualization techniques is to visually convey the recovered quantities to the user. Ideally, this is done in a way that lets the expert focus on the relevant features.

When an ensemble is produced by running multiple simulations, the events A_k $(k = 1, \ldots, n)$ constitute the *input parameters* and the random variables $X_{k,u}$ represent the deterministic algorithms used for performing the simulation. Note that non-deterministic algorithms can always be rewritten into a deterministic form by shifting the random elements to the input parameters, i.e., by extending the sample space accordingly.

Ensemble visualization is discussed in Chapters 4, 5, 7. A data set that is frequently used throughout this thesis comprises the results of a weather forecast simulation. In this example, different meteorological quantities like temperature, humidity, wind velocity, are predicted. Each ensemble member contains the outcome of a different simulation run at grid points over a certain geographical area.

Scattered Data

Formally, a *scattered data set* is an ensemble consisting of 1 member at scattered points $u \in U \subset T$ within a topological space T. By scattered, we mean that points are not necessarily aligned at a regular grid. However, here we pursue a different objective: By using interpolation or approximation techniques, we generate a random field $\boldsymbol{X} : T \ni t \to X(t)$ subject to minimizing the difference between the expected outcomes and the real-world values.

When visualizing the result, it is important to present the predicted values together with a measure of uncertainty, e.g., the standard deviation. Otherwise, we would induce a false sense of certainty about the reconstructed values. For instance, we would expect a greater degree of certainty in the vicinity of the measured values. To generate such an outcome, we can resort to the distance metric associated with T, provided that T is a metric space.

Visualizing scattered data sets is discussed in Chapter 3. A previously mentioned example is the auto data set, where fuel efficiency was measured together with certain criteria of the car. Visualizing this data set allows the expert to analyze the influence of such criteria on the outcome and ideally to discover possibilities of optimization. For instance, Kriging is a commonly used method for reconstructing a random field based on scattered data and was used to predict the distribution of gold deposits based on a few measurements [Kri51].

2.3.7 Numerical Methods

When handling sample-based data sets, in the implementation stage, numerical procedures are used to estimate PDFs or statistical quantities. We now introduce some of these methods used within this thesis.

Let $(\Omega, \mathcal{F}, P)$ be a probability space and let $X_k(A)$ $(k = 1, \ldots, n,\ n \in \mathbb{N})$ be the outcome of a real-valued random sample for an event $A \in \mathcal{F}$, in the following, for the sake of simplicity referred to as the *sample*. Moreover, let $X : \Omega \to \mathbb{R}$ be an equally distributed random variable, i.e., having the same distribution function $F = F(x)$ as for every X_k. Lastly, let $x_k = X_k(A)$ for all $k = 1, \ldots, n$.

Basic Statistical Quantities

Although we have no prior knowledge about F, we can use the sample to estimate some statistical quantities. According to the law of large numbers, these quantities will converge to the true quantities of X derived from the underlying distribution F (recall that, by definition, all X_k are identically distributed and mutually independent). However, in practice, we do not have an infinite supply of sample values. Hence, there is a discrepancy between the sample-based quantities and the true quantities of X. For this purpose, let $\overline{X}$ refer to the random variable induced by the distribution function in accordance with the sample.

A numerically stable method to compute the sample-based mean and variance in one pass goes back to an algorithm by Welford [CGL83, Wel62]. To begin with, we set $\mu \leftarrow 0$, $q \leftarrow 0$. Then, by iterating over all $k = 1, \ldots, n$, we update

$$\begin{aligned} q &\leftarrow q + \frac{k-1}{k}\left(x_k - \mu\right)^2 \\ \mu &\leftarrow \mu + \frac{x_k - \mu}{k} \end{aligned} \tag{2.3.36}$$

Finally, we end up with the sample-based quantities

$$\mathbb{E}\left[\overline{X}\right] = \mu \text{ and } \operatorname{Var}\left[\overline{X}\right] = \frac{q}{n}. \tag{2.3.37}$$

To compensate for the discrepancy between the sample-based and the true quantities, we can use Bessel's correction and thus obtain [Upt08].

$$\mathbb{E}\left[X\right] \approx \mu \text{ and } \operatorname{Var}\left[X\right] \approx \frac{q}{n-1}. \tag{2.3.38}$$

As mentioned earlier, relying on mean and variance is problematic, if the underlying distribution cannot be assumed as approximately Gaussian. However, we can derive other quantities from the sample that are more robust to non-Gaussian distributions; namely, the median, and the minimum and maximum of all sample values. The median is defined as the quantile $Q\left({}^{1}/_{2}\right)$ (see Section 2.3.4). The motivation behind taking the minimum and maximum value, is the fact that they are roughly equal to the lower and upper quantiles

$$\begin{aligned} \min\left\{x_k : k = 1, \ldots, n\right\} &\approx Q\left(\frac{1}{n}\right) \\ \max\left\{x_k : k = 1, \ldots, n\right\} &\approx Q\left(1 - \frac{1}{n}\right). \end{aligned} \tag{2.3.39}$$

Thus, the user is provided with a range, where most of the outcomes probably occur and with a threshold that separates the lower half from the upper half, i.e., the median. Unlike the mean and variance, these quantities obviously do not rely on any particular shape of the probability distribution in order to convey meaningful information. Of course, other quantiles can be presented to the user as well. Another advantage of this practice is the fact that we only communicate data to the user that actually exists in the sample. Hence, we do not have to worry about the possibility of showing meaningless information that does not occur in reality.

Mixture Models

There exist basically two strategies to provide the user with a deeper insight into the underlying distribution. Either, we directly visualize the distribution of the samples, or we first reconstruct an estimate of the underlying probability distribution derived from the given sample. The latter case is in particular preferable, when the samples are prone to substantial (measurement) errors or when smoothing operations are otherwise desirable, e.g., to simplify the data such that the user is not distracted by visual clutter.

A commonly employed approach to reconstruct probability distributions is the use of mixture models. They represent mixture distributions consisting of multiple components. Each component corresponds to a single probability distribution. We begin with the following definition [Dey10].

Let $f_l : \mathbb{R} \to \mathbb{R}$ $(l = 1, \ldots, m)$ be a set of $m \in \mathbb{N}$ probability density functions. Moreover, let $\alpha_l \in \mathbb{R}_0^+$ $(l = 1, \ldots, m)$. Then, any combination

$$f = \sum_{l=1}^{m} \alpha_l f_l, \text{ such that } \sum_{l=1}^{m} \alpha_l = 1 \tag{2.3.40}$$

is called a *mixture*.

Typically, the PDFs f_i are generated by a parametric family $f(x|\theta)$, such that

$$f_l(x) = f(x|\theta_l) . \tag{2.3.41}$$

A frequently used algorithm for this purpose is the EM algorithm (Expectation-Maximization algorithm) that produces a Gaussian mixture. That is, a mixture consisting of Gaussian PDFs, each specified by mean and variance, i.e. $\theta_l = (\mu_l, \sigma_l^2)$ (see Section 2.3.3). It begins with an initial guess for the parameter settings $(\alpha_1, \ldots, \alpha_m, \theta_1, \ldots, \theta_m)$, and then iteratively updates these parameters. The updating process is based on the given sample points, such that a weight is calculated for each pair of sample point and mixture component. These weights are then used to determine the next parameter settings. A complete explanation of this algorithm would go beyond the scope of this thesis. Instead, let us refer to Bilmes [Bil98].

Clusterings

The EM algorithm also yields a clusterization, namely it separates the sample points into distinct groups or clusters. More precisely, it returns the probability that point x_k belongs to cluster (i.e., mixture component) l, for all pairs of sample points and clusters [GC11]. This gives rise to the following definition [KMN98].

Let $(\Omega, \mathcal{F}, P)$ be a probability space and let $x_k = X_k(A)$ $(k = 1, \ldots, n,\ n \in \mathbb{N})$ be a sample for an event $A \in \mathcal{F}$. Let $m \in \mathbb{N}$. Then, a mapping

$$\mathcal{C} : \{1, \ldots, n\} \times \{1, \ldots, m\} \to [0, 1], (k, l) \to p_{k,l} \tag{2.3.42}$$

is called a *clustering* or *clusterization* of the sample, where $p_{k,l}$ is the probability of point x_k belonging to cluster l. It is said to be comprising m *clusters*. Moreover, if

$$p_{k,l} \in \{0, 1\} \text{ for all } k = 1, \ldots, n,\ l = 1, \ldots, m, \tag{2.3.43}$$

then $\mathcal{C}$ is said to be a *hard clustering*. Otherwise, it is said to be a *soft* or *fuzzy clustering*.

Notice that a soft clustering is subject to some extent of uncertainty, which is intuitively justified, because of the uncertainty introduced by empirical measurements or simulation runs. However, it still is sometimes preferred to use hard clustering, e.g., for the sake of simplicity. K-means and hierarchical clustering are commonly used algorithms that yield hard clusterizations. The concept behind hierarchical clustering is introduced in Section 5.3.2. For a comprehensive overview of clustering algorithms, let us refer to Jain [Jai10].

2.4 Visualizing Uncertain Data Sets

After establishing a mathematical foundation, we now discuss the visualization of multidimensional uncertain data sets. As it has likely become apparent to the reader, visualizing uncertain data is a difficult endeavor. Brodlie et al. have suggested some explanations, why this is the case [BOL12]:

- **Complexity.** Uncertainty is a complex issue, as it is self-referential, i.e., there is always uncertainty about uncertainty.
- **Different representations.** There are different ways of representing uncertainty that have to be considered, when designing a visualization scheme. Measurement accuracy, ensemble data and PDFs are some examples of such representations.

- **Propagation.** As already mentioned in Section 2.1, uncertainty propagates throughout the whole visualization pipeline. Hence, linking uncertainty to a single source is not faithful to reality.
- **Extra dimension.** Uncertainty adds an extra dimension, which has to be considered during visualization. This is not a trivial task, for instance, because occlusion artifacts have to be avoided and the human perception is limited to three dimensions.
- **Prominence.** Often, uncertainty is visualized in a prominent way and thus, areas of greatest uncertainty tend to outshine areas of greater certainty, which is not desirable in many applications.
- **Another discipline.** Many visualization techniques have been developed in interdisciplinary collaboration. Brining in uncertainty requires cooperation with experts from the field of statistics in order to evaluate and advance the theoretical background.
- **Rendering hardware.** Modern graphics hardware often requires a linear approximation of geometry, thus introducing the undesirable effect of rendering uncertainty.

Thus, for developing effective visualization systems, it is paramount to abstract from the raw uncertainty information in a way that preserves the important features from an expert's point of view. Such abstractions are often based on simplifying the underlying probability distributions or reorganizing the data, e.g. by clustering or segmentation.

Generally speaking, uncertainty visualization can be classified according to different criteria. In the following, we give an overview of popular approaches and present some examples of the respective visualization techniques. Hereby, we concentrate on multidimensional data as this is the focus of the present thesis. In this section, we give a short overview of related work and present a classification scheme, which allows us to put our contributions into context of the research area of uncertainty visualization. A more thorough outline of work that is related to ours is presented together with our specific contributions in the following chapters.

2.4.1 Data Dimensionality

According to Potter et al., uncertainty visualization can be categorized by data dimensionality and uncertainty dimensionality [PRJ12]. Data dimensionality is discussed in this section. The next section addresses the uncertainty dimension. Data dimensionality corresponds to the independent variables, i.e., the dimension of the sample point's space. Formally speaking, this is the dimension of the topological space of the underlying random field, provided that it is a Euclidian space (see Section 2.3.5). Some examples are shown in Figure 2.5.

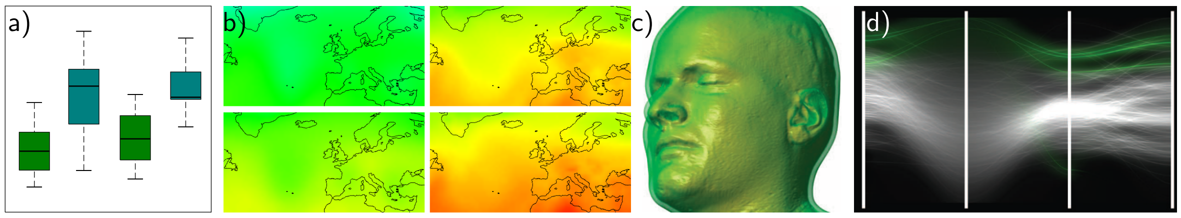


Figure 2.5: Examples for different data dimensionality. a) 1D: A typical box plot. b) 2D: Slice-based spatial view from our Multi-Charts visualization scheme [DDW14] (see Chapter 4, © 2014 IEEE). c) 3D: Combination of conventional isosurface with volume rendering techniques [JS03] (© 2003 IEEE). d) Higher dimensions: Parallel coordinates extended to include uncertainty information [FKLT10] (© 2010 IEEE).

One Dimension

For visualizing one-dimensional data with uncertainty, box plots are the most commonly used technique [BHJ*14]. In their basic form, they show five characteristic points, namely the minimum, maximum, lower and upper quartile, i.e., $Q\left({}^{1}\!/_{4}\right)$ and $Q\left({}^{3}\!/_{4}\right)$, and the median. Various modifications have been proposed, e.g., such that information about the density function is included [RM78].

Two Dimensions

Many techniques have been proposed for this case. Most prevalent is the use of color maps, often extended by additional features, such as glyphs, contours and annotations [PRJ12]. For instance, Potter et al. have introduced a visualization system that presents mean and standard deviation based on ensemble data, by color-coding the mean and indicating standard deviation in the form of overlaid contours [PWB*09a]. Osorio and Brodlie have extended the concept of contouring to probabilistic, i.e., uncertain, contours [AOB08]. Glyph-based visualization techniques are introduced in Section 2.4.3.

Three Dimensions

When visualizing three-dimensional data, it is significantly harder to integrate information about uncertainty into the visualization scheme. The reason for this lies in the fact that any 3D view suffers from inherent occlusion effects. Moreover, as screens are limited to two dimensions, some kind of transformation—typically a projection—is necessary to put 3D data on a 2D screen. Of course, in this process, the spatial context cannot be preserved, which constitutes another obstacle that has to be addressed. This problem can be rectified to a certain extent by allowing the user to move and rotate the object in order to get a better impression of its shape. Yet, the fundamental difficulty of dimensionality reduction prevails, in particular when adding another dimension of uncertainty. Clearly, it is infeasible to visualize full PDFs at every point in a 3D environment. Hence, sophisticated techniques are essential to address this issue in a way that does not oversimplify uncertainty information and at the same time avoids distracting the user by visual artifacts.

A great amount of research is devoted to visualizing uncertain 3D data sets. Djurcilov et al. have proposed a technique for integrating volumetric uncertainty into direct volume rendering, by either modifying the rendering equation or by running a post process [DKLP01]. Pöthkow et al. introduced a method to visualize the positional uncertainty of isosurfaces, which relies on the probability distribution of isosurfaces [PH11]. Pfaffelmoser et al. have built upon their method by factoring in correlations in order to obtain more reliable probabilities [PRW11]. By combining conventional isosurface with volume rendering techniques, surrounding regions of uncertainty can be visualized as areas of lower opacity [JS03]. Recently, Athawale et al. have introduced a probabilistic method to extract isosurfaces from uncertain scalar fields. Their approach quantifies and visualizes spatial uncertainty based on a probabilistic marching cubes algorithm [ASE16].

Higher Dimensions

The most prominent application of higher dimensional data is the integration of time as another dimension. Typically, such data sets are visualized by sequentially animating over time steps, which will be discussed in Section 2.4.3. Other applications include non-spatial domains, where dimensions correspond to parameters that can represent basically any (numerical) quantity. Visualizing such data sets is evidently an even more challenging task, as in this case even projections are in most situations useless. This is due to the fact that our visual cognition is limited to three dimensions, and with the possible exception of some very basic views, most human beings cannot conceive renderings of higher dimensional projections, forgetting even about the severe occlusion effects that would occur. Hence, completely different approaches have to be developed for this issue.

Two common alternatives to handle higher dimensional data sets are parallel coordinates and multiple views. Parallel coordinates are named after the fact that their axes are all aligned parallel to each other. Data points are then inserted as line strips such that each vertex is placed on the axis, which corresponds to the respective component of the data point vector. Ge et al. have used parallel coordinate plots to visualize uncertainty in remotely sensed data [GLLL09]. They classify data by utilizing a soft clusterization method and then show probability, fuzzy membership and the associated uncertainty, which are produced by their

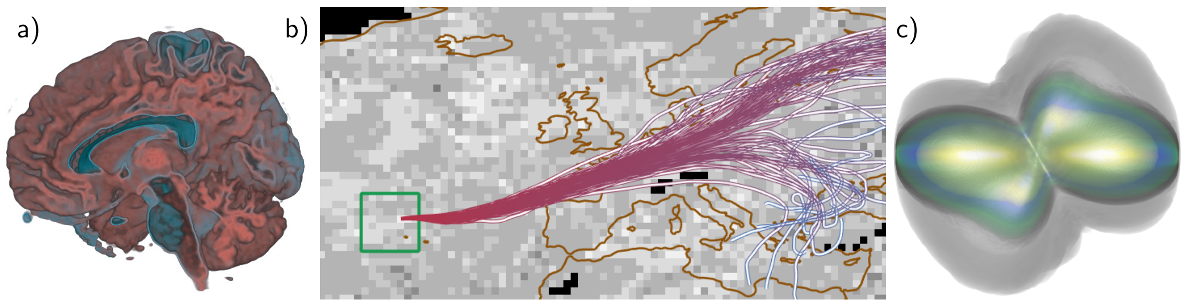


Figure 2.6: Examples for different uncertainty dimension. a) Scalar: Segmenting scalar data according to user-selected criteria of importance [KVUS*05] (© 2005 IEEE). b) Vector: Visualizing transport variabilities in flow field ensembles by enhanced spaghetti plots [JKW16] (© 2016 WSCG). c) Tensor: Volume rendering to depict the uncertainty from fiber orientation distribution function glyphs [JPGJ12] (© 2012 IEEE).

classification algorithm, on parallel coordinates. Feng et al. generate density plots based on uncertain data sets and then extend the concept of parallel coordinates to include uncertainty information [FKLT10].

2.4.2 Uncertainty Dimension

Depending on the dimension of the measure of uncertainty, different visualization techniques have been proposed. The dimensionality of uncertainty corresponds to the dependent variables, i.e., the dimension of the sample's image space (see Section 2.3.5). Examples are depicted in Figure 2.6.

Scalar

Visualizing scalar fields, i.e., where the sample's image space is real-valued, is perhaps the most thoroughly investigated area in the field of uncertainty visualization [PRJ12]. Numerous methods have been proposed to address this kind of data. Here, we will pick out only a few such examples. For instance, Mihai and Westermann have proposed a method for analyzing the stability of certain relevant points, namely the critical points, in scalar field ensembles. They derive statistical quantities from the ensemble, and then classify domain points accordingly with respect to the occurrence and type of critical points that can occur [MW14]. Coninx et al. have proposed a technique that combines a color-based approach with the integration of Perlin noise to pack data with their associated and present it to the user [CBDT11]. Kniss et al. have developed a system that visualizes fuzzy classifications by combining them into a single probabilistic space. Their classification algorithm is based on segmenting scalar data according to user-selected criteria of importance [KVUS*05]. Also notice, that most of the approaches mentioned in the previous section belong to the category of scalar fields.

Vector

When dealing with vector fields, where the sample's image space is vector-valued (i.e., multivariate), approaching uncertainty becomes a significantly more complicated issue. Intuitively, this is the case, because for vectors affected by uncertainty, deriving statistical quantities is even more prone to errors and misinterpretations than for scalar values. For instance, the concepts of mean and variance are apparently useless when vectors are pointing in very different or even opposite directions. Likewise, quantiles, and in particular the median, cannot be defined meaningfully in a straightforward fashion.

Visualizing vector fields can be subcategorized into conveying global and local information to the user. Depending on the application, a different strategy has to be chosen. Various techniques have been developed that extend both kinds of techniques to uncertain vector fields. Global visualization schemes are typically based on rendering pathlines of particles that are injected into the vector field, which is interpreted as a flow field. Botchen et al. have introduced a texture-based method to visualize uncertainty in non-stationary flow fields by advecting a texture over time. They indicate uncertainty by blurring the respective regions [BWE05]. A different approach has been proposed by Ferstl et al. that uses principal component analysis to discover trends in an ensemble of streamlines. These trends are then visualized as confidence regions, where streamlines will likely exist [FBW16]. Jarema et al. have proposed a comparative visualization technique for analyzing transport variabilities in flow field ensembles. They draw enhanced spaghetti plots using color and opacity to encode temporal evolution and representativeness for each member [JKW16]. Local schemes are often visualized by glyphs, which will be introduced in Section 2.4.3.

Tensor

Not so much work has been devoted to visualizing uncertain tensor fields as compared to the aforementioned cases. As this issue is clearly beyond the scope of this thesis, we introduce it only briefly by two examples. Jiao et al. have proposed a visualization technique for ensembles of fiber orientation distribution function glyphs. Their method utilizes volume rendering to depict the uncertainty extracted from the given ensemble [JPGJ12]. Zhang et al. have developed a glyph-based method to visualize local differences between diffusion tensors and have demonstrated the effectiveness of their approach in a user study [ZSL*16]. We will discuss glyph-based visualization techniques in Section 2.4.3.

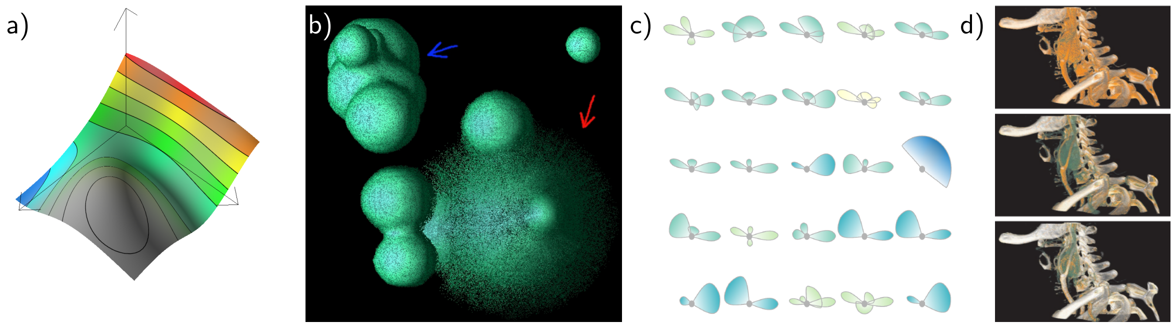


Figure 2.7: Examples of different approaches according to Pang et al. a) Attributes: Hyperslices from our contribution of visualizing progressive response surfaces, data certainty represented by saturation [DW13] (see Chapter 3, © 2013 EUROGRAPHICS and Blackwell Publishing). b) Geometry: Uncertain surfaces rendered as point clouds with a displacement proportional to uncertainty [GR02] (© 2002 IEEE). c) Glyphs: Glyph-based representation of directional distributions [JDKW15] (© 2015 IEEE). d) Animation: Time-varying appearance of uncertainty by using a probabilistic transfer function (three frames are shown) [LLPY07] (© 2007 IEEE).

2.4.3 Approach

Pang et al. have grouped the area of uncertainty visualization according to different approaches [PWL97]. Since then, various techniques have been developed that fit into those categories. We now introduce these groups of approaches and present a few examples. Some of them are shown in Figure 2.7.

Attributes

Modifying attributes of the geometry is perhaps the most easily understandable approach for incorporating uncertainty. Here, in the process of rendering, shading and lighting techniques are modified to produce renderings that feature information about uncertainty. Examples of this include color-coding uncertainty, altering reflectivity coefficients and varying surface normals.

Rhodes et al. have proposed a method for rendering isosurfaces such that a scalar extent of uncertainty is color-coded by modifying the hue in the resulting visualization. By altering opacity and hue, their approach can convey additional information along the uncertainty to the user [RLBS03]. In a recent article by Potter et al., uncertainty is determined by using the Shannon entropy which does not rely on the existence of a mean value, and is then mapped to a coloring scheme [PGA13]. Hengl has proposed a method that uses the Hue-Saturation-Intensity (HSI) coloring scheme to visualize both the predicted values and the uncertainty by encoding them into different channels of the HSI scheme: That is, the value is encoded as the hue, and the uncertainty as the saturation, where more certain values correspond to more saturated colors [Hen03].

Geometry

Adding or modifying geometry is another popular approach to depict uncertainty. For instance, contour lines or isosurfaces can be added to indicate the degree of uncertainty. Transforming existing geometry, e.g., by translation or rotation, is another way of displaying uncertainty.

As a typical example, consider spaghetti plots—a commonly used tool for showcasing isocontours of all members at the same time [All10]. Pfaffelmoser and Westermann have built upon this technique by reconstructing probability density values based on the initial ensemble. Then, on this basis, uncertainty and topology of isocontours are computed and visualized in an effective way [PW13]. Lasagna plots are another extension of spaghetti plots that confine contour lines to prevent overlapping and make use of color gradients to support the user in interpreting the result [SCJ*10]. Another approach was proposed by Grigoryan and Rheingans. They render uncertain surfaces as point clouds and displace the points along the surface normal by an offset proportional to the degree of uncertainty [GR02].

Glyphs

Another approach of visualizing uncertain data sets is the utilization of glyphs. Glyphs are geometric objects that encode information through shape and color. As glyphs can easily show directional quantities, they are often used in the context of vector or tensor fields. Although glyphs can be regarded as a special case of using geometry, according to Pang et al., they differ insofar as they represent discrete information [PWL97]. That is, glyphs do not convey continuous information. Instead, information from connected regions is aggregated and then a glyph is rendered at that region, which encodes the respective information.

For instance, Jarema et al. have proposed a technique for visualizing 2D vector field ensembles. To construct glyphs, they first generate directional distributions by using mixture models. Then, the result is mapped to a glyph-based representation, which is communicated to the user. By employing an interactive panning and zooming mechanism, experts can gain detailed insights into specific regions of interest [JDKW15]. Schultz et al. have introduced a system to visualize fiber probability distributions based on an embedding into a Hilbert space. This information is then rendered as a glyph-based view, which also provides an insight to the underlying uncertainty [SSSSW13].

Animation

Animation introduces another dimension in visualization schemes, thus allowing for integrating an additional quantity without increasing the complexity of any single frame. However, there are two obstacles that have to be considered when animation is employed: First, animations are necessarily sequential, meaning that the user can hardly recognize the connection between frames that are further apart in the animation. Second, animations cannot be printed out (except for printing out the individual frames), although this might not be such an important issue in an age of widely available computers and handheld devices.

Animation has been used in various ways in the context of uncertainty visualization. For instance, Gershon has presented a system that presents an otherwise static image by segmenting it into different components and then looping over these components. By using blurring techniques, uncertain details can be emphasized [Ger92]. Lundström et al. have introduced a method that uses animation to communicate uncertainty by using a probabilistic transfer function in the context of direct volume rendering. This results in time-varying appearances of uncertainty [LLPY07]. Another technique was proposed by Zuk et al. that reveals temporal uncertainty for archaeological data sets with different time stamps. By animating over the time, uncertainties can easily be discovered by the expert [ZCG05]. Recently, Hao et al. have introduced a method for visualizing temporal ensembles by employing 3D shape comparison techniques to detect common changes in shapes over time and members. They use animation to display these changes by continuously fading in and out member segments [HHB16].

Others

Other approaches include psycho-visual effects and sonification, i.e., communicating information to the user by using sound. As these approaches are clearly beyond the scope of this thesis, we do not go into further details. Instead, let us refer to Pang et al. for a detailed overview on that issue [PWL97].

2.4.4 Composition

Based on the scheme proposed by to Bonneau et al. [BHJ*14], we can classify uncertainty visualization techniques by the way in which they compose basic views. We finish our discussion about the classification schemes with this section.

Data Dimensionality	**1D**	**2D**	**3D**	**Higher Dimensions**
		b,d	b,c,d	a
Uncertainty Dimension	**Scalar**	**Vector**	**Tensor**	
	a,b,c,d	(b)		
Approach	**Attributes**	**Geometry**	**Glyphs**	**Animation**
	a,b	b,c,d	d	c
Composition	**Multiple Views**	**Integration**		
	a,b,d	c,d		

Figure 2.8: Our techniques in context of the presented classification schemes. Notice that some techniques fit into multiple categories of the same scheme. a) Progressive Response Surfaces [DW13] (see Chapter 3). b) Multi-Charts [DDW14] (see Chapter 4). Note that applying this method to vector field ensembles might be a worthwhile future research direction. c) Screen-space Silhouettes for Isosurface Visualization [DKW16] (see Chapter 5). d) Visualizing the Centrality of Shapes (see Chapter 7). Note that our contribution "Vector-to-Closest-Point Octrees" is not a visualization technique [DW15] (see Chapter 6), but it serves as a foundation of d).

Multiple Views

Often, uncertainty can be seen by visualizing the differences between images generated from different points of view or from different algorithms. A common approach for this is visualizing multiple views side-by-side such that they can be compared by the expert.

An example of this method can be found in the work of Jiao et al., where they quantify the differences between images produced by fiber tracking algorithms. By showing a side-by-side comparison, each pair of these algorithms can be analyzed [JPS*10]. Kehrer et al. have proposed a model for comparing many categories of multi-variate data sets in a small-multiple display. They organize categories in a hierarchical way, and then utilize this hierarchy to show meaningful comparisons between multiple graphics [KPBG13].

Integration

As an alternative to drawing multiple views next to each other, they can be integrated or overlaid in the same image. This has the advantage that the spatial coherence between the different views is preserved. However, it comes at the expense of additional occlusion effects, which has to be taken into careful consideration.

Wittenbrink et al. have presented several techniques of showing data with uncertainty in the same view that can be applied both to spatial and temporal domains. For instance, they overlay line glyphs in order to indicate regions of greater uncertainty [WPL95]. Jospeh et al. have developed a system for displaying uncertainty in isosurfaces. By visualizing statistical information, they enable users to compare isosurfaces generated by different algorithms. Uncertainty is visualized by various techniques, including overlaying, coloring and transparency [JLRP99].

2.4.5 Context

Now that we have introduced various criteria for classifying uncertainty visualization, we present an overview of our contributions and publications in the context of this scheme. This is shown in Figure 2.8. In this table, each of our techniques is assigned to its respective category of each classification scheme.

2.5 GPU Acceleration

Every visualization technique, when implemented, is built upon an underlying rendering system, i.e., a system that produces images from geometric models. In particular, when implementing 3D visualization techniques, it is paramount to have efficient rendering methods at hand in order to achieve interactive frame rates. Modern graphics cards are capable of running sophisticated visualization techniques in real-time. Hence, they provide the user with feedback mechanisms that allow steering the visualization interactively, e.g., by moving the camera or by rotating an object. Real-time rendering is the discipline concerned with this issue, and since we rely throughout this thesis on efficient rendering algorithms, we will give a brief introduction on this subject in the following.

Due to the progress in the development of modern graphics cards, they can nowadays also be used in the context of highly efficient computations. This is of particular interest in the context of uncertainty visualization, as these techniques often rely on transforming the raw data, e.g., ensemble data, into a form that is suitable for rendering. The previously mentioned mixture models, for instance, have this requirement. Consequently, using GPUs—graphics processing units—constitutes a great advantage on this issue, as it enables the user to quickly see results from the input data. For instance, when running a simulation in the background that continuously supplies new data by reacting to the expert's requests, visualizing the output can be a challenging task, where GPU-accelerated computations can prove as helpful. As being of great importance for some of the techniques presented in this thesis, we will also give a short introduction on this topic.

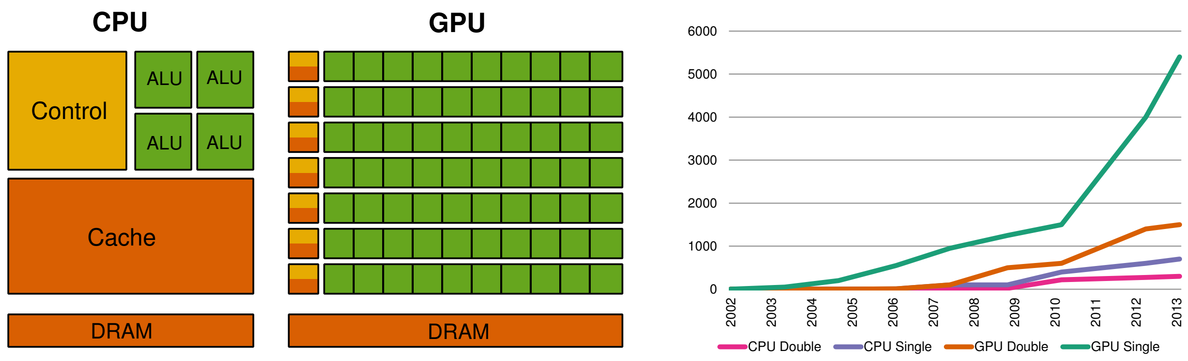


Figure 2.9: Left: Fundamental differences in design philosophies of CPUs and GPUs (adapted from NVIDIA [NVI10]). GPUs devote more transistors to data processing, CPUs have a greater priority on caching and flow control. Right: Comparison of theoretical floating point operations per second, in GFLOP/s, since 2002 [NVI10]. Notice the increasing gap between GPU and CPU.

2.5.1 Background

Let us begin with an overview of modern GPU's capabilities and let us point out the differences to traditional CPU-based implementations. Since 1945, when von Neumann first described the concept behind the von Neumann architecture, software programs are typically written in a sequential form [vN93]. That is, such programs are executed by sequentially processing the specified instructions. Historically, the expectation was that each new generation of processors involved a further increase in computational power. Thus, users and developers have continuously seen running their software faster without the need for modifying the concept of sequential execution. However, this trend has stopped in the past years and it will probably not resume in the foreseeable future. As a consequence, the paradigm of writing strictly sequential programs has to be abandoned for achieving significant performance improvements. Instead, by writing programs that run multiple threads concurrently, major increases in performance are possible. This is due to the fact that modern processors consist of multiple cores, where all cores execute instructions at the same time. Hence, the computational throughput is in the best case multiplied by the number of cores that are used. However, implementing parallelized algorithms is not a trivial task and not all algorithms can be parallelized at all [Sut05].

Rendering is a discipline that highly benefits from parallelization, as geometry and fragments can be processed independently from each other. Moreover, many of these instructions are floating point operations leading to a design philosophy of GPUs that maximizes their throughput of floating point operations [KH10]. Consequently, CPUs and GPUs have fundamental differences in their design philosophies. This is shown in Figure 2.10 (left). CPUs follow the multicore design that focuses on the optimization of sequential code by using control logic that distributes sequential instructions to multiple cores. Furthermore, they have large cache memories to reduce latencies of complex applications. This allows for running different instructions for each thread. Thus, threads are typically of heavy weight and managing and scheduling is done explicitly. In contrast, GPUs follow the manycore design that aims at maximizing computational throughput instead of minimizing latency. As threads are very lightweight and are managed on the hardware level, GPUs are adapted for running massive numbers of numerical computations concurrently. Consequently, they do not perform as well as CPUs on tasks that are not adapted to this design philosophy [KH10, Reg08]. The advantages of GPUs are not only relevant in the context of rendering, but they can be expanded to all areas of numerical computations, provided that efficient GPU implementations of such algorithms are possible. Figure 2.10 (right) shows a comparison of floating point operations per second for modern GPUs and CPUs. Clearly, the GPU outperforms the CPU significantly.

2.5.2 Real-Time Rendering

Real-time rendering is an interactive area of computer graphics, meaning that it produces images so quickly that action by the user is almost immediately followed by visual feedback. Due to the advances in modern graphics hardware, efficiency and quality of computer-generated images is ever-increasing [Möl08]. OpenGL, developed by the Khronos Group, and Direct3D, by Microsoft, are currently the two major graphics application programming interfaces (API) for desktop PCs [KH10]. Since the work in this thesis was implemented by exclusively using Direct3D for real-time rendering, we will only give a brief introduction to the Direct3D APIs. However, all techniques presented in this work can also be implemented without any restriction in OpenGL.

Introduction to Direct3D

Direct3D is a low-level graphics API for drawing 3D graphics and it is part of the DirectX software development kit (SDK) [Mic16]. The Direct3D API can be used for rendering triangles, lines and points, and starting from version 11, it can also be used for general-purpose parallel operations. By fully utilizing the graphics hardware and emphasizing parallel processing, very high computational throughput can be achieved. Direct3D provides an abstraction for various GPU implementations, thus saving the developer the time of learning manufacturer-specific hardware details. At the core of Direct3D is the graphics pipeline, where input data flows from different sources and images (or computational results) come out. Parts of the graphics pipeline are programmable, which makes Direct3D perfectly suitable for highly customized

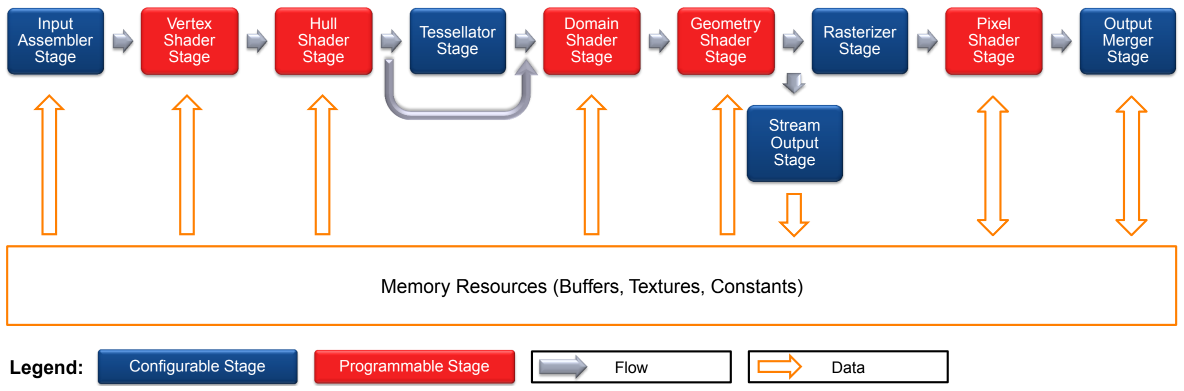


Figure 2.10: Graphics pipeline for Direct3D 11 (adapted from Microsoft [Mic16]). Stages in red are fully programmable using HLSL.

visualization techniques. Such programs are called shaders, and they are implemented in HLSL, the High Level Shading Language for DirectX. We will now continue with an overview of the graphics pipeline.

Graphics Pipeline

Figure 2.10 shows a data flow diagram through the stages of the programmable graphics pipeline for Direct3D 11. It is worth noting that each stage can be configured. Moreover, stages drawn in red feature shader cores and they are fully programmable using HLSL. We will now go on with a description of each stage [Mic16].

- **Input-Assembler Stage.** In the first stage, primitive data, e.g., triangles represented by vertex and index buffers, are read and assembled into primitive types, such as triangle lists, that will be processed by the next pipeline stages.
- **Vertex-Shader Stage.** In this programmable stage, vertices passed from the input assembler are processed, while optional per-vertex operations are carried out. For instance, vertices can be transformed, and other attributes such as lighting can be computed.
- **Geometry-Shader Stage.** This stage is also programmable and it has the ability to create or delete vertices. A geometry shader operates on primitives rather than on vertices. Hence, they can be used to alter the topology of the scene's geometry. Sprite rendering and fur generation are typical examples of using a geometry shader.
- **Stream-Output Stage.** The purpose of this stage is to stream vertex data to buffers in memory. Such data can then, for instance, be copied to a resource that can be read by the CPU and fed back into the pipeline.

- **Rasterizer Stage.** In this stage, vector information is rasterized into an image. This is achieved by transforming each primitive into fragments (or pixels) that are covered by the primitive's area. In doing so, per-vertex values are interpolated across the fragments. Moreover, clipping against the view frustum, perspective projection, and mapping to a viewport is carried out.
- **Pixel-Shader Stage.** The rasterizer invokes a pixel shader for every fragment covered by a primitive, which is executed in this programmable stage. It enables the implementation of highly customized shading techniques, such as lighting, post-processing and even ray-casting as demonstrated in one of our contributions (see Chapter 6).
- **Output-Merger Stage.** Finally, the previously generated fragments are combined and merged into the output image. Optionally, depth and stencil testing are performed to remove fragments that are not visible to the viewer, e.g., when they lie behind another object. Moreover, blending is optionally evaluated as a combination of multiple semi-transparent fragments on the same screen-space position.
- **Hull-Shader, Tessellator, Domain-Shader Stage.** Here, more complex geometry is converted into triangles. Since tessellation is beyond the scope of this thesis, we do not go into further details on this issue.
- **Compute-Shader Stage.** Although not shown in Figure 2.10 (and not part of the graphics pipeline), in this stage general-purpose computations can be carried out, separate from the actual rendering process. This technology is known under the name DirectCompute. We discuss the subject of general-purpose computations in the next section.

2.5.3 GPGPU Computing

General-purpose computing on graphics processing units (GPGPU) is a technique that uses GPUs to solve general-purpose computation problems. It started in the last decade, when developers discovered the great advantage of exploiting the numerical capabilities of GPUs for general computations. At first, such problems had to be transformed into graphics operations, which were then usually carried out on the pixel shader using Direct3D or OpenGL and written to a texture. However, this was a rather complicated task and it turned out that shaders were lacking some crucial abilities, e.g., means to write to calculated memory offsets. A major progress occurred in 2007, when NVIDIA released CUDA—a platform for parallel GPU computing—and a new generation of graphics cards equipped with CUDA-enabled chips. Since then, other platforms for GPU computing have been developed, namely, OpenCL, by the Khronos Group, and DirectCompute, by Microsoft, as a part of the DirectX

11 SDK [KH10]. We restrict our introduction to CUDA, as this was the only platform used for GPGPU computations throughout the work presented in this thesis.

CUDA

CUDA—originally an acronym for Compute Unified Device Architecture—is a parallel computing platform released by NVIDIA in 2007 that exploits the capabilities of the GPU. As it can be seen as an extension to C, developers familiar with C benefit from a low learning curve in CUDA [NVI10]. CUDA programs encompass two types of phases: Phases that are executed on the CPU, i.e., the *host*, and phases that are executed on the GPU, i.e., the *device*. The host code is written in standard C and it contains no or only little data concurrency. The device code is written in an extension of C by CUDA-specific keywords for identifying functions, that are executed concurrently on the GPU, the *kernels* [KH10]. The key idea behind CUDA programs is to outsource the computational heavy tasks to the GPU, where it can be processed much faster. Thus, the workflow of a typical CUDA application can be described as follows.

- First, data that has to be processed concurrently, is copied from host memory to device memory.
- Next, the CPU launches the execution of a kernel.
- The kernel is then executed by the GPU with random read and write access to device memory.
- Finally, the result is copied back from device memory to host memory and processed by the CPU.

To maintain scalability and efficiency, threads are partitioned into *blocks*, where each block contains a fixed number of threads and shared memory resources. All threads within one block have access to the block's shared memory and to the global device memory. However, accessing shared memory is significantly faster, and therefore it is necessary to organize threads in such a way that the use of share memory (versus global memory) is maximized in order to achieve high efficiency. In addition to that, there are many more pitfalls, when it comes to optimizing efficiency and fully utilizing the GPU's capabilities, which are clearly beyond the scope of this thesis. For further information, we refer the reader to the "NVIDIA CUDA C Programming Guide" [NVI10] and to Kirk and Hwu [KH10].

CUBLAS

Various libraries have been developed that use sophisticated algorithms to perform common tasks in computation based on CUDA. One such example is cuBLAS, a library provided by NVIDIA that implements BLAS—Basic Linear Algebra Subprograms—on top of the CUDA platform [NVI10]. BLAS is a de facto standard for linear algebra routines, such as matrix multiplications or linear combinations [LHKK79]. By utilizing the GPU, cuBLAS is a powerful tool for processing large data sets. We make use of cuBLAS in Chapter 3, where we utilize it to perform our progressive Kriging interpolation scheme on the GPU. Similar to CUDA, the developer must first copy data to the device memory, then call cuBLAS routines, and finally copy the result back to the host.

3 Progressive Visualization of Multidimensional Scattered Data

In this chapter, we introduce a novel method that allows the user to interactively explorate scalar-valued data sets via progressive response surface prediction from multidimensional input samples. This chapter is largely based on our publication:

Demir I., Westermann R.: Progressive High-Quality Response Surfaces for Visually Guided Sensitivity Analysis. *Computer Graphics Forum (Proceedings of EuroVis 2013) 32*, 3 (2013), 21–30. `doi:10.1111/cgf.12089`. [DW13]. © 2013 EUROGRAPHICS and Blackwell Publishing.

3.1 Introduction

Major increases in computation power over the last decades enable us to handle more and more complex and large data sets. More measurements for different variables can be taken into account leading to multidimensional data sets, which can be used in computations and analyzed in numerous ways. However, it is still challenging to visualize such data sets in a way, which is intuitive and easy to understand for the end user. Since it is virtually impossible for humans to recognize visual representations, which show more than three dimensions, it is necessary to reduce the dimensionality of multidimensional data sets in some fashion in order to make it comprehensible to the user. In this chapter, we focus on multidimensional data sets, represented by scalar functions, $f : \mathbb{R}^n \to \mathbb{R}$, where the values depend on many independent variables. In many of these applications, the purpose is to examine the sensitivity of a process and its outcomes with respect to the parametrization of the input variables, and to finally optimize the result considering the detected dependency relations.

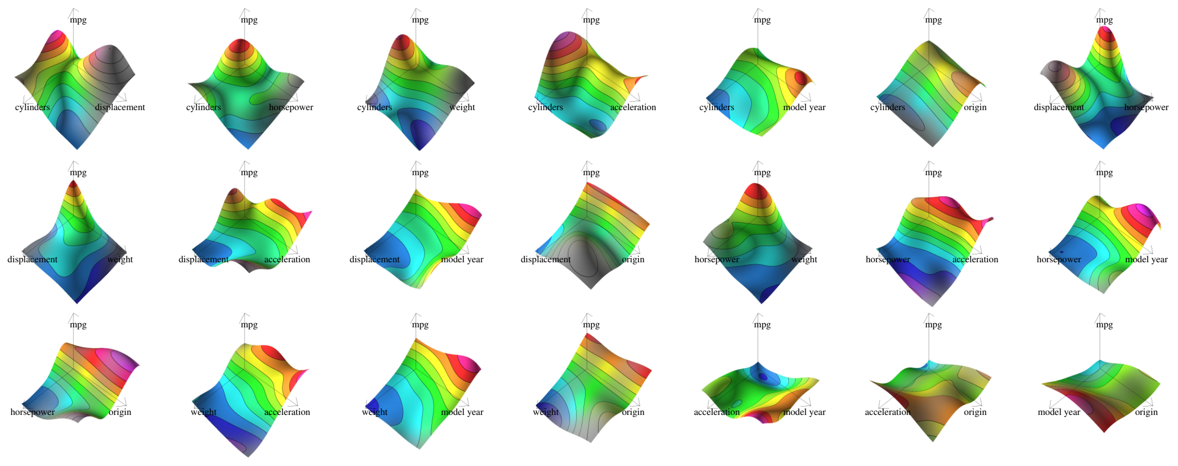


Figure 3.1: Visualization of response surfaces for all pairs of parameters of a multidimensional scalar function. Incremental update and visualization of the surfaces is performed at less than 5 ms.

Very often, multidimensional functions are not given analytically, but values can only be accessed at a discrete set of sampling points in the n-dimensional parameter space. Despite often being a time-consuming preprocess to generate the results, sometimes the result can be updated at very high rates from a given new parametrization. This possibility becomes more and more feasible in numerical experimentations due to the ever-increasing capabilities of computing architectures. As already recognized by van Wijk and van Liere [vWvL93], it is particularly interesting, since it gives rise to a visual navigation system in higher dimensional parameter domains and allows steering towards an ideal solution by interactive adjustments.

Nevertheless, visual exploration of multidimensional data in an intuitive way is a daunting task, since recognizing visual representations beyond three dimensions is practically impossible for human perception. Therefore, multidimensional data analysis, in general, requires reducing the dimensionality of the multidimensional data set and thus, to enable visualizations using common techniques for data with less than 4 dimensions.

A widespread approach for dimensionality reduction is to slice the data or to project it orthogonally onto some two-dimensional subspace of the n-dimensional parameter space, and then to view the resulting value distributions in these subspaces. Since in general, the sample positions are sparse in the selected subspaces, dimensionality reduction requires interpolation between sample points to calculate a continuous representation in the employed 2D sampling structures. This gives rise to an effective prediction scheme of the values at unobserved sites and the dependencies between parameters and function values.

In particular, if the continuous approximation is visualized as a response surface, namely the graph of the continuous objective function plotted over the two-dimensional subspace, an improved exploration of the relationships between parameters and the objective is conceivable. The option to consider features of the surface's topology such as extreme points, gradients, or ridges, has the potential of greatly elevating the understanding of the dependency relations.

Unfortunately, calculating response surfaces from multidimensional input sample points is computationally costly, for the reason that it requires a mechanism for interpolating between scattered points in the sampling structure, over which the surface is formed. For large input samples, a time consuming initial preprocess is then necessary to get the interpolation weights for surface reconstruction. Even more notably, when input samples are created progressively, for instance by user-driven parameter space navigation, every new sample point involves reiterating the exhaustive preprocess, prohibiting an interactive visual analysis of the evolving surface. Especially the memory consumption can quickly become a bottleneck, as many sample points are needed to resolve a multidimensional space at a sufficiently fine resolution.

The contribution of this chapter is to propose a technique, which provides the user with an intuitive visualization of higher dimensional scalar data sets. Our method enables an interactive visual exploration of multidimensional scalar functions via high-quality response surfaces (see Figure 3.1 for an example). The surfaces are computed via Kriging interpolation [Kri51], a Gaussian process regression model for inference from scattered samples in a multidimensional parameter domain. Kriging determines the interpolation weights solely by the data configuration and the covariance model, and it finds the least squares estimate of a stationary random function that minimizes the variance of the random function increments. We make the following contributions to the field of multidimensional scalar data interpolation and visualization:

- **Fast incremental updates of Kriging calculations.** To make Kriging suitable for interactive exploration of large sets of multidimensional scalar samples, we provide a progressive updating scheme for the Kriging interpolation weights. This scheme builds upon incremental matrix inversion [Ban37], and it enables updating the interpolation weights with only minor computational effort when new sample points are added. We employ a similar principle as underlying online learning algorithms [Opp98], where only the last example is used for updating a learning network's parameters. For data sets being so large that the construction of a response surface would consume an unacceptably amount of time, the scheme allows to construct the surface incrementally by considering only one new sample point at a time. As Kriging requires exhaustive use of matrix-vector and vector-vector operations to find the interpolation weights and data estimations, we have implemented the entire scheme on the GPU, including incremental updates of the inverse Kriging matrix and interpolation.

- **Response Surface Selection.** To enable an interactive visual analysis of multidimensional scalar functions, we have embedded the progressive updating scheme for Kriging interpolation into a slice-based navigation interface similar to scatterplots and HyperSlice [vWvL93]. We propose the use of parallel coordinates to interactively select the 2D subspaces of the high-dimensional sample space, for which response surfaces should be computed and visualized. As we provide immediate visual feedback about the structure of the response surfaces in all subspaces, new sample values can be generated while the user can steer the location of these points interactively to further refine regions of interest.

The remainder of this chapter is organized as follows: Next, we discuss work that is related to ours. In Section 3.3, we introduce the concept behind Kriging and sketch its use for scattered data interpolation. Our progressive scheme, including a complexity analysis and GPU aspects, is presented in Section 3.5.2. Then, we discuss approaches for selecting response surfaces via slicing and projection. We finally present results using a real-world data set as well as detailed performance statistics, and we conclude the chapter with a discussion about future extensions of our work.

3.2 Related Work

Our scheme is built upon popular procedures for dimensionality reduction of multidimensional data, in particular orthogonal projection and slicing [Asi85, FB94, vWvL93]. Scott provides an overview of techniques in multidimensional data visualization [Sco92]. Another overview is given by Grinstein et al. [GTC01]. Our model for displaying the reduced data is that of a response surface, i.e., a continuous surface that predicts the objective values from given sample points over a selected two-dimensional space [MM95, Mon06]. Unlike scatter plots [Cle85], we aim at displaying continuous regions by drawing response surfaces, from which the user can see correlations with less effort and estimate values at unmeasured sample positions. A different approach is the use of parallel coordinates [Ins85, Weg90], where the data are mapped to 1D graphs rather than points in a 2D subspace. Parallel coordinates perform a dimensionality reduction and support findings of the existence of linear dependencies between variables; however, spatial relationships are lost and topographic attributes as for response surfaces cannot be determined.

To compute a continuous approximation of a discrete multidimensional function with respect to a given sample, we employ methods for scattered data interpolation [FN91, Wen05]. Examples of frequently used methods for scattered data interpolation problems are radial basis functions (RBFs) [Buh03], since they are able to interpolate arbitrary constraints in a smooth manner. Even though compactly supported radial basis functions can be used to efficiently interpolate large amounts of data, because they resemble sparse linear systems, they do not provide the same approximation quality as basis functions of global support. On the other hand, globally supported RBFs yield dense matrix structures and, hence, significantly less efficient solution algorithms. Even though improved approaches exist, such as multipole methods [CBC*01] and incremental least-squares solutions [BK05], these procedures involve a considerable amount of preprocessing, when new sample points emerge.

A different group of statistical means for predicting an underlying function from a given set of discrete sample points are Gaussian processes models [OK78, Nea99]. They utilize a Gaussian process to express the a priori uncertainty with reference to the function based on empirical observations. These models try to characterize the dependency of an observation on a corresponding input by a conditional distribution. If the observation, and hence, the function, is a one-dimensional scalar, then we call the distribution a regression model. In geostatistics, regression with Gaussian processes to predict a spatial phenomenon at unobserved sites is known as Kriging, which constitutes a BLUE (best linear unbiased estimator) interpolation scheme [Kri51, MB62, Mat63].

So far, in the field of visualization, only few approaches have adopted statistical prediction of continuous representations from discrete sets of multi-dimensional samples for sensitivity analysis. Examples include the work by Pieringer et al. [PBK10], where surface plots of multidimensional functions have been used to analyze the sensitivity of surrogate models. Torsney-Weir et al. [TWSM*11] have employed Gaussian process models for predicting quality metrics in image segmentation based on selected parameter settings. Moreover, they exploited the concept of response surfaces to display the variation of the used objective function. Berger et al. [BPFG11] have used nearest-neighbor and model-based statistical prediction methods to derive objectives at unobserved locations in parameter space. In their system, they have implemented statistical sampling in parameter space to obtain a set of training data, based on which they train a response function. Recently, Schlegel et al. [SKS12] shed light on the interpolation properties of Gaussian process regression, including Kriging interpolation, for the purpose of interpolation and offered analytical descriptions of the underlying spatial basis functions.

To the best of our knowledge, in none of the previous works, efficiently running continuous statistical predictions was addressed in the context of discrete sets of multi-dimensional samples, which are updated gradually. Present approaches are usually predicated on the conjecture of a static set of observations at known positions in the parameter domain. The opportunity to handle progressive updates of the internal representations, for instance, when new observations emerge, has not been taken into consideration so far. Hence, we regard our approach as an essential addition to these works that opens up new prospects for interactive multidimensional data discovery.

3.3 Kriging Interpolation

Kriging interpolation is a probabilistic method to describe a quantity at unobserved sites from a discrete set of observations at given locations. The principles underlying Kriging interpolation were introduced by Krige [Kri51] in the context of geostatistics, and later put into a formal concept by Matheron and Blondel [MB62, Mat63]. Underlying Kriging is the notion of a random function, which describes a quantity over a spatial domain as a set of random variables at the given locations. The spatial relations between the given observations are expressed by the covariance structure for all pairs of variables, often represented by the variance of the observed increments over distance, i.e., the variogram.

The main advantages of the Kriging method are the ability to interpolate between scattered sample points, i.e., there is no need for the sample points to be aligned on a regular grid, and the fact that Kriging is a best linear unbiased estimator (BLUE). That is to say, Kriging minimizes the mean squared error and has a mean error of 0 while using only linear combinations of the given sample points.

Kriging first performs a structural analysis of the given observation to derive the dependency structure, and then estimates interpolation weights for each given location by solving a least squares problem. A thorough introduction to Kriging is given by Cressie [Cre93]. Even though the interpolation properties of Kriging are well known, its use outside geostatistics is limited. In our opinion this is mainly because of the inherent computational complexity for calculating the linear interpolation weights. A major drawback of the Kriging interpolation is the relatively high amount of necessary computations in order to calculate the linear weights. Solving the Kriging equations directly for n observations involves inversion of an $n \times n$ matrix.

To overcome this limitation, we have developed a GPU-friendly method, which makes heavy use of parallelization. Other acceleration schemes have been proposed, like ad-hoc methods [Haa95] using locally adaptive covariance prediction, fixed rank Kriging [CJ08] using empirical low-rank variances and covariances estimates, or a GPU implementation which intertwines the calculation of Kriging weights and interpolation in a very efficient way [HCL*11]. To the best of our knowledge, possibilities to recompute Kriging weights progressively upon the arrival of new sample points on the GPU has not been proposed until now.

3.3.1 Main Principles

The main idea behind Kriging is to spatially interpolate the quantity at a point x^* by finding linear weights λ_i of the k collected data points $(x_i, f(x_i)), 1 \leq i \leq k$, yielding the interpolated value

$$\hat{f}(x^*) = \sum_{i=1}^{k} \lambda_i (x^*) f(x_i) . \tag{3.3.1}$$

The estimation is such that the error conditions mentioned before are satisfied. There have been developed different types of Kriging, which employ different ways of calculating those weights, such as simple, ordinary and universal Kriging. Here, we consider only ordinary Kriging, since, aside from the sample, it only needs a covariance function to accomplish the interpolation. The covariance is a probabilistic measurement, which specifies to which degree two random variables change together. More formally, it is defined as

$$Cov(x, y) = \mathbb{E}\left[(x - \mathbb{E}[x]) \cdot (y - \mathbb{E}[y])\right],$$

where $\mathbb{E}$ denotes the expectation value operator. Intuitively the covariance should increase when two points of the random field are closer together. Assuming that the underlying random field is stationary, i.e., expectation value μ and variance σ^2 are constant, and covariance depends only on the distance between two points, we can use a simple model to estimate the unknown covariance. Here, we use the Gaussian model given by

$$Cov(x, y) = \begin{cases} (\sigma^2 - n) \exp\left(\frac{-\|x-y\|^2}{a \cdot r^2}\right) & x \neq y \\ \sigma^2 & x = y \end{cases},$$

where σ^2 denotes the sill, to which the covariance tends if the distance between x and y decreases, and r denotes the range, which determines how rapidly the covariance decreases with a larger distance. Note that the sill is also equal to the variance, since the covariance equals the variance if $x = y$. A nugget effect is modeled by n to prevent oscillatory results when sample points with different values lie close together. The value of a allows further adjustment of the impact of the range and is typically set to $a = 1/3$.

3.3.2 Ordinary Kriging

We now briefly introduce the ordinary Kriging method. In order to calculate the linear interpolation weights $\lambda_i(x^*)$ at the interpolation point x^*, let us consider the covariances between all sample points $Cov(x_i, x_j)$, and the covariances between the interpolation point and all sample points $Cov(x_i, x^*)$, where $1 \leq i, j \leq k$. To begin with, we try to find weights $\lambda_i(x^*)$ such that

$$Cov(x_i, x^*) = \sum_{j=1}^{k} \lambda_j(x^*)\, Cov(x_i, x_j)$$

holds for every $i \in \{1, \ldots, k\}$. In other words, the covariance between any sample point x_i and the interpolation point x^* is equal to the linear combination of the covariances between x_i and every sample point x_j weighted by λ_j.

By defining the covariance matrix C as

$$C = \begin{pmatrix} Cov(x_1, x_1) & \cdots & Cov(x_1, x_k) \\ \vdots & \ddots & \vdots \\ Cov(x_k, x_1) & \cdots & Cov(x_k, x_k) \end{pmatrix}$$

and the covariance vector c and weight vector x^* as

$$c(x^*) = \begin{pmatrix} Cov(x_1, x^*) \\ \vdots \\ Cov(x_k, x^*) \end{pmatrix}$$

as well as the weight vector

$$\lambda(x^*) = \begin{pmatrix} \lambda_1(x^*) \\ \vdots \\ \lambda_k(x^*) \end{pmatrix}$$

The problem of finding the Kriging weights now comes down to solving the system of linear equations

$$C \cdot \lambda(x^*) = c(x^*),$$

where C is the full covariance matrix, (x^*) is the weight vector, and c is the covariance vector to the sample point.

However, one obstacle needs to be overcome, namely the fact that the weights do not necessary sum up to 1, which is generally required for the interpolation value to be unbiased. Therefore we introduce a Lagrange multiplier $\nu(x^*)$ such that

$$C \cdot \lambda(x^*) + \mathbf{1} \cdot \nu(x^*) = c(x^*), \lambda^T(x^*) \cdot \mathbf{1} = 1.$$

By defining the extended matrix and vectors

$$C_+ = \begin{pmatrix} C & \mathbf{1} \\ \mathbf{1}^T & 0 \end{pmatrix}, \; c_+(x^*) = \begin{pmatrix} c(x^*) \\ 1 \end{pmatrix}, \; \lambda_+(x^*) = \begin{pmatrix} \lambda(x^*) \\ \nu(x^*) \end{pmatrix}$$

the problem can be rewritten as

$$C_+ \cdot \lambda_+(x^*) = c_+(x^*)$$

yielding the solution

$$\lambda_+(x^*) = C_+^{-1} \cdot c_+(x^*) . \tag{3.3.2}$$

In addition, we calculate the Kriging variance to obtain a measure for the uncertainty in the current sampling

$$\begin{aligned} \sigma_k^2(x^*) &= \sigma^2 - c_+^T(x^*) \cdot C_+^{-1} \cdot c_+(x^*) \\ &= \sigma^2 - c_+^T(x^*) \cdot \lambda_+(x^*) . \end{aligned} \tag{3.3.3}$$

This uncertainty tells, where the density of observations is too low such that no reliable estimate of the response surface is possible. In our current tool, the uncertainty is used to guide the user towards regions in the parameter space where additional samples should be retrieved.

It can easily be recognized that ordinary Kriging is an actual interpolation method meaning that interpolating at the position of any given sample, i.e., $x^* = x_i$, returns exactly the value of the very same sample, i.e., $\hat{f}(x^*) = f(x_i)$. That is because in this case we have $(C_+)_i = c(x^*)$ leading to the solution $\lambda_+(x^*) = (0, \ldots 0, 1, 0, \ldots 0)^T$, where 1 lies at the i-th position. However, this may also lead to difficulties when sample points lie very close together or have even an identical position, which would imply a singular Kriging matrix with two identical columns. To overcome this issue we can modify the implementation of the covariance function in a way that it only returns σ^2 if x and y refer to the actual same given sample and apply the nugget effect in any other case. This modification results in a more stable approximation method.

3.4 Minimizing Uncertainty

Since we are adding new sample points incrementally, it is desirable to add them in a way that uncertain areas diminish as soon as possible. Therefore, when adding new points, we favor such positions, where the distance to the nearest previously added sample position is maximized, yielding a preferably even distribution. In this section, we will present an efficient algorithm that achieves this goal approximately.

3.4.1 Largest Empty Circle Algorithm

To begin with, let us consider a two-dimensional finite point set P bounded by a rectangular domain $D \subset \mathbb{R}^2$. Our aim is now to find a point within D such that it is the center point of the largest empty circle. By that we mean a circle, which does not contain any point in P. This center point obviously satisfies the requirement of maximizing the distance to its nearest point in P. We start by constructing a Delaunay triangulation for P, which is a triangulation, such that no circumcircle of any triangle contains any point in P, also called the Delaunay condition. There have been developed different algorithms for this construction, but since we are adding points incrementally, we also use an incremental algorithm for the Delaunay triangulation based on edge flipping as proposed by de Berg et al. [dBCvKO08]. However, wo will not go into more detail here.

After constructing the Delaunay triangulation, we iterate over all triangles in order to find the center of the largest empty circle. For this, we check for each triangle the radius of its circumcircle and update the center position accordingly, if the radius is greater than the largest radius found so far, and the center point lies within D. In addition, we check for each triangle, if it is a border triangle, i.e., there is no neighbor at the edge $e = \overline{p_0 p_1}$. In this case, we compute the perpendicular bisector of e and its nearest intersection $\hat{c}$ with the boundary of D. Now, we consider $\hat{c}$ also as a possible center point of a circle with radius $\|p_0 - \hat{c}\| = \|p_1 - \hat{c}\|$, since e is a border edge, which means that no other point in P lies within this circle. Note that each triangle might have up to three border edges, in which case we consider each border edge separately.

Finally, we consider the four vertices of D as possible center points. Also, note that after this process, we have found the largest empty circle: Any other point that lies within a circumcircle of a triangle is closer to at least one of its vertices. Furthermore, any center point that does neither lie within a circumcircle of a triangle nor on the boundary of D can be improved by moving it towards the boundary, thus increasing the radius of its largest empty circle. Finally, any center point on the boundary of D, which is neither a vertex of D nor does it lie on a perpendicular bisector of a border edge, is suboptimal, since it is closer to one vertex of its nearest border edge than to the other and can thus be improved by moving it either closer to the intersection with the perpendicular bisector or to a vertex of D.

Note that special care has to be taken, if P contains less than two points. This case, however, will not be discussed here.

3.4.2 Extension to More Than Two Dimensions

Although it is possible to extend the Delaunay triangulation to higher dimensions, we refrain from doing so, since the time consumption of this process increases rapidly with higher dimensions. Therefore, we make use of a different approach based on projection, which gives us an efficient approximation of the optimal sample position for $n \geq 2$ dimensions. We start by constructing $\binom{n}{2}$ domains D_{ij}, $1 \leq i < j \leq n$, such that each domain covers the projection onto two Cartesian axes of the sample positions. Then for every new sample, we add its projection to all domains D_{ij} and then run for each D_{ij} the largest empty circle algorithm separately yielding the center points c_{ij} with radii r_{ij}. Now, we construct an empty $n-$dimensional center point c^* and while there are at least two empty components of c^*, we copy the center point c_{ij} with the greatest radius r_{ij}, which suffices the requirement that both components $c_i^*$and c_j^* are still empty to these components. If there is still one empty component left, which is the case, when n is odd, we assign a random number from its domain to it.

3.5 Progressive GPU Kriging

To perform ordinary Kriging interpolation, the inverse of the extended covariance matrix C_+ is required. Since C_+, and thus its inverse, have to be updated every time new sample points are added, Kriging can be very time- and memory-consuming when implemented naively. To address this problem we propose a method, which enables us to update the Kriging matrix incrementally, meaning that the previous inverse matrix can be reused and only a small matrix has to be inverted. This method is based on block-wise matrix inversion proposed by [Ban37].

In block-wise inversion one assumes that the $(k+l) \times (k+l)$ matrix to be inverted is in block form, i.e., it consists of an upper left $k \times k$ and lower right $l \times l$ matrix P and S, and a lower left $l \times k$ and upper right $k \times l$ matrix R and Q. It is assumed that the block P is invertible. Then, the matrix inverse can be computed as

$$\begin{pmatrix} P & Q \\ R & S \end{pmatrix}^{-1} = \begin{pmatrix} W & X \\ Y & Z \end{pmatrix}, \text{ with}$$

$$Z = \left(S - RP^{-1}Q\right)^{-1}, \; X = -P^{-1}QZ, \; Y = -ZRP^{-1}$$

$$W = P^{-1} - P^{-1}QY = P^{-1} - XRP^{-1}.$$

In our scenario, since both C and C_+ are symmetric due to the fact that $Cov\,(x, y) = Cov\,(y, x)$, we can assume $P = P^T$, $R = Q^T$, $S = S^T$. Thus, we arrive at the following simplified formula

$$\begin{pmatrix} P & Q \\ Q^T & S \end{pmatrix}^{-1} = \begin{pmatrix} W & X \\ X^T & Z \end{pmatrix}, \text{ with}$$

$$Z = \left(S - Q^T P^{-1} Q\right)^{-1}, \; W = P^{-1} - P^{-1} Q X^T. \tag{3.5.1}$$

Our method for incrementing the inverse of the extended covariance matrix is now split into two stages: From the given previous sample positions $x_i, 1 \leq i \leq k$, the previous covariance matrix C and its inverse C^{-1}, and the added sample positions $x'_j, 1 \leq j \leq l$, the new inverse covariance matrix C'^{-1} is calculated in the following way. Let $P := C$, $(Q)_{ij} := Cov\left(x_i, x'_j\right)$, and $(S)_{j\tilde{j}} := Cov\left(x'_j, x'_{\tilde{j}}\right)$, where $1 \leq i \leq k$, $1 \leq j, \tilde{j} \leq l$. Then we have

$$C' = \begin{pmatrix} P & Q \\ Q^T & S \end{pmatrix}$$

which enables us to calculate C'^{-1} by block-wise inversion. Here it is worth noting that it is not necessary to store C since it is not required in order to calculate C'^{-1}. Only C^{-1} has to be stored. Also, note that the only matrix that needs be inverted is of dimension $l \times l$, where l equals the number of added sample points.

In the second stage we calculate C'^{-1}_{+} by a similar procedure. We now set $P := C'$, $Q := \mathbf{1}$, $S := 0$ and obtain

$$C'_{+} = \begin{pmatrix} P & Q \\ Q^T & S \end{pmatrix}.$$

By applying the block-wise inversion, we finally obtain C'^{-1}_{+}. Again, C' is not required in the computation, and only C'^{-1} has to be stored. The matrix to be inverted is of dimension 1×1. Note that most operations in this process are matrix multiplications, which can be efficiently parallelized on the GPU.

3.5.1 Computational Complexity

We now study the computational complexity of our proposed method and compare it with a conventional matrix inversion algorithm, which does not exploit reusing the previous inverse matrix.

In the progressive Kriging approach, the matrix C'^{-1} is obtained by first calculating $V := P^{-1}Q$, Z, X, and W, and then putting these intermediate results together. The reason for calculating V explicitly is that it is used in subsequent steps. Since it takes $\mathcal{O}(pqr)$

floating point operations to compute the matrix multiplication of one $p \times q$ and one $q \times r$ matrix, $\mathcal{O}(pq)$ operations for adding two $p \times q$ matrices, and $\mathcal{O}(p^3)$ operations for inverting a $p \times p$ matrix, calculating V, Z, X and W respectively takes $\mathcal{O}(k^2 l)$, $\mathcal{O}(kl^2 + l^3)$, $\mathcal{O}(kl^2)$, and $\mathcal{O}(k^2 l)$ operations. This gives a total of $\mathcal{O}(k^2 l + kl^2 + l^3)$ operations for increasing the inverse covariance matrix by l new sample points. A similar consideration leads to $\mathcal{O}(k^2)$ operations for computing $C_+'^{-1}$.

Let us now consider a state-of-the-art GPU matrix inversion method as proposed by Ezatti et. al. [EQOR11]. Their algorithm takes $\Theta(p^3)$ operations to invert a matrix of size $p \times p$, i.e. $\Theta\left((k+l)^3\right)$ operations for calculating $C_+'^{-1}$. Compared to the progressive approach this yields a significantly higher run-time complexity when l is small compared to k, i.e., $l = \mathrm{o}(k)$. In this case our method has a complexity of $\mathcal{O}(k^2)$ as opposed to $\Theta(k^3)$ for the conventional method. In section 3.7.2 we verify this result in practice.

3.5.2 CUDA Parallelization

The capabilities of recent GPUs are employed via the CUDA programming language to perform both the calculation of the Kriging interpolation weights and the final interpolation of the initial data sample in a highly efficient way. The first step when a new sample point arrives and has to be considered in the Kriging interpolation is the calculation of the covariance vector with respect to the new sample position. This process is carried out in a straightforward way by computing each element of the covariance vector by one CUDA thread in parallel. In all our examples we used the Gaussian model with nugget effect as the covariance function. Even for large samples, the overall time for performing this step is so small that it does not affect the overall runtime.

From the discussion of (progressive) Kriging, it becomes clear that all time-critical computations, which are required when new sample points are added and have to be considered in the interpolation step are matrix multiplications. Matrix-matrix multiplication can be parallelized quite effectively on the GPU by computing each entry in the result matrix in parallel. This concept can be further improved by a technique called tiling, which reduces redundant memory accesses (see Kirk and Hwu [KH10]). In our current implementation we use cuBLAS for virtually all matrix operations, a highly optimized GPU matrix library available in CUDA. One of the great achievements of cuBLAS is that it makes efficient use of the GPU memory hierarchy, trying to best utilize the fast memory segments that are shared by certain groups of threads.

Updating the Kriging matrices C^{-1} and C_+^{-1} is done in a straightforward way by restating Eq. 3.5.1 with cuBLAS functions. To calculate the weight vectors $\lambda_+(x^*)$ at the interpolation points all at once with a single call to the cuBLAS library we utilize the fact that we can combine all matrix-vector products of Eq. 3.3.2 to a single matrix-matrix product. That is to say, we define

$$\hat{C}_{+} = \begin{pmatrix} c_{+}\left(x_1^*\right) & c_{+}\left(x_2^*\right) & \cdots \end{pmatrix}$$

as a matrix containing all interpolation points, leading to

$$\hat{\Lambda}_{+} = C_{+}^{-1} \cdot \hat{C}_{+},$$

where each column of $\hat{\Lambda}_{+}$ contains the weight vector of the corresponding interpolation point.

The only exception where we do not make use of cuBLAS is for computing the Kriging variance (see Eq. 3.3.3) as we need to calculate a dot product at each interpolation point. Since cuBLAS does not provide a routine, which allows parallelized computation of many dot products at once, we implemented a CUDA method for this task. It is based on the most efficient parallel reduction method presented by Harris [Har07].

3.6 Visualizing Response Surfaces

Let $P \subset \mathbb{R}^n, n \geq 2$ denote a discrete multidimensional data set with a given scalar value assigned to each element via the mapping $f : P \rightarrow \mathbb{R}$. Now, every tuple $(p, f(p))$ represents a sample point, where $p \in P$ indicates the position and $f(p)$ the value of the sample point. Our goal is to visualize this data set by a set of surfaces such that each surface represents the scalar values over two distinct dimensions. For each selected parameter pair, we define a two-dimensional planar sampling grid and interpolate the scalar values at the sample positions in this grid via Kriging. These values are then used to form a surface. Each surface is rendered as a triangle mesh on the GPU, using colors in HSL space to indicate differences in value and uncertainty. The hue of a surface point is determined by its altitude, i.e., the scalar value at the corresponding grid point, while its saturation is set inversely proportional to the uncertainty. Additionally, we draw contour lines to provide the user a better classification of the surface points.

3.6.1 Response Surface Selection

For selecting the response surfaces that should be visualized, we utilize the HyperSlice method. Given a user-selected center point, for each pair of parameter space axes a 2D slice parallel

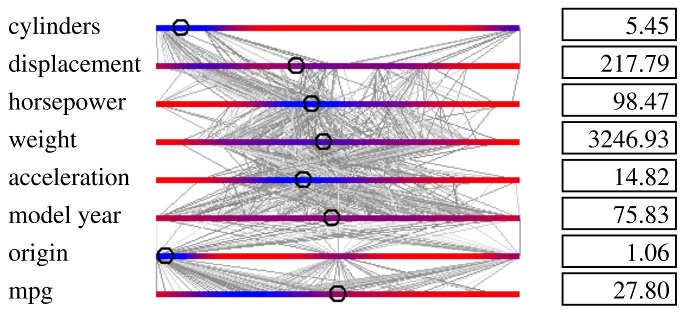


Figure 3.2: Utilizing parallel coordinates to adjust the center point (indicated by circles) in HyperSlice. Parallel coordinates are color coded to show the sample density on each parameter axis.

to the plane spanned by the two axes and going through the selected point is defined. For every slice a separate response surface is computed.

More formally, for each pair $1 \le i, j \le n$, the corresponding slice is defined by the set

$$\{(c_1, \ldots c_{i-1}, x_i, c_{i+1}, \ldots c_{j-1}, x_j, c_{j+1}, \ldots, c_n) \mid x_i, x_j \in \mathbb{R}\} .$$

The HyperSlice method enables the user to interactively change the center point and thereby steer through the whole data set. In the original work, this was performed by pressing a mouse button while the cursor points to a slice, and then dragging the center point according to this slice. In contrast, we employ an approach, which makes use of parallel coordinates to indicate the sample density in the region, where the center point is actually positioned. In this way, the center point can be adjusted towards those parameter intervals that are already well resolved, or, for instance, in a visual steering application, those intervals that are poorly resolved can be prioritized for point selection. The proposed selection procedure is demonstrated in Figure 3.2, where each sample point is represented by a polyline, which has its vertices aligned at the parallel coordinate axes. To support the user in choosing the position of the center point, every axis is color-coded from blue to red according to decreasing sample density.

Besides visualizing the response surfaces of all selected slices in a structure similar to a scatterplot matrix (see Figure 3.1 for such a visualization according to the selected center point in Figure 3.2), the use of parallel coordinates for center point selection gives rise to an alternative visualization strategy: The user changes the coordinate of the center point along one selected parameter axis interactively, while keeping all other coordinates fix. Thus, the response surfaces for every pair of axes involving the selected axis remain unchanged, and only for every other pair a change of the surface is triggered.

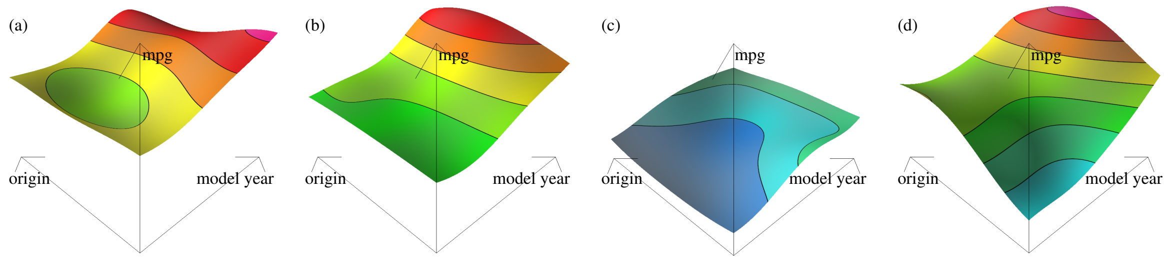


Figure 3.3: HyperSlice is used to display the fuel efficiency (mpg) according to origin and model year for three different tuples of horsepower and weight: (a) $(50, 1900)$, (b) $(80, 2500)$, (c) $(150, 3600)$. Note that there is virtually no impact of origin. (d) The whole sample is projected to the selected subspace and smoothly interpolated. Now one can see that cars from one origin tend to have low fuel efficiency, meaning that they generally differ in other attributes like horsepower or weight.

This is demonstrated in Figure 3.3, where we analyze data describing the fuel efficiency of automobiles in miles per gallon (mpg) [Aut93]. It contains a total of 398 sample points, where each point corresponds to one specific car, comprising the values mpg, cylinders, displacement, horsepower, weight, acceleration, model year, car name, and origin, a discrete value representing different states (note here that the use of Kriging for nominal data like origin is for demonstration purposes only, and that in general any interpolation of nominal data requires some data-specific rational). Figs. 3.1 and 3.2 show visualizations of this data. By choosing three center points with distinct values for horsepower and weight, one obtains three response surfaces displaying the fuel efficiency according to origin and model year as shown in Figure 3.3a,b,c.

In Figure 3.3d, all automobile sample points are projected to the subspace represented by the same axes used in HyperSlice before, and a response surface is computed from the projected sample. The advantage of projection over HyperSlice is that the whole sample is visualized simultaneously, giving an immediate overview of the whole data set without requiring the user to adjust a focal point. The corresponding response surface shows that cars from one origin generally have low fuel efficiency. However, for now we do not know if this is really caused by the origin or just by the fact that cars from other countries tend to differ in other attributes, for instance they might generally have less horsepower. This difficulty arises because the distances between sample positions in the high-dimensional sample space get lost due to the projection. This leads to the false impression that some distant sample points lie close together.

When HyperSlice was used in Figure 3.3a,b,c, the center point was placed at three different positions, with vastly different values of horsepower and weight. In contrast to projection, HyperSlice shows that the value of origin does not have a significant impact on the fuel efficiency even if entirely different values for horsepower and weight are chosen, which generally have the greatest influence. For other slices not shown here, we obtain similar views. We can thus conclude that the origin influences certain other attributes, for instance horsepower, which on their part affect the fuel efficiency. This example also demonstrates the difference between the surface projection and the HyperSlice method. Especially in an interactive session, where the user moves the center point coordinate along the selected parameter axis, the corresponding changes of the response surfaces can effectively reveal more subtle dependencies.

Based on this example, we can conclude, that it might be a good practice to first use projection and then HyperSlice to visualize a multi-dimensional data set. The projection method gives the user a general overview of the whole data set, even when only a small but representative subset of the sample is considered. For instance, such a subset can be generated by randomly selecting points from the initial sample. Relevant features can then be further analyzed using HyperSlice, by incrementally adding sample points that are close to the currently selected center point in all but two variables. Whenever the user changes the center point, new sample points will always be selected according to this new position. Due to the specific selection strategy of the next points to be considered, even for very large data sets a rather small subset of the given sample might already suffice for an accurate analysis.

3.6.2 Slice-based Interpolation

Our next step is to apply Kriging interpolation to HyperSlice and projection. Since all slices visualized in HyperSlice share the same space, the method requires only one Kriging matrix containing the covariances of the actual multidimensional sample positions. This matrix needs to be updated only when new sample points are added, because it does not depend on the position of the currently selected slices. As the inverse Kriging matrix is required to perform the interpolation, it has to be updated as well if the Kriging matrix was changed. For each interpolation point, i.e., the grid points of the sampling structures used to represent the selected slices, the covariance vector is calculated and used to determine the interpolation weights of every other point via Eq. 3.3.2. These weights are finally used to compute the interpolation values via Eq. 3.3.1.

More precisely, given an $n-$dimensional data set leading to $\binom{n}{2}$ slices, where each is displayed at a grid resolution of $r_i \times r_j$ we need to perform $\sum_{1 \leq i < j \leq n} r_i \cdot r_j$ Kriging interpolations overall. This formula can be simplified to $\binom{n}{2} \cdot r^2$ if $r_i = r$ for $1 \leq i \leq n$.

The interpolation step can be fully parallelized since every interpolation can be computed independently of each other. In order to calculate the covariance vectors at the interpolation points we need to embed the 2D sampling structures into the multidimensional sample space.

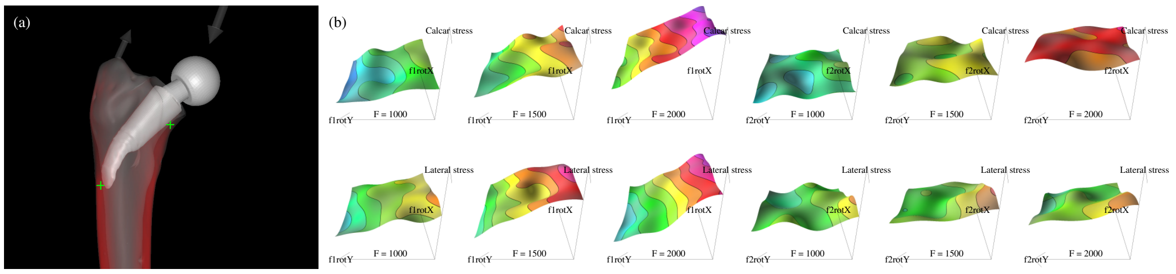


Figure 3.4: (a) Simulation setting, including arrow glyphs indicating simulated forces. (b) Response surfaces for different force magnitudes depending on force direction.

This is performed by setting the coordinates of the interpolation points along the two spanning parameter axes according to their relative position in the respective 2D subspace and filling the remaining coordinates with the values of the center point. This also implies that each time the center point is moved or the resolution of the sampling grid is changed the interpolation process has to be repeated.

The projection method requires a procedure that differs mainly in the computation of the covariances. Since in this case every sample position is projected to a 2D subspace, we have to maintain a separate Kriging matrix and its inverse. For the Kriging matrix, we use a covariance function that takes only those components of each sample position into account that correspond to the respective subspace. The same holds for the computation of the covariances at the interpolation points. Unlike embedding the grid points into the multidimensional space, we project the sample positions to the selected subspace and then calculate the covariance between the projected sample positions and the grid points.

3.7 Results

3.7.1 Multidimensional Real-World Data Set

In addition to the data that we have used so far for demonstrating the potential of response surfaces for sensitivity analysis, we have employed our approach in a computational steering setting for analyzing material stresses depending on applying forces. The underlying application is an implant planning environment for hip joint substitutes, where in a step prior to the surgery, a doctor tries to find the patient-specific best implant shape, size and position [DGBW08]. The design of the steering tool was created such that external forces can act on the surface of an implant, which has been inserted into the bone, and the resulting stresses in the bone interior can then be simulated and visualized. To achieve a high degree of realism of stresses that happen during walking, we consider an additional force of a constant scale applied by the muscles. Figure 3.4a shows the simulation scenery, where arrows indicate the simulated forces.

The simulation calculates the internal stresses, i.e., the scalar von Mises stress norm, in less than 200 milliseconds after the external forces have been specified. The volume visualization in Figure 3.4a shows the simulated scalar field. In our specific scenario, we have investigated the sensitivity to the introduced forces of the stresses at two critical points P_1 and P_2 in the calcar region and the lateral femoral wall, marked by green crosses in Figure 3.4a. To this end, we have incorporated the response surface approach into the simulation tool, thus, allowing the user to alter the points interactively, at which the forces are acting and the force magnitude through a navigation tool similar to the one shown in Figure 3.2. We stipulate a force direction in polar coordinates of two virtual spheres enclosing the regions of contact that are acting towards a sphere's center, and compute the intersection points between the force vectors and the implant and bone. Overall, we have 5 degrees of freedom, namely 2 angles for each point and a force magnitude, and we obtain stress values for the points P_1 and P_2.

Upon simulating the stresses, the simulation module exports the values at P_1 and P_2, and these values are then considered in the computation of new response surfaces. Figure 3.4b shows the response surface visualization using HyperSlice after approximately 7000 altered parameter settings have been used. In an interactive session, the expert would now begin with refining the parameters in local extremum regions to analyze the stress sensitivity in more detail, or he would compare the response surfaces for a different implant. As we will show next, by using a traditional method for response surface construction from the given sample, an interactive parameter domain exploration would be infeasible.

3.7.2 Performance Analysis

The performance analysis is structured in two parts. First, we measure the performance of incrementally constructing the covariance matrices, when new points are added, and then we measure the time to perform the Kriging interpolation at different grid resolutions. All measurements were performed with HyperSlice on an NVIDIA GeForce GTX 580 graphics card.

Figure 3.5 shows the time in milliseconds for computing the inverse of a covariance matrix, when the inverse for a given number of points exists and one new point arrives. In our tests, we have considered samples of different dimensions $d \in \{4, 6, 8, 10\}$, and we have made the respective graphs distinguishable by using different colors. Note that for a particular d, we compute $\binom{d}{2}$ surfaces and therefore, the same number of covariance matrices is required. These are computed and inverted in parallel on the GPU. We compare the run-times to those

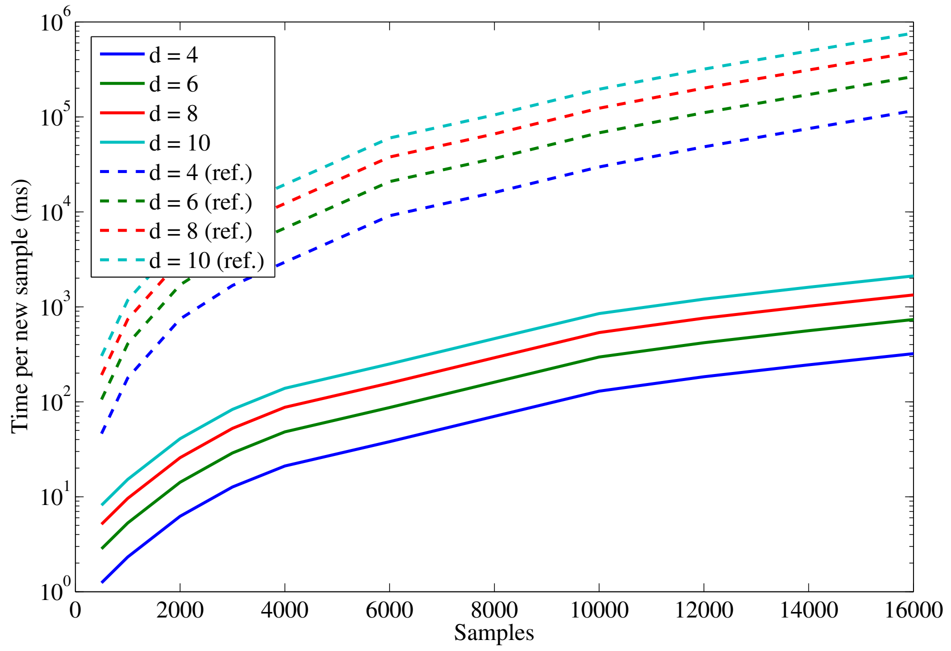


Figure 3.5: Computation times in *ms* for inverting the covariance matrix when sample points are added incrementally. Times for different numbers of dimensions d are considered and compared with a direct non-incremental approach as reference.

being achieved with a state-of-the-art GPU matrix inversion method as proposed by Ezatti et. al. [EQOR11]. Here, for each new sample point, the whole Kriging matrix has to be inverted at once. The timings are shown in Figure 3.5 as dashed lines. By using a logarithmic scale, one can observe that the progressive approach is by several magnitudes faster than the conventional GPU approach.

Before the Kriging interpolation can be performed, the inverse Kriging matrices have to be computed using the inverse covariance matrices. This step needs to be carried out only once, before the interpolation takes place, and its time consumption is negligible compared to that of the other parts.

The performance of the final interpolation of data estimates at the points of the discrete sampling grids has been measured for different data sets comprising 100, 500, 1000 and 2000 points, and at different grid resolutions of 10×10, 20×20, 50×50 and 100×100. The timing statistics is shown in Figure 3.6. We compared the times of our GPU-based implementation and its CPU counterpart, computed on an Intel Xeon X5675 CPU at 3.07 GHz. It can be clearly seen that the GPU method is significantly faster and allows moving the center point nearly in real-time even for large samples. Since the resolution can be changed at any time without affecting the Kriging matrix, it is possible to use a lower resolution grid, while adding

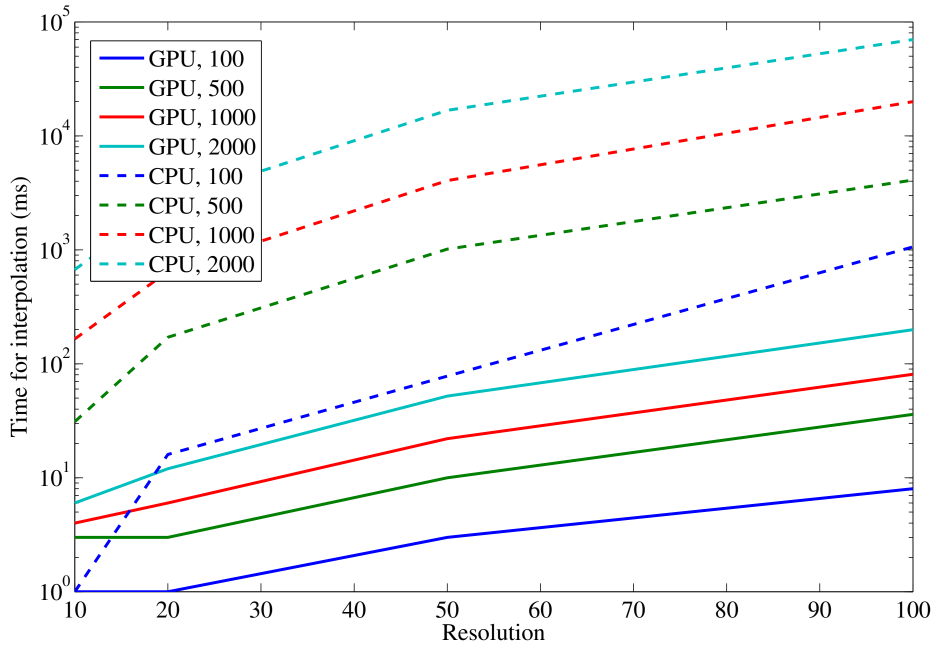


Figure 3.6: GPU/CPU Computation times for Kriging interpolation at different resolutions and for different sizes of samples.

new points or moving the center point and switching to a higher resolution grid for analyzing the surfaces in more detail.

3.8 Limitations

This section discusses the limitations of our technique. First of all, our approach becomes practically infeasible, if we want to analyze data sets with a great number of dimensions, i.e., roughly more than $n = 10$ dimensions. The main reason is due to the fact that we need to visualize $\binom{n}{2}$ surfaces, which leads for $n \gtrapprox 10$ to very small surfaces that can hardly be recognized and analyzed on any normal screen. This problem can be partially resolved by displaying only a subset of all surfaces chosen by the user at one time. In that case, however, it is harder to get an overview of the whole data set. Another reason is that the computation time for the matrices and for performing the interpolation increases with the number of surfaces, thus slowing down the whole process.

The covariance function also imposes a limitation not related to computation power. Since we do not know the correct covariance model for a data set unless it is explicitly specified, the expert has to choose a covariance function, such that it is a good approximation to the real covariance. This can lead to inaccurate surfaces and even to inaccurate information about uncertainty. Another issue lies in the method itself that we use for visualizing the data set. By projecting the sample to surfaces, we lose information about the distances between the sample positions and for the HyperSlice method, we cannot visualize the whole sample at once. Furthermore, it might not be the best choice to use only Cartesian axes, but our method can be extended to projections and slices for different axes in a straightforward manner. However, it is a non-trivial task to find optimal axes, which can be achieved for instance by projection pursuit techniques as proposed by Friedman and Tukey [FT74].

3.9 Conclusion and Future Work

In this chapter, we have presented a novel approach for visualizing multidimensional data via response surfaces using the Kriging interpolation. Kriging has the advantage that it allows interpolating between scattered sample positions and results in smooth surfaces provided that the covariance function is suitable for the given data set. It also provides us with a measurement of uncertainty that supports the user in getting more confident results. For constructing the surfaces, we have proposed two different methods: By projecting the sample point onto each surface and then interpolating between these points, we obtain a general overview of the whole data set, where the main structures become visible after only adding relatively few points to the interpolation scheme. HyperSlice, in contrast, interpolates between the given sample points in multidimensional space, in which each visualized surface represents a plane with respect to different axes through a user-specified center point. This center point can be changed by the user at any time. Due to the GPU-accelerated implementation, interpolating and hence moving the center point can be done without lagging. Finally, we tested our approach with real-world data and were able to obtain intuitive and effective results by using our visualization technique.

Our approach opens up different aspects that can be investigated in the future: Firstly, by integrating GPU-Kriging as proposed in Huang et al. [HCL*11], the interpolation step can be accelerated. Even though this step is not the bottleneck in our current implementation, it might become so, when interpolating on higher resolution sampling structures or in 3D. A user-study would provide insight to the question of how well the response surfaces' topography can guide the user towards specific features. Even though it was observed by Tory et al. [TSD09] that colored point plots in a 2D domain can be remembered easier by the viewer than 2.5D visualizations such as response surfaces, the surface representation can encode local features and their relative positions and elevations in a more effective way. Local fluctuations of the surface, especially in regions, where the sample set is sparse, can be easier quantified. By analyzing functions for which the locations of extreme points are known, and by examining the occurrence of response surfaces in the vicinity of such points, shape-based feature indicators might be discovered. Ideally, these indicators guide the user towards locally maximum or minimum locations, even though the sample points, at which the extreme values occur, have not yet been generated.

4 Comparative Visualization of 3D Scalar Field Ensembles

In this chapter, we introduce a visualization technique based on bidirectional linking between extended bar charts (multi-charts) and volume visualization. Our approach enables the expert to visually analyze three-dimensional scalar field ensembles. This chapter is largely based on our publication:

Demir I., Dick C., Westermann R.: Multi-Charts for Comparative 3D Ensemble Visualization. *IEEE Transactions on Visualization and Computer Graphics 20*, 12 (Dec. 2014). doi:10.1109/TVCG.2014.2346448. [DDW14]. © 2014 IEEE.

4.1 Introduction

Various approaches exist to visually convey the information in 3D scalar fields. A classic example of such methods is volume rendering. In combination with transfer functions, volume rendering effectively reveals the location and shape of relevant features in an interactive environment. Moreover, it is a spatially coherent visualization scheme, but it faces the problem of occlusion and attenuation effects. Consequently, the number of features to be seen at the same time is limited, which constitutes a restriction on the effectiveness of volume rendering, when prior (spatial) knowledge about features is lacking. In particular, when exploring ensembles of fields, quantitative comparisons between different parameters are of great importance, which poses a further limitation on volume rendering.

By combining techniques from information visualization and volume rendering via brushing and linking, this problem can be addressed [GRW*00, DH02, PKH04, ZMH*09]. These methods begin with an overview of the data by drawing attribute views, and by brushing functionality in these views, the user can then select data values meeting certain criteria. Now, the attribute views are linked to a 3D view generated by volume rendering. In this scenario, every selection in attribute space causes an instant visual feedback on the corresponding spatial positions. Widespread attribute space representations are scatterplots, histograms, and parallel coordinates.

In this chapter, we shed light on the use of brushing and linking in the context of volume rendering for visualizing 3D scalar field ensembles, $f : \mathbb{R}^3 \times \{1, ..., m\} \to \mathbb{R}$. In such a scenario, each ensemble member embodies the values of the same attribute at a set of locations in 3D space. For larger numbers of ensemble members, e.g., 40,000 in one of our applications, handling ensembles as multi-variate data with m parameters per spatial point and relying on conventional visualization techniques for such data is infeasible.

Our method aims at quickly guiding the user towards locations, where the data values and their distributions are similar or differ considerably over the ensemble members. Moreover, we aspire to provide the user with a quantitative comparison scheme for specific members in the context of all others. In theory, this task can be accomplished by determining confidence intervals at each point in space and then visualizing these intervals by means of traditional volume rendering. Yet due to the restrictions of volume rendering, it is a difficult task to visually resolve the desired information in the generated views. Another well-researched methodology is to condense the displayed information by focusing on particular features in the data [PWL97, DKLP02], like isosurfaces, and display their potential deviations across the ensemble. Recently, comparative visualization of data variations at designated positions in space and time has become an encouraging strategy for 2D ensembles [HMC*13, HMZ*14].

We introduce an alternative visualization technique for 3D scalar field ensembles that does deliberately not rely on prior knowledge about relevant features and their spatial occurrences. In this situation, a global comparative visualization of the ensemble at the data level is essential. On the other hand, using a fully automated analysis procedure is generally prohibited, due to the lack of confidence about what we are looking for.

As a solution, we propose a graphical interface that allows the user to visually analyze statistical properties of 3D ensembles effectively and, consequently, to permit the user to guide the analysis process towards features of interest. Our approach is built upon a novel volumetric representation based on an abstract view, where spatial locations and variations of the data values at these locations over the ensemble members can be distinguished simultaneously. The abstract view is constructed by linearizing the 3D space and by using multiple stacked and combined bar and line charts, the *multi-charts*. In the bar charts, statistical ensemble summaries in sub-regions of the 3D domain are encoded, and the line charts showcase the deviations of individual members at the same spatial locations. Our tool allows the user to interactively brush the multi-chart and in this process select regions exhibiting certain statistical attributes. Linked 3D views provide an instantaneous visual feedback to these actions. Furthermore, to provide a visual analysis of the relationships between the variations at spatial locations apart from each other, we utilize a similarity-based sorting algorithm and clustering methods based on these variations.

Our specific contributions of this chapter are:

- A 1D visual level-of-detail volume representation that relies upon a Hilbert curve linearization.
- The simultaneous use of bar and line charts. Bar charts are used to present statistical summaries. Individual ensemble members are visualized as superimposed line charts.
- By combining automated statistical data mining techniques and visually guided user interaction, we aspire to detecting and analyzing relevant data characteristics.

The remainder of this chapter is organized as follows. First, we review work that is related to ours. We then give an overview of our approach, including a description of the essential functionality that it offers. In the next section, we introduce multi-charts, and we describe their principal layout as well as the kind of statistical information they encode. This is followed by a discussion of the similarity-based preparation of the chart, and the bidirectional linking technique that we propose for 3D spatial views of the selected data points. We then demonstrate applications of multi-charts in several real-world cases from meteorology, fluid simulation, and astrophysics. We conclude this chapter with remarks on future research directions.

4.2 Related Work

Our method can be classified as a summary-based ensemble visualization technique. We compare sample attributes at fixed locations and also provide means to study the relations between the attributes at different locations. Based on descriptive statistics for data analysis, our approach considers the frequency of occurrences of data values over multi-dimensional regions and in particular their correlations. In addition to ensemble visualization, our technique overlaps with methodologies in the area of brushing and linking, and diagram techniques.

Uncertainty Visualization. A vast number of overviews and taxonomies of uncertainty visualization techniques has been published [JS03, MRH*05, THM*05, GS06, PRJ12, HBG*12]. In the context of ensembles, uncertainty has often been revealed by visualizing quantities such as mean and standard deviation via color maps, opacity, texture, animation, and glyphs [WPL02, DKLP02, RLBS03, LLPY07]. Feature-based approaches have been introduced to extract the uncertainty information from 3D ensembles with regard to position and arrangement of particular structures, such as isosurfaces, and to visualize such characteristics by drawing confidence envelopes [PWL97, ZWK10, PH11, PW12], geometric displacements [GR04], or by using surface animation [Bro04].

Potter et al. [PWB*09a] have confirmed the effectiveness of combining linked statistical visualization techniques with user interaction for 2D weather ensembles. Thompson et al. [TLB*11] offered a volume rendering technique showcasing statistical summaries at multiple resolution levels in 3D space. In meteorology and geoscience, spaghetti plots are broadly used to display all isocontours of a scalar ensemble at the same time. Sanyal et al. [SZD*10] improved upon spaghetti plots by drawing glyphs and confidence ribbons to emphasize the spread of contour-based ensembles. Höllt et al. [HMC*13, HMZ*14] have shown the use of time-series glyphs for a comparative visualization scheme for 2D ensembles at two different points in time.

Brushing and Linking. Such methods are founded on the concept of arranging two or multiple separate views next to each other, all representing the same data, to stress different aspects [BC87, Shn98, WBWK00, Kei02]. Moreover, these views can be linked to each other, so that selections in one view—performed by brushing or picking functionality—are directly propagated to the other views. Modifications and additions of this methodology have been suggested regarding alternative attribute views and enhanced algorithms that support the user with smart selection strategies. Prevalent attribute space representations include multi-dimensional scatterplots [PKH04], histograms [KBH04], and parallel coordinates [Ins85, HLD02]. Augmenting brushing and linking techniques with focus and context mechanisms has been proposed in various publications [Hau05, Dol07]. In particular, for the analysis of multivariate 3D fields, a multitude of tools that are based on brushing and linked views has been suggested [GBS*99, GRW*00, DH02, KLM*08, ZMH*09, KFH10, KMDH11]. Some of these proposals can even handle time-dependent data. A number of multiple-view approaches have been used in the context of volume rendering applications as well that strive for improved feature discovery and sophisticated navigation techniques [TPM05, WS06, GXY12].

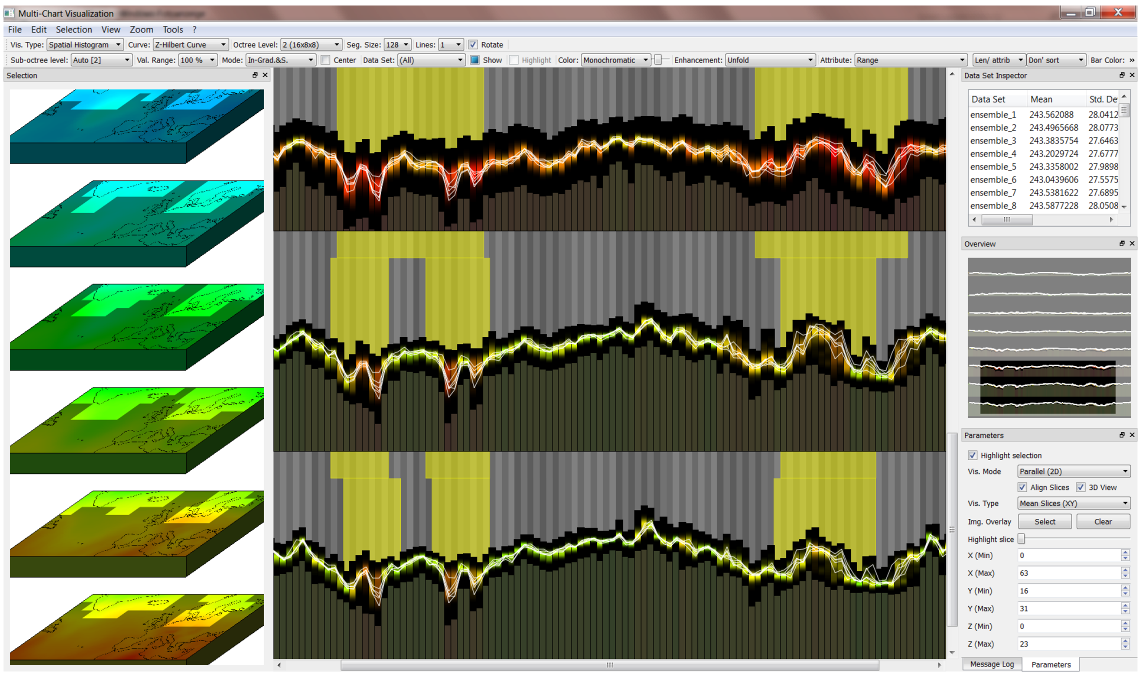
Diagram Techniques. Bar, box and line charts are regularly used in the area of information visualization to analyze large sets of categorical and numerical data. Our approach is also predicated on such primitives. Bar charts are broadly employed for summarizing categorical data; in particular, showing aggregated values for the categories is a typical application of such charts. To make bar charts suitable for presenting in-depth information of the categories, bars have adapted such that this information is encoded at the pixel level [KHDH02, KHDL07]. On the other hand, box plots [EB03] use bars as basic elements, but they are augmented by additional visual cues to communicate data values and other important characteristics of the data, such as spread or quartile. Potter et al. [PKRJ10] have extended the concept of box plots to also visualize statistical data summaries. Their approach makes use of the concept of violin plots [HN98] in order to reveal distributions at the data level by drawing shapes of the plotted primitives accordingly. More recently, Whitaker et al. [WMK13] have presented contour boxplots that enable improved visualizations of uncertainty by drawing spaghetti plots of isocontours or level sets, while considering the order of multivariate data. A two-sided violin glyph was introduced by Höllt et al. [HMZ*14] to express the data distributions at two different locations in space. Voxel bars [MCW*08] constitute another diagram based technique as an extension of pixel bar charts, where groups of 3D voxels are packed into graphical primitives in a 2D coordinate system. Then, their approach emphasizes features and relationships via specific bar colorings.

4.3 Overview

Our method starts with an ensemble of scalar data sets given on a fixed 3D grid structure. We restrict our discussion to 3D Cartesian grids, even though arbitrary grids can be handled in much the same way as described. The data is first structured in a hierarchical manner, computing at each node statistical information of the data values this node represents (see Section 4.5). The visualization is then performed according to the following concepts and principles:

Multi-Charts. First, an abstract volume representation is constructed. It is based on the idea to view a 3D ensemble data set by linearizing the data locations and visualizing the data variations over the ensemble members via bar and line charts, the multi-charts. The data values are drawn as a dense sequence of primitives over a 1D domain, and multiple such sequences are shown at once depending on the size of the plot area. Figure 4.1 shows the basic layout of a multi-chart in the plot area.

Multiresolution Ensemble Summaries. Every bar encodes aggregate statistical information about the data variations across multiple ensemble members in a certain spatial area, and every line speaks for a particular member and allows comparing this member against all others (see Figure 4.1). Since the viewport, into which the chart is drawn, has a fixed resolution, only a limited number of locations can be visualized at once. Therefore, overviews taken from a multi-resolution representation of the 3D ensemble are shown first. According to the information seeking mantra “Overview first, zoom and filter, then details-on-demand” proposed by Shneiderman [Shn96], the user can then zoom into the data to guide the analysis towards those regions showing interesting behaviors.



Vis. Type: Spatial Histogram | Curve: Z-Hilbert Curve | Octree Level: 2 (16x8x8) | Seg. Size: 128 | Lines: 1 | ✓ Rotate

Sub-octree level: Auto [2] | Val. Range: 100 % | Mode: In-Grad.&S. | ☐ Center | Data Set: (All) | Show | ☐ Highlight | Color: Monochromatic | Enhancement: Unfold | Attribute: Range | Len/ attrib | Don' sort | Bar Color:

Data Set Inspector

Data Set	Mean	Std. De
ensemble_1	243.562088	28.0412
ensemble_2	243.4965668	28.0773
ensemble_3	243.3835754	27.6463
ensemble_4	243.2029724	27.6777
ensemble_5	243.3358002	27.9898
ensemble_6	243.0439606	27.5575
ensemble_7	243.5381622	27.6895
ensemble_8	243.5877228	28.0508

Parameters

✓ Highlight selection
Vis. Mode: Parallel (2D)
✓ Align Slices ✓ 3D View
Vis. Type: Mean Slices (XY)
Img. Overlay: Select | Clear
Highlight slice
X (Min): 0
X (Max): 63
Y (Min): 16
Y (Max): 31
Z (Min): 0
Z (Max): 23

Figure 4.1: Multi-chart visualization of a temperature ensemble forecast from the ECMWF Ensemble Prediction System (EPS), ECMWF's operational ensemble weather forecast system [LP08]. The ensemble consists of 51 members of resolution $256 \times 128 \times 64$ each. Each bar in the multi-chart is associated with a distinct 3D subdomain, and encodes the distribution of the ensemble members in this subdomain by means of a histogram. In addition, a few user-selected ensemble members are depicted using polylines. By means of brushing in the multi-chart view (indicated by yellow background color), the user has selected regions where the range over the ensemble members and thus the uncertainty is high. The selected regions are instantly emphasized in the 3D view.

Brushing and Querying. To enable a data-driven analysis, the user can interactively select regions, in which the data values meet certain criteria, and these regions are instantly highlighted in the 3D view. Selecting is performed by brushing or querying in the chart domain, as outlined in Figure 4.1 by the yellow shaded region. To ease the selection process, regions, i.e., their corresponding bars, can be sorted automatically according to increasing or decreasing values of derived statistical measures. The selection can also be manipulated in the 3D view, with the multi-chart view being synchronized accordingly.

Spatial Clustering and Correlations. When analyzing ensembles, one further important operation is clustering of regions, which show similar value distributions over the ensemble members. Such operations can help domain experts to obtain insight into the sensitivity of the simulation processes to the kind of parametrization, and they are especially important to detect short- and long-range correlations in the data. To this end, we provide automatic cluster mechanisms based on the similarity of value distributions (or local histograms), and we arrange bars according to the data similarity in the corresponding regions automatically. Spatial clustering is demonstrated in Figure 4.9.

We now describe the different components of the ensemble visualization technique and their realization. Throughout the following discussion, the main focus is laid on an interactive visual exploration of scalar 3D ensembles, to locate regions in which the data distributions exhibit certain properties. Since locating and picking regions via 3D volume rendering is difficult, we ease the navigation in 3D space via a slice-based visualization of the 3D domain, i.e., the volume is decomposed into a set of slices having a width of one voxel block (see Figure 4.1). In this view, spatial locations and relationships can be perceived far more effectively, and interesting regions can be picked in an intuitive manner.

4.4 Multi-Charts

We assume that the ensemble data set is given on a 3D voxel grid. To provide multi-charts at multiple levels of detail, we partition the voxel grid into approximately equally-sized voxel blocks consisting of roughly $2^\ell \times 2^\ell \times 2^\ell$ voxels, where the level ℓ is selected by the user. $\ell = 0$ denotes the finest level, where each voxel block consists of a single voxel. The multi-chart is then constructed at the granularity of voxel blocks, i.e., each voxel block is represented by a single bar, or by a single point, which is connected to its neighboring points via line segments. Bar and line charts can be displayed together.

4.4.1 Linearizing the 3D Grid

Our multi-chart visualization method is based on linearizing the 3D grid of voxel blocks. We have identified two particular design goals for this linearization: First, the linearization should preserve spatial relationships as good as possible, in particular, linearization-adjacent voxel blocks should also be adjacent in 3D space. Second, the linearization should be consistent over multiple levels of detail, i.e., the eight voxel blocks that constitute a single voxel block at the next coarser level should be adjacent within the linearization. This ensures visual coherence when the user switches between two different levels of detail, in that a bar is decomposed into eight adjacent bars when switching to the next finer level, or eight adjacent bars are merged into a single bar when switching to the next coarser level.

To construct a linearization with these properties, we employ a space-filling curve. In particular, the Hilbert curve is known to preserve spatial relationships well, and due to its octree-based hierarchical construction principle, leads to the desired consistency over multiple levels of detail. Since the standard Hilbert curve is defined on grids with the same power-of-two extent in each dimension, we use a generalized Hilbert curve [HRC08] that supports unequal power-of-two extents in each dimension. To allow for the application of our method to a voxel grid of arbitrary size $n_1 \times n_2 \times n_3$, we proceed as follows: Let $k_i := \lfloor log_2(n_i) \rfloor$, such that 2^{k_i} is the largest power-of-two less or equal than n_i $(i = 1, 2, 3)$. Then, at level $\ell \in \{0, 1, ..., \min\{k_1, k_2, k_3\}\}$, the voxel grid is decomposed into $2^{k_1-\ell} \times 2^{k_2-\ell} \times 2^{k_3-\ell}$ voxel blocks. Thus, the voxel block grid has power-of-two extents, allowing for the construction of a generalized Hilbert curve. Let blocks and voxels be identified by 3D integer indices, counting from zero in each dimension. Voxel block $(\tilde{x}_1, \tilde{x}_2, \tilde{x}_3)$ then consists of all voxels (x_1, x_2, x_3) satisfying

$$\left\lfloor \tilde{x}_i \cdot 2^\ell \cdot \frac{n_i}{2^{k_i}} \right\rfloor \le x_i < \left\lfloor (\tilde{x}_i + 1) \cdot 2^\ell \cdot \frac{n_i}{2^{k_i}} \right\rfloor \qquad (i = 1, 2, 3).$$

The resulting voxel blocks are almost of equal size (along each direction, their size varies by at most one voxel), and exhibit compact shape (the ratio of the largest to the smallest edge is at most two). Per construction, subsets of $2 \times 2 \times 2$ voxel blocks constitute a single block at the next coarser level. At level $\ell = 0$, the blocks consist of between one and eight voxels. To allow the user to explore the data set also at voxel granularity, an additional level $\ell = -1$ is added, where voxel blocks consist of individual voxels. The linearization at this level is derived from the linearization at level $\ell = 0$, with each voxel block being further linearized in lexicographical order.

4.4.2 Chart Layout

Now, the multi-chart is generated by displaying bars and line segments corresponding to voxel blocks in the order imposed by the described linearization. To fully utilize the size of the screen, the chart is displayed in a set of multiple rows. At the beginning of the visual exploration process, the number of rows is set to eight, and we automatically select a level of detail such that the number of bars per row is at most 256.

For navigation within the chart, we provide zooming and panning by operating the mouse. Zooming can be performed independently for each axis, or simultaneously for both axes. To simplify the navigation, the current position and extent of the viewport are displayed in an overview window (see Figure 4.1). The user can prescribe a specific level of detail, or use automatic level of detail selection coupled to horizontal zooming. The latter is implemented by automatically switching to the next finer (coarser) level of detail, whenever the bar pixel width exceeds (falls below) a certain threshold (we use a threshold of 64 and 8 pixels, respectively). To achieve visual coherence, the width of the bars is decreased (increased) by a factor of eight, when switching to the next finer (coarser) level, such that a bar and its associated set of refined bars occupy the same position. The bars are displayed using a uniform width, except for level $\ell = -1$, where we have to use a variable width in order to compensate that the bars at level $\ell = 0$ are decomposed into a variable number of bars.

To further improve the visual coherence, when switching between levels of detail during zooming, we employ a background pattern based on vertical stripes that are aligned with the bars (see Figure 4.2). The stripes are colored alternately using gray at two different intensities, and are used to indicate the sets of refined bars resulting from the decomposition of the bars at the previous coarser level. Consider the case of zooming into the chart (zooming out is similar). When switching to the next finer level, the stripes initially remain unmodified, such that the sets of refined bars become visible. Only when further zooming into the chart, the stripes are also refined using a gradual transition. This transition is completed before switching to the next finer lever again, such that the process restarts from the beginning.

While in our current implementation, we consider only data sets on 3D voxels grids, in principle our method can also be applied to unstructured grids. This can be achieved by using a hierarchy of rectangular grids to partition the 3D domain at multiple levels of details, such that vertices of the unstructured grid lying in the same grid cell are assigned to the same subset. Our method then is applied to these subsets, i.e., the subsets replace the voxel blocks in the previous description. This construction principle also works, when the unstructured grid is time-dependent, since the rectangular grid remains fixed.

4.5 Multiresolution Ensemble Summaries

To guide the visual exploration of the ensemble data set, we provide two different types of multi-charts.

4.5.1 Analyzing the Quantitative Distribution of Uncertainty

Our first type of multi-chart (see Figure 4.3, top) is devoted to analyzing how uncertainty is quantitatively distributed over the 3D domain. For each voxel of the data set, we compute the range (i.e., the difference between the maximum and the minimum value) over the set of ensemble members, and use this value to quantify the level of uncertainty at that voxel. In order to reduce the influence of outliers, the quantile range between the p and $(1 - p)$ quantiles can be used alternatively ($p \in [0, 0.5)$). Then, we depict the distribution of the

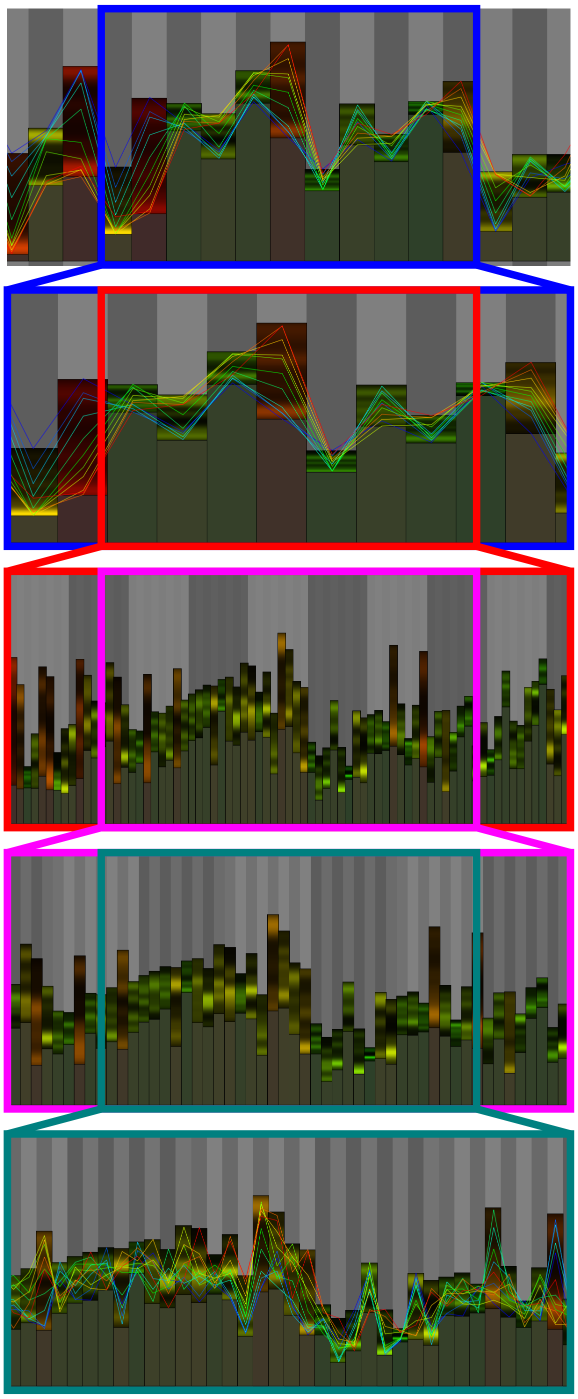


Figure 4.2: From top to bottom: Zooming into the multi-chart. The bars are automatically refined, when their width exceeds a certain threshold. To improve visual coherence, a vertically striped background pattern is employed to illustrate set of refined bars.

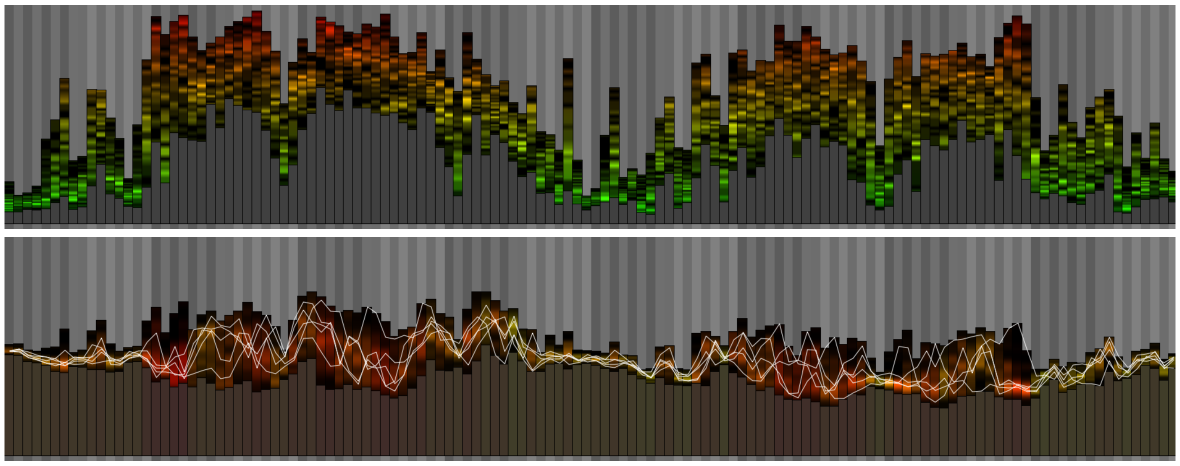


Figure 4.3: Two different types of multi-charts are provided. The bars are encoding histograms, either showing per-voxel statistical quantities (here: the range over all ensemble members) within the sub-region associated with each bar (top), or the distribution of the ensemble members, based on the average of each ensemble member within each sub-region (bottom). In the latter type of multi-chart, a user-selected subset of the ensemble members can be additionally displayed via horizontal line segments or polylines. In the top and bottom image, corresponding bars show the same sub-region.

range values within each voxel block by means of a histogram encoded into a bar as follows: The histogram is determined by means of kernel density estimation [Ros56, Par62], using 128 bins and Gaussian kernels. The vertical extent of the bar reaches from zero to the maximum of the range values in the considered block. The histogram is visualized by using the histogram density values (scaled by a user-specified factor) as the color intensity along the vertical range of the bar. Between zero and the minimum range value, a constant color intensity of 0.25 is used. In this way, the minimum and maximum range values become clearly visible.

In particular, the color intensity $I \in [0, 1]$ as a function of the delta value δ is determined from the density function f according to

$$I(\delta) = f(\delta)/\max\{s, \max_{\delta}\{f(\delta)\}\},$$

where s is a user-defined scaling constant. By including the maximum of the density function in the denominator, oversaturation is avoided.

Notice that by showing the distribution of range values (rather than for example only depicting their average or maximum), each voxel is accurately represented at all levels of detail, and the method is robust in the context of outliers. To support the user in rapidly identifying regions with low and high uncertainty, we additionally color-code the range value by means of a linear green-red color scale, where green and red denote low and high uncertainty, respectively. Also, note that this color-coding is redundant, in that the range value is already determined by the distance from the zero line.

In addition to the range value, it is also possible to visualize per-voxel-block histograms of other statistical quantities, such as the mean or the standard deviation when analyzing ensemble data sets that are known to follow a Gaussian distribution.

4.5.2 Analyzing Distribution and Relationship Among Members

While the first type of multi-chart can effectively reveal the distribution of uncertainty over the 3D domain, the second type (see Figure 4.3, bottom) is designed for providing a detailed insight into the particular distribution of the ensemble members, as well as studying the relationships among a subset of the members, or between a member and the entire ensemble. For each ensemble member, we first compute the average of the values as well as the difference (here referred to as delta value) between the maximum and the minimum value within each voxel block. The delta value describes the spatial variance of each ensemble member in the considered block. Our visualization approach is then based on representing each ensemble member by its average value within each voxel block. Clearly, using average values can lead to misinterpretations if the spatial variations are too high. In this case, it is necessary to perform the analysis at a spatially higher resolution.

We visualize the distribution of the ensemble members by plotting the distribution of the corresponding average values within each voxel block by means of a histogram encoded into a bar. We proceed in the same way as for the first type of multi-chart, i.e., we again construct the histogram via kernel density estimation, which is then depicted by means of adapting the color intensity along the vertical range of the bar. To incorporate the delta values, for each block, we compute the maximum delta value over the set of ensemble members, and depict this maximum by means of color-coding of the entire bar using a linear green-red color scale. Using green and red for low and high delta values, respectively, the red color indicates that it is necessary to switch to a finer level of detail, in order to perform an accurate data analysis in the respective sub-region associated with the bar.

In addition to showing the distribution of the ensemble members via bars, we provide the possibility to visualize the ensemble members—or a user-specified subset—via lines, which are laid over the bars. We have implemented two variants (see Figure 4.4): Depicting ensemble member using horizontal line segments, or using polylines. Optionally, color-coding of the lines can be used to support tracking of individual ensemble members or groups of ensemble members throughout the diagram. Colors can be assigned automatically by member id using

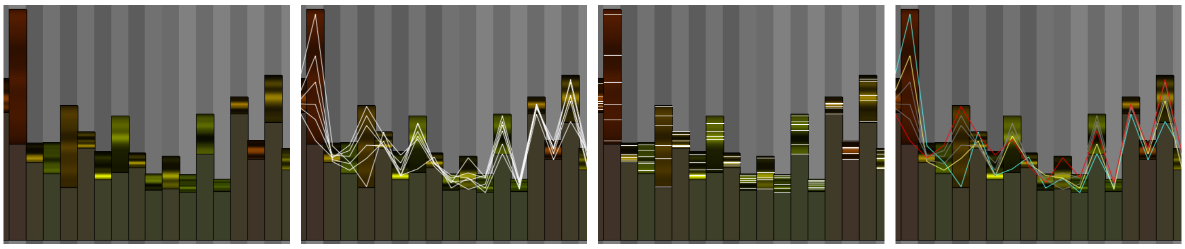


Figure 4.4: Visualization of the distribution of the ensemble members. From left to right: Histogram only, histogram and polylines, histogram and horizontal line segments, polylines with color-coding of individual members.

a discrete color table, or manually by mouse operation. We further provide the option to encode the delta value using a linear blue-red color scale, where blue and red denote low and high delta values.

It can be seen that the polyline-based approach facilitates tracking of individual ensemble members throughout the multi-chart. In contrast, it is almost impossible to track individual members in the visualization based on horizontal line segments. However, the latter approach can be employed as an alternative to histograms for visualizing the distribution of the ensemble members, when the ensemble consists of only a few members. A drawback of the polyline-based approach is that line segments can have a very high slope, when the bar width is small, leading to visual cluttering. We address this issue by automatically hiding the polylines when the bar width is less than 16 pixels.

To facilitate the interactive exploration of ensemble data sets with a large number of ensemble members, the ensemble summaries are precomputed and stored on disk. In particular, for each voxel block (on all levels), we precompute the histogram of the per-voxel ranges and the histogram of the per-member averages, as well as statistical quantities such as the minimum/maximum/mean of the per-voxel ranges, the minimum/maximum/mean/standard deviation of the per-member averages, and the maximum of the delta values. In total, the ensemble summaries require about 1 kB per voxel block (using 128 bins for the histograms).

Depicting individual ensemble members requires the per-member average values within each voxel block. These values are stored in a separate data file, consecutively for each voxel block. Note that on the finest level, where each voxel block consists of a single voxel, the initial ensemble data set is stored.

The ensemble summaries are completely loaded into main memory at the beginning of the interactive visualization session. This is required to enable the sorting of voxels blocks according to specific criteria, as described in the following section. It is worth noting that the

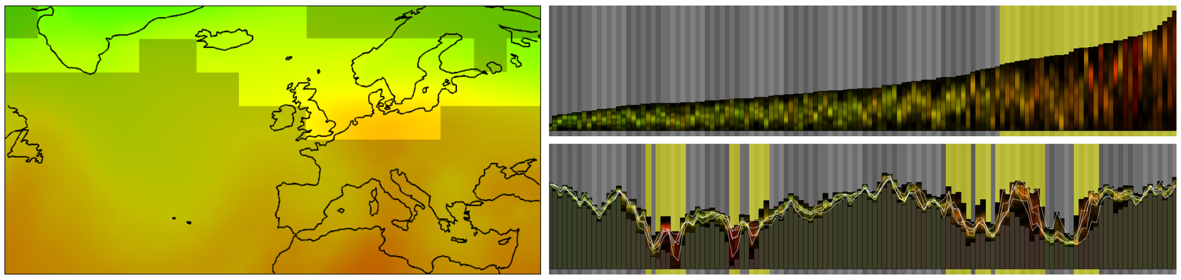


Figure 4.5: Sorting of the bars to support selection of regions with similar characteristics: After sorting of the bars, the user brushes into the multi-chart (right, top). The selection remains active after the bars have been rearranged into their original order (right, bottom). The respective selection is also shown in the 3D view (left). Here and in the following figures, only a single slice of the 3D volume is shown for illustration purposes.

memory size of the ensemble summaries is independent from the number of ensemble members, but depends on the spatial resolution of the data set. For very high-resolution data sets, a viable option to reduce memory requirements is to skip pre-loading of histograms for the very finest levels (which however disables sorting of voxel blocks according to histograms on these levels). In contrast, the per-member average values are fetched from disk on demand, i.e., only data values that are currently visible in the viewport are loaded into memory. To hide disk access latencies, we have implemented a prefetching scheme based on rectangular prefetching regions around the viewport (one per level of detail).

4.6 Brushing and Querying

The user can brush arbitrary bars interactively in the multi-chart view, and the spatial regions corresponding to the selected bars are instantly highlighted in the 3D view. Brushed bars are highlighted by a yellow background as shown in Figure 4.5 for the ECMWF temperature ensemble forecast, introduced in Figure 4.1.

In addition, we provide the option to sort the bars with respect to different statistical measures, such as mean value and standard deviation of the average values per ensemble member in each region, or the maximum/minimum/mean of the per-voxel statistics in each region. Sorting with respect to histogram similarity will be described in the next section. By brushing in the sorted representation, the user can select all relevant spatial regions in an easy way. This is shown in Figure 4.5 (right, top), where the spatial regions exhibiting highest difference between any two ensemble members are selected. Note that in this chart, the bar height encodes only the range (difference between maximum and minimum), rather than the

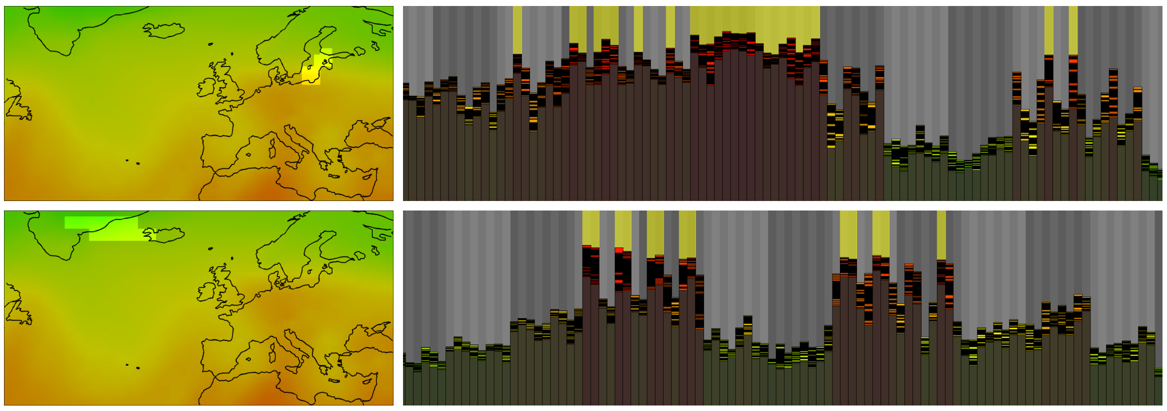


Figure 4.6: Top and bottom row: After zooming into two of the regions determined in Figure 4.5, visualizing histograms of per-voxel ranges reveals areas of particular high uncertainty.

maximum over all ensemble members. In the 3D view, the selected regions are highlighted instantly.

When switching back to the order imposed by the Hilbert curve, as shown in Figure 4.5 (right, bottom) (now the bar height encodes the maximum), the brushed selection remains active (see yellow background), and, in the current example, the selected regions distribute on roughly four clusters in the abstract multi-chart view. Due to the good preservation of locality of the Hilbert curve order, these clusters correspond to spatial clusters, as shown in Figure 4.5 (left). One can clearly see that a higher degree of uncertainty occurs in regions exhibiting lower mean values. Interestingly, although not shown here, this property holds true for the whole domain.

When relating each cluster in the multi-chart view with its corresponding geographical area on the map, one finds that the third cluster from the left roughly embodies the region containing the North Sea and Baltic Sea. To further analyze the selected regions, we visualize the per-voxel range and zoom into the third (Figure 4.6, top row) as well as first and second (Figure 4.6, bottom row) clusters. Since the delta values are relatively high for these regions, as indicated by the bars' red color, we switch to a chart depicting histograms of per-voxel ranges. We then select regions where the uncertainty is particular high. This guides us to a region containing two larger islands in the Baltic Sea, and two regions in the coastal area between Greenland and Iceland.

To select regions based on specific data values or statistical measures, the user can also use select queries. This is realized by using expressions, such as *Min* and *Max* on all the different statistical measures. *Min* and *Max* queries give all bars and corresponding spatial regions

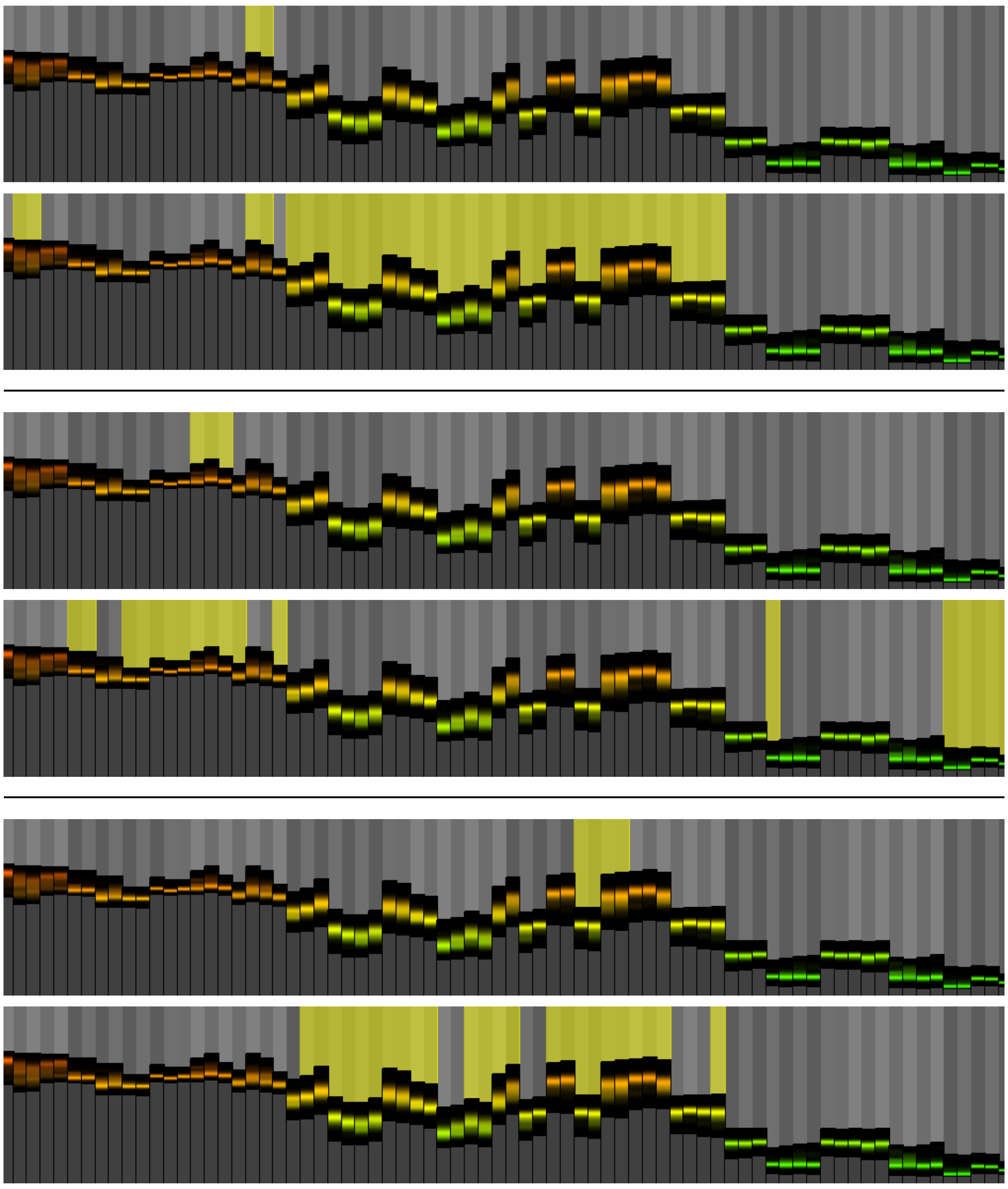


Figure 4.7: Selection of bars by querying operations. Each pair of images, from top to bottom: The user selects some bars (top), and asks for all other bars having a range that is higher than the minimum / lower than the maximum / within the interval given by the minimum and maximum of the ranges of the selected bars (bottom).

where the measures are respectively below and above a specified value. Range queries allow selecting regions exhibiting measures in specific intervals. Furthermore, the user can select particular members and let them highlight as a line chart in the multi-chart view. Some typical query operations are shown in Figure 4.7.

4.7 Spatial Clustering and Correlations

Besides a visual ensemble analysis relying purely on the analysis of statistical measures in selected spatial regions, an important requirement is to analyze the (relative) behavior of the data values among selected regions of interest or different ensemble members. The rationale behind this is to quickly identify regions showing similar data distributions and data variations over the ensemble members, and to reveal dependencies between the data values in different regions. To achieve this, we have integrated techniques for grouping and sorting of spatial histograms. In addition, we consider the correlations between the data value in different spatial regions to analyze their behavior relative to each other.

4.7.1 Histogram Clustering

Per-region histograms are computed as described in Section 4.5, and clustered based on their similarity using the k-means algorithm [Llo82]. The similarity of two histograms $h_1, h_2 : \{1, \ldots, n\} \to \mathbb{R}_0^+$ with $n \in \mathbb{N}$ bins is computed by their mean squared distance as

$$\frac{1}{n} \sum_{i=1}^{n} \left(h_1\left(i\right) - h_2\left(i\right)\right)^2 .$$

The user can select either the number of clusters to be generated or a similarity threshold, so that only histograms having higher similarity than indicated by this threshold are grouped.

To find an initial guess of k means in order to start with the k-means algorithm, we proceed as follows. We begin with an empty set of clusters and a fixed similarity threshold, and we then iterate over all histograms. For every histogram, the cluster with the least difference between the considered histogram and the respectively first histogram assigned to that cluster is identified. If the difference is smaller than the threshold, the histogram is assigned to that cluster. Otherwise, or if no cluster exists at all, a new cluster is generated and the histogram is assigned to it. This step is repeated until all histograms have been assigned. If the number of generated clusters is greater (less) than the requested number of clusters, we divide (multiply) the threshold by 2 and start clustering all over again. This is repeated until the number of clusters has crossed the desired number. The respectively first histograms that were assigned

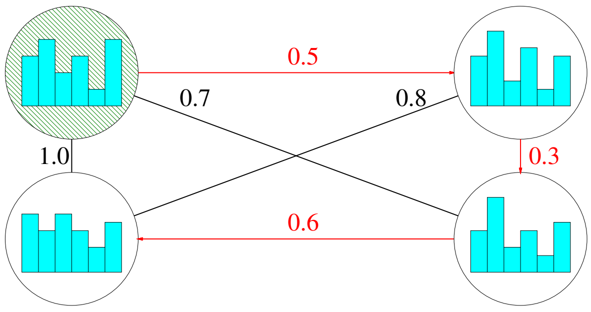


Figure 4.8: Ordering histograms by approximating a TSP solution. We start with the node marked as green and calculate the distance to all of its neighbors. We then proceed with the neighbor, to which the distance is minimal, here 0.5. This is then repeated for the next node, until no more unvisited nodes are left. The path obtained by this algorithm is drawn in red.

to each cluster are used as initial cluster centroids in the upcoming k-means algorithm. Note that the desired number of clusters can be selected and changed by user at any time.

After running the standard k-means algorithm (see Lloyd [Llo82]), the histograms in each cluster are sorted by approximating the solution of a traveling sales person problem (TSP) in a weighted undirected complete graph. For each cluster, the histograms in this cluster comprise the nodes of the associated graph, and the similarities between histograms are used as edge weights. An illustration of the graph of a cluster consisting of 4 histograms is depicted in Figure 4.8. For every cluster, the algorithm starts with an empty list and selects the first node belonging to the first histogram in this cluster, marked as green in the figure. The following operations are then repeated until no more unvisited nodes are left in the cluster: For the currently visited node, we compute the distance to all unvisited nodes and append the node, to which the distance is minimized to the list. This node is then marked as visited and will be traversed next. As a result, the list contains an approximate sorting according to histogram similarities.

The benefits of our clustering approach are demonstrated in Figure 4.9. In our example, 10 clusters have been determined. To make the visualization easier to understand, a unique color is assigned to each cluster, and all bars are scaled to a uniform height. The user has selected some regions within the slice-based 3D view (Figure 4.9, top left). It can be observed that the bars fall into two different clusters (Figure 4.9, top right). If one picks all bars in these clusters (Figure 4.9, bottom right), one obtains a remarkable result, which is depicted in Figure 4.9 (bottom left): Clearly, it can be seen that mostly coastal areas have been selected, meaning that these regions feature similar distributions of the ensemble members.

Let us now discuss another way of classifying histograms. For this consider two histograms of

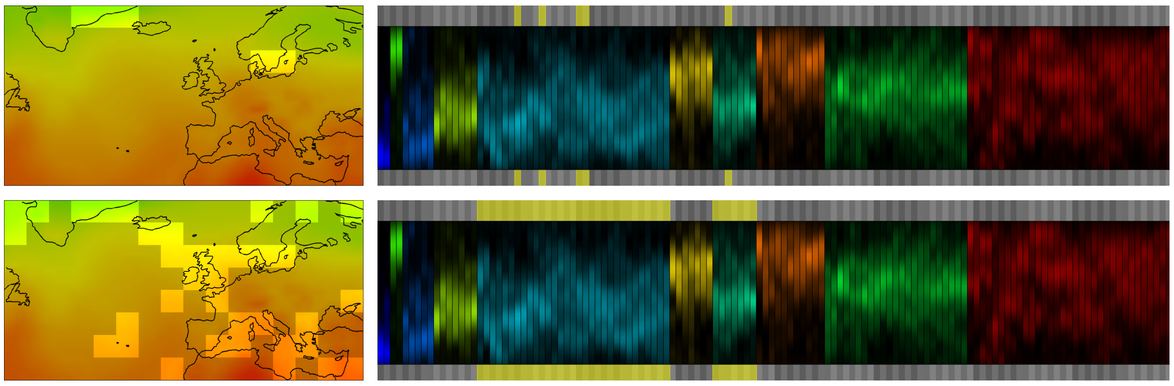


Figure 4.9: Selection based on clustering of bars according to histogram similarity. After brushing into the 3D view (top left), the selected regions fall into two clusters (top right). By selecting the remaining bars from the two clusters that contain initially selected bars (bottom right), regions with similar distributions are identified (bottom left).

which each has exactly one peak but at different positions. According to our previous method based on the mean squared distance, these histograms would be classified as completely different, although they might be seen as similar in the following sense: By translating one histogram such that the peak fits the peak of the other histogram, the mean squared distance would vanish. In other words, such histograms are congruent. To compute a quantifiable value of congruence, we iteratively translate the first histogram step by step with regard to both directions, and calculate the mean squared distance concerning the translated histogram and the second histogram for each iteration. Since the histograms are created by kernel density estimation, we can apply translations of any given extent. Finally, the minimum of all distances is returned as the result. In this way, a translation-independent clusterization is obtained.

4.7.2 Correlations

Besides a statistical analysis focusing purely on the data variations in certain regions, an important task is to find regions in which the data values show certain correlations to each other. While in principle, local correlations can be visualized via glyph-based approaches, e.g., as proposed by Pfaffelmoser and Westermann [PRW11], visualizing long-range interdependencies in 3D is challenging. It is, on the other hand, especially the existence of such long-range interdependencies, which are important in a number of applications. For instance, in meteorology, local features are often dependent on distant phenomena. To analyze such dependencies, we have built into our tool means to compute a) the correlations between the

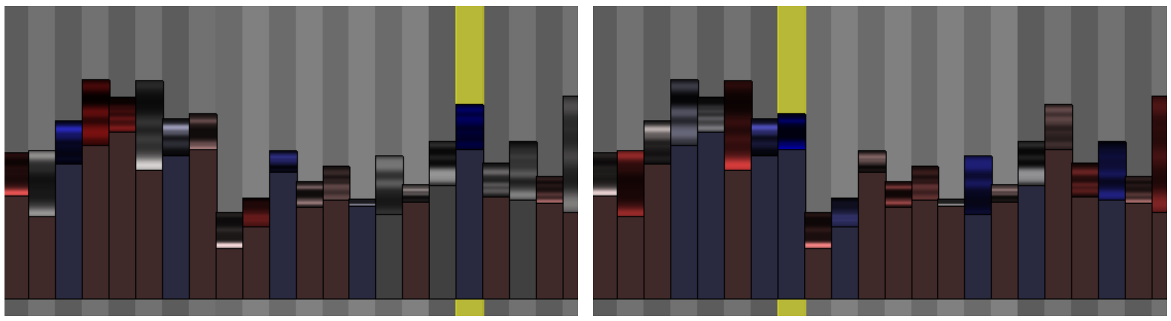


Figure 4.10: Visualization of the correlations between the sub-regions associated with the bars based on per-member average values. Left and right image, respectively: After picking a single bar (highlighted in yellow), the correlation between this sub-region and each other sub-region in the viewport is computed and visualized by color-coding of the bars.

average member values in the region represented by one selected bar and all other regions represented by bars in the current view, and b) the correlation between the variation of the data values in two user-selected sub-regions over all members. In the former case, the correlations are instantly visualized by mapping the correlation values to the colors of the involved bars, as depicted in Figure 4.10, using a red-white-blue color scale, where red/white/blue denote negative/no/positive correlation, respectively. Furthermore, we use the correlation as a distance metric for computing clusters of regions in which the data values are strongly correlated using the k-means algorithm. Here, we exploit the fact that for correlations sufficiently close to 1, the correlation property is transitive, thus making the metric an equivalence relation. Regions not strongly correlated will fall apart into single-element clusters.

Correlation computation always yields a numerical value in the interval $[-1, 1]$, serving as a measure of the strength of the relationship between the values in pairs of regions. In particular, we make use of the Pearson's correlation coefficient to measure the strength of a linear association. Thus, negative and positive values indicate an inverse and positive correlation. To generate robust results even under weak assumptions with regard to the distribution, we utilize a variant of the quadrant correlation coefficient as proposed by Blomqvist [Blo50]. We have modified this algorithm slightly, since it relies on the signum function, which turned out to produce inconclusive results in many of our cases. Therefore we make use of a weakened signum function as described below.

Let $I = \{1, \ldots, m\}$ denote an index set of $m \in \mathbb{N}$ ensemble members and let A, B denote two distinct spatial areas represented by bars in our visualization. Also, let us denote the value distribution of the ensemble with respect to areas A and B by $h_A : I \to \mathbb{R}^{|A|}, h_B : I \to \mathbb{R}^{|B|}$.

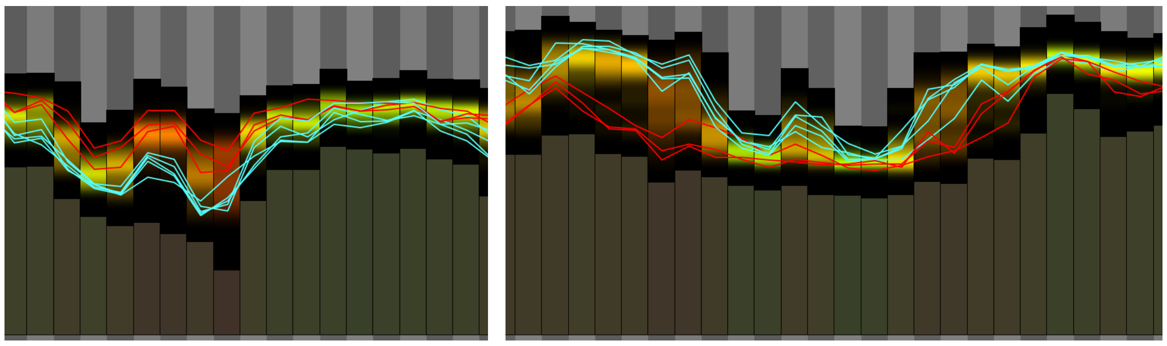


Figure 4.11: Two areas (left and right image, respectively) that were identified as being inversely correlated. To demonstrate the inverse correlation, several ensemble members are depicted by means of colored polylines. Whereas red members have higher values than blue members do in the left area, the situation is vice versa in the right area.

Then, we obtain the modified quadrant correlation coefficient as

$$\frac{1}{m}\sum_{i=1}^{m} wsgn\left(h_A(i)-\tilde{h}_A\right)\cdot wsgn\left(h_B(i)-\tilde{h}_B\right).$$

Here $\tilde{h}_A, \tilde{h}_B$ describe the component-wise median of h_A, h_B, and we define $wsgn : \mathbb{R}^k \to [-1,1], k \in \mathbb{N}$ as the weakened signum function

$$wsgn(x) = \begin{cases} 1 - neg(x)/pos(x) & \text{if} \quad pos(x) > neg(x) \\ -1 + pos(x)/neg(x) & \text{if} \quad neg(x) > pos(x) \\ 0 & \text{else} \end{cases},$$

where, $pos(x)$ and $neg(x)$ denote the number of positive and negative entries of the vector x, respectively.

To demonstrate the kind of information that can be revealed by the proposed correlation visualization, let us take another look at the temperature forecast ensemble introduced in Figure 4.1. Here, we focus on the region containing Greenland (first block of selected bars from the left in Figure 4.5, bottom) and the region containing the North Sea and Baltic Sea (part of the third block of selected bars). To study the interdependencies between the ensemble members in these regions, the user selects these regions and asks for the correlation coefficient. In the current example, a correlation coefficient of -0.45 is computed, meaning that the distributions in both regions are inversely correlated. To illustrate this result, in Figure 4.11 a few representative ensemble members are shown.

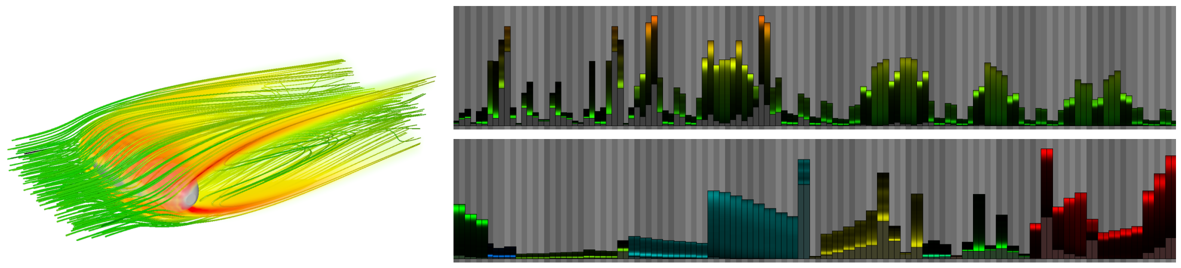


Figure 4.12: Ensemble data set for an incompressible 3D Navier-Stokes fluid simulation. Left: A single member is visualized by streamlines and volume rendering. Right, top: Multi-chart, showing the distribution of the ensemble members via histograms. Right, bottom: Bars are clustered according to histogram similarity.

4.8 Further Results

In addition to the ECMWF temperature ensemble, used to demonstrate the effects of our ensemble visualization technique, we now present further results on the basis of additional scalar ensembles.

4.8.1 Incompressible Fluid Flow Simulation

We begin with an ensemble featuring an incompressible fluid flow evolving around an ellipsoid obstacle, which was numerically simulated on a $145 \times 49 \times 49$ Cartesian grid using the Navier-Stokes equations. 56 simulation runs were performed using slightly different viscosities. The vorticity magnitude produced by each simulation run after the same simulation time was written out as the scalar ensemble field. A visualization of a single time step is shown in Figure 4.12 (left).

For visualizing this ensemble, we start with an overview first to isolate regions, where the data values show an outlier behavior compared to other regions. The overview is shown in Figure 4.12 (right, top), where the distribution of the ensemble members is shown in a multi-chart. It can be clearly seen that there are regions where the vorticity varies significantly over the ensemble members, whereas in other regions, it changes only to a very small extent. Another interesting aspect, which can be recognized by this view, is that small variations occur only, where the vorticity is relatively small as well. In other words, there are no regions where the ensemble exhibits constant high vorticity throughout all members.

Next, we study the distribution of vorticity among regions corresponding to the bars and try to discover striking spatial relations between these distributions. For this, we order the bars in such a way that bars with similar histograms are clustered. To make it easier for the user to

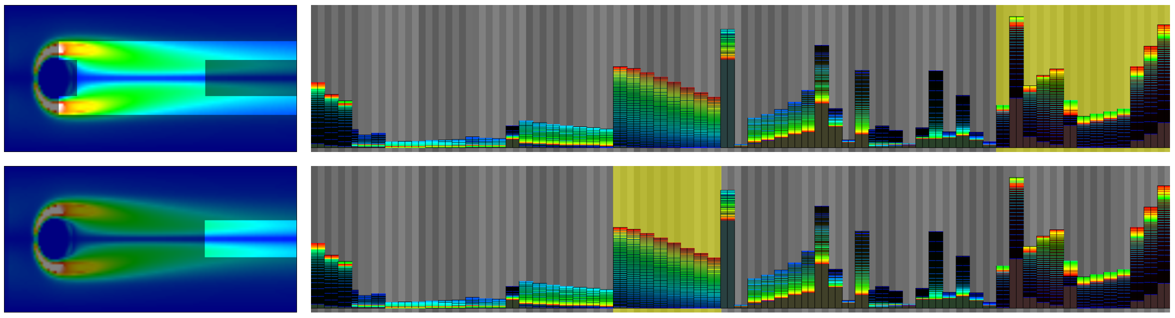


Figure 4.13: Top row: Selection of the red cluster from Figure 4.12 via brushing. The selected region is highlighted in the 3D view. Bottom row: Selecting the emphasized region in the 3D view yields a part of the cyan cluster. In both rows, the individual ensemble members are additionally depicted by horizontal line segments, which are color-coded according to ensemble member id.

visually recognize this order, a unique color is assigned to the bars of each cluster. The result is shown in Figure 4.12 (right, bottom). We can now analyze common features of specific clusters. For instance, the cluster colored red features a peak at higher values, meaning that vorticity in this region is high throughout most of the simulations. By looking at individual members, color-coded from blue to red, we find that a low vorticity occurs in this region only for a few members. After selecting this cluster by brushing, we see the associated regions in spatial context. As depicted in Figure 4.13 (top row), one can observe that these regions are located around the obstacle and in an outer region behind the obstacle. Note however, that a certain region behind the obstacle is not covered by that cluster. When we select this region, we discover that it belongs to a cluster, which has no clear peak. Instead, the ensemble members are distributed evenly with a steadily increasing vorticity, as can be seen in Figure 4.13 (bottom row) from the color-coded line chart.

4.8.2 Cosmic Density Field

The third scalar ensemble field we analyze is a cosmic map comprising 40,000 different, yet possible versions of the observable universe. Each version constitutes a scientifically plausible cosmic density field on a 256^3 voxel grid according to the available probabilistic model. A volume rendering of one of these cosmic maps is shown in Figure 4.14 (left). For further information let us refer to Jasche et al. [JKLE09]. For this data set, the ensemble summaries occupy about 16 GiB, the per-member average values (which also include the initial ensemble data set) about 2.8 TiB.

Our technique enables the user to visualize the entire ensemble, once the preprocess to generate the multiresolution hierarchy has been performed. In the preprocess, all statistical data which

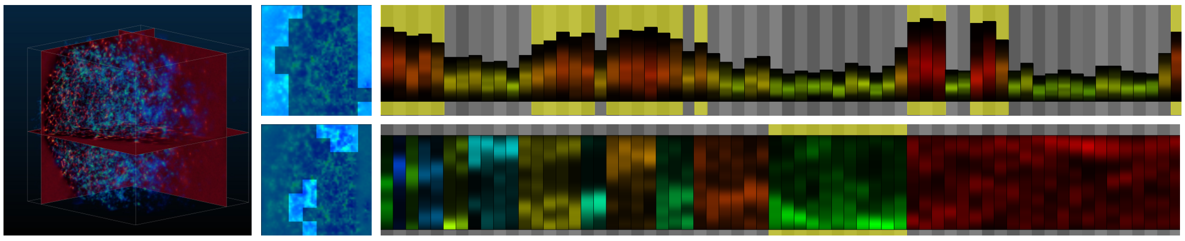


Figure 4.14: Application of our multi-chart visualization method to study a large-scale data set, consisting of 40,000 possible maps of the universe. Left: Volume rendering of a single ensemble member. Image courtesy of Jasche et al. [JKLE09]. Top right row: Multi-chart showing the distribution of the ensemble members. By selecting bars corresponding to areas with high uncertainty, it can be observed that these areas are primarily located in outer regions of the domain. Bottom right row: After clustering the bars according to histogram similarity, a cluster is selected, where the ensemble members have low values (here: density), yielding two distinct regions.

is required to pursue the proposed visual ensemble analysis is pre-computed and saved. During the visualization the pre-computed data are loaded into memory on-demand, according to the currently selected visualization option. This enables the user to explore even large and high-resolution ensembles at interactive rates.

Since this data set is rather extensive we restrict ourselves to analyze only a small region in order to demonstrate the potential of our method.

In the analysis, we start by visualizing the uncertainty at a certain level of detail (see Figure 4.14, top right row). Note that in this figure, the bar height encodes the range over the ensemble members (difference between maximum and minimum) rather than the maximum. By selecting bars indicating high uncertainty and linking to the 3D view, it can be seen immediately that the uncertainty almost only occurs in the outer regions of the domain. This conforms to the way the data set was generated. As far less observational data was used for the galaxy reconstruction in the outer regions than in the inner ones, much higher uncertainties are introduced in the outer regions.

We further cluster bars by histogram similarity and then pick a cluster, where the highest density occurs largely on the bottom of each bar (see Figure 4.14, bottom right row). In other words, most values in that region lie at the lower bound with a small ratio of outliers, which accounts for the low intensity on the top of each bar. Since these bars belong to two distant regions in space, we are now interested in finding out whether there exists a certain correlation between these two regions. Therefore, we zoom into the region of selected bars until the bars decompose into smaller bars at the next finer level. After picking both regions and asking for the correlation between them, one finally discovers that the ensemble

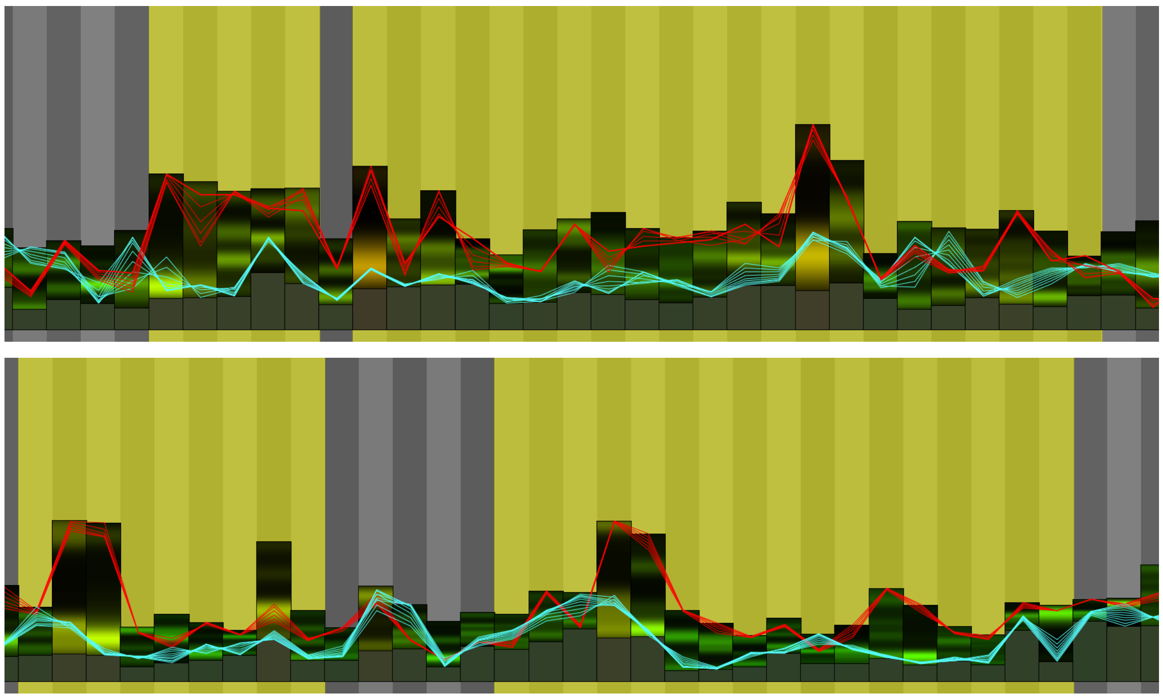


Figure 4.15: Two areas (top and bottom image, respectively) in the cosmos ensemble data set that were identified to be positively correlated. Several ensemble members are exemplarily depicted. In both areas, red members exhibit higher values than blue members do.

members at these two regions are positively correlated. This is also illustrated by a few chosen members as demonstrated in Figure 4.15. Note that it will be a very daunting task to find such relations by conventional methods like volume rendering. This is because of the occlusion effects inherent to such methods and, in particular, because displaying even a small part of the current ensemble becomes virtually infeasible for the user to recognize.

4.9 Conclusion and Evaluation

In this chapter, we have introduced a novel technique for the interactive visual exploration of large 3D scalar field ensembles. When using this technique, we found that users spend most of their exploration time in the abstract view, comprising bar and line charts to convey region-specific statistical data measures. Since the abstract view does not suffer from occlusion effects, prominent characteristics of the data distributions as well as interdependencies between the data values in different regions are very effectively communicated to the user. By linking to a view, where the spatial context of specific characteristics is shown, our technique enables an effective exploration of the whole ensemble space. We consider it as an outstanding feature of our technique that it does not limit the number of ensemble members to be analyzed. In our opinion, this has the potential of significantly changing the way experts investigate 3D ensembles in the future.

We have developed this method in close cooperation with a few domain experts from meteorology and astrophysics, however, we have not conducted a formal user study to assess the technique's effectiveness. Nevertheless, in the experiments we have accomplished so far, and in which the domain experts participated actively, we have made some interesting observations that we consider as noteworthy: Firstly, the linear organization of spatial locations and corresponding sample sets did not introduce any mental barriers in the analysis process. At the beginning, we were not so sure that the abstraction from 3D physical space would be accepted, yet it turned out that all users were immediately willing to accept this abstraction—and the associated loss of spatial coherence in this abstraction—and perform their investigation in the multi-chart view. Even more astonishingly, all users spent most of the analysis time in the abstract view, while linking to the 3D view only very rarely to get an overview of the spatial locations of the regions that they have actually analyzed. As stated by the domain experts, the reason for this behavior was due to the fact that during multi-chart-based analysis, the user essentially focuses entirely on the analysis of the variability of the ensemble members regardless of the spatial context, where these variations occur. Only at the very end of the analysis, when remarkable relationships between the members were identified in the abstract view, they took a further look at the spatial view in order to discover the corresponding regions in space.

Finally, in our experiments, all users confirmed the effectiveness of the multi-chart view, mainly for the reason that they could not envision alternative techniques allowing for an in-depth sample analysis at such a fine granularity. Several domain experts had used 3D fields—or cross-sections through these fields—containing mean values and standard deviations to analyze the ensembles. Yet there was strong agreement that this kind of analysis does simply not allow any meaningful insights into the sample distributions across the ensemble members. The experts found it especially appealing that sub-regions exhibiting strong disparities have already been detected in the overview, and that by zooming into these areas, even the distribution characteristics of the sample values across the members could be studied. All users confirmed that the richness of specific statistical features that could be retrieved, appeared to them as an outstanding functionality, especially due to the fact that all selections could be steered interactively. Regarding the spatial view, there was an agreement that a simultaneous visualization of correlations in the abstract and the spatial view would amplify the value of the proposed analysis technique to a great degree.

Different aspects of our approach can be investigated in future research: By using compression and parallelization strategies, handling large and high-resolution ensembles could be improved. In particular, by bringing our technique into the cloud, remote computing capacities for statistical analysis would be on hand. Since our technique condenses the data effectively and requires only limited information for graphical display, it seems very well suited for remote cloud environments. Furthermore, the extension to time-varying ensembles will be challenging, as well as the integration of abstract parameter views to support a visual exploration of multi-parameter data. By adapting the bar layout accordingly, our approach might also be suitable for visualizing vector field ensembles. Although designing an adequate layout for vector data will presumably be challenging task, it might be worthwhile to continue research in that direction.

5 Silhouette-Based Visualization of 3D Isosurface Ensembles

In this chapter, we introduce a novel visualization technique for ensembles of isosurfaces in three-dimensional space by drawing screen-space silhouettes. Our approach is spatially coherent and maintains the major shape of the surfaces. This chapter is largely based on our publication:

Demir I., Kehrer J., Westermann R.: Screen-space Silhouettes for Visualizing Ensembles of 3D Isosurfaces. In *Proc. IEEE Pacific Visualization Symp. (Visualization Notes)* (2016). `doi:10.1109/PACIFICVIS.2016.7465271`. [DKW16]. © 2016 IEEE.

5.1 Introduction

Scalar field ensembles play an important role in many areas of science and engineering. Recall that ensembles are typically generated by $N \in \mathbb{N}$ repeated simulation runs, where different input models or parameters are used in each run. For 3D scalar fields, this can be described formally as a mapping $\{1, ..., N\} \times \mathbb{R}^3 \to \mathbb{R}$. Visualizing 3D scalar field ensembles is a challenging task, since a large amount of data at each spatial location has to be conveyed to the user, such that important information is neither lost nor occluded.

Different solutions exist to approach this problem. For instance, volume rendering is an effective technique to visualize the information stored in 3D scalar fields, especially with the help of transfer functions. By interactively adjusting this function, the shape of relevant features can be revealed. In this context, isosurfaces are of particular interest, i.e., locations exhibiting a constant value [Han05]. However, volume rendering suffers from occlusion effects, even if only a single scalar field is rendered. Consequently, this effect is amplified, when multiple ensemble

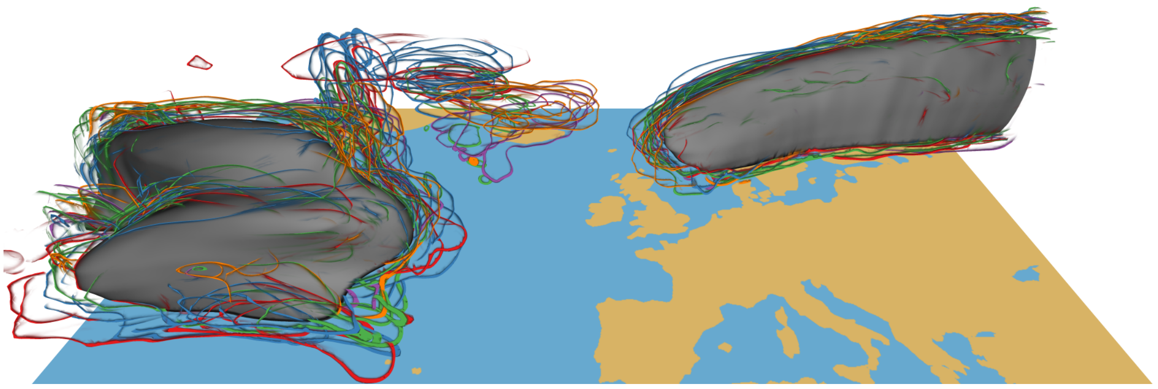


Figure 5.1: Visualization of a weather forecast ensemble from the ECMWF Prediction System [LP08]. This ensemble comprises wind velocity data and consists of 50 members. For one isovalue, all members are visualized as silhouettes of isosurfaces. Additionally, a mean ensemble member is rendered as a gray isosurface to enhance the visual perception of the spatial context. Color is used to cluster members by their similarity.

members are superimposed, which makes an expert's analysis quickly unfeasible. As an alternative, different members can be visualized by drawing them next to each other [GAW*11]. However, in this case, the spatial context between similar features in different members is lost. Furthermore, this method is only feasible for a relatively small number of members, due to the screen space limitation. Another approach is to provide the user with statistical data derived from the original ensemble instead of the individual members [PH11, PRW11]. For instance, mean and standard deviation can be computed at each spatial location and then presented to the user. In such a scenario, however, differences between members are smoothed out and important information, such as outliers, is lost.

To remedy the problem of occlusion, techniques from information visualization such as parallel coordinates or histograms can be combined with volume rendering by utilizing brushing and linking [GRW*00, DH02, WBWK00, KH13]. The idea behind these approaches is to show multiple linked views displaying different aspects of the original data. In this scenario, the user selects regions of interest in one view and is then provided with visual feedback in all linked views. Another approach is to cut the original volume into slices and to render each slice separately. Then, ensemble members can be visualized via drawing spaghetti plots, by overlaying isocontours of each member [PWB*09a]. However, in this representation, the spatial relationships between different slices are lost.

In this chapter, we propose a novel rendering approach to interactively visualize 3D isosurface ensembles. Our method preserves spatial coherence, while occlusion effects do not disturb the visual perception to a considerable degree. By providing interactive mechanisms, we enable the user to further explore the data on a more detailed level. A data set visualized by our technique is presented in Figure 5.1. In particular, the main contributions of our approach are:

- A novel efficient visualization technique for 3D isosurface ensembles based on rendering semi-transparent silhouettes. Our method is spatially coherent both within each member and between different members.
- An efficient implementation to convert 3D scalar field ensembles into polygonal meshes for different isolevels. In this process, all information necessary for rendering is precomputed and stored at the vertex level. Thus, the workload during rendering is minimized, meaning that even a large number of ensemble members can be displayed at interactive frame rates.
- By integrating movable cutting planes directly into the 3D view, we enable the user to gain more detailed insights at specific spatial locations without losing spatial coherence.
- Different hierarchical clustering algorithms based on isolevels enable the user to group and analyze members according to their similarity. Since we compute all clusters in a preprocess, the user can select and explore them interactively.
- Picking and brushing mechanisms as well as animations that enable the user to view selected members of interest as complete isosurfaces, thus enhancing the spatial recognition.

5.2 Related Work

Ensemble visualization belongs to the area of uncertainty visualization, which has been an important research topic in visualization for almost two decades [PWL97, JS03]. Many useful approaches have been published and a number of surveys exist [GS06, HBG*12, KH13, BHJ*14]. For example, diagram-based techniques have been proposed to augment the spatial context by visual cues, such as glyphs, charts or box plots [PKRJ10, SSSSW13, DDW14, JDKW15]. Clustering algorithms are used to break down larger data sets into groups of similar characteristics [Jai10, BKS04, BM10b, BHGK14, FBW16]. In our work, we use clustering to classify silhouettes with respect to their similarity. Recently, research has also focused on analyzing spatio-temporal ensemble data [PWB*09b, KSDD14, HHB16, LMK*15].

Spaghetti plots are a powerful technique to visualize 2D scalar field ensembles by simultaneously rendering an isocontour per member [PWB*09a, SZD*10, HMC*13, HMZ*14, PW13]. We extend this method to 3D data sets and provide interactive techniques to aid the user in the visual analysis, such as animation and picking. However, these approaches are typically less suitable to understand the 3D shape of isosurface ensembles. Several methods have been proposed to visualize isosurfaces extracted from 3D ensemble data, e.g., by means of animation [Bro04], volume rendering [DKLP01, TLB*11], or confidence envelopes [ZWK10, PH11, PRW11]. Wei et al. automatically select isosurfaces based on their representativeness [WLS13]. Other methods aim at rendering multiple isosurfaces into a single 3D view [BBF*11, AWH*12]. However, only a few ensemble members can be visualized simultaneously due to cluttering and occlusion. Matković et al. [MGKH09] visualize ensemble data as families of data surfaces in combination with cutting planes showing intersections with the surfaces. We pursue a similar approach by using cutting planes to guide the user towards specific regions of interest.

A number of methods exist for reducing 3D shapes to feature lines such as silhouettes, suggestive contours, ridge and valley lines, or apparent ridges [CSD*09, WMK13, WZ13, MWK14]. We compute silhouettes per member based on curvature due to its simplicity and computational efficiency [KWTM03], although other methods could be used as well.

5.3 Preprocess

To begin with, let us consider a scalar field ensemble given at the vertices of a 3D Cartesian grid, $f : \{1, ..., N\} \times \mathbb{R}^3 \to \mathbb{R}$. That is, at each grid point, a scalar value in $\mathbb{R}$ is associated with each member. Moreover, an isovalues is specified. Now, we generate a polygonal mesh for each member. As a result, we end up with N meshes, where $N \in \mathbb{N}$ denotes the number of ensemble members. Note that we repeat this process for each isovalue in a given set $I \subset \mathbb{R}$. Then, we generate a clusterization in due consideration of the current isovalue.

5.3.1 Mesh-Generation

Given a scalar field, i.e., one ensemble member, on a Cartesian grid and an isovalue, we first make use of the marching cubes algorithm to extract the corresponding isosurface in the form of a polygonal mesh. The marching cubes algorithm, originally proposed by Lorensen et al. [LC87], however, produces a uniformly resolved mesh. Consequently, polygonal primitives are generated at a very fine resolution, thus wasting memory. To overcome this drawback, we apply a mesh decimation algorithm that takes the underlying geometry into account. Our implementation is based on the quadric mesh simplification method as proposed by Garland et al. [GH97]. As a result, we end up with an adaptively resolved triangle mesh. Next, for every vertex x, we compute the following attributes and use them in the rendering process:

- We compute the *vertex normal* by taking the normalized gradient, i.e., $n = -g / \|g\|$, where $g = \nabla f_i(x)$ denotes the gradient and i refers to the member index. Since the ensemble is given on a Cartesian grid, we make use of finite differences to compute the respective derivatives numerically.

- The *principal curvatures* κ_1, κ_2 as well as the first *principal curvature direction* (PCD) p are calculated according to the method described by Kindlmann et al. [KWTM03]. Let $P = I - n \cdot n^\mathsf{T}$, $H = \left(\frac{\partial^2 f}{\partial x_i \partial x_j}(x) \right)$, the Hessian matrix, and $G = -PHP / \|g\|$. Next, we compute the trace T and the Frobenius norm F of G. Now, we obtain the principal curvatures
$$\kappa_{1,2} = \frac{T \pm \sqrt{2F^2 - T^2}}{2}.$$
Finally, p is obtained as the eigenvector of G corresponding to the eigenvalue κ_1.

- We compute the *outlyingness* as the mean squared distance to all members, i.e.,
$$o = \sum_{j=0}^{N} \left(f_j(x) - f_i(x) \right)^2 / N.$$

To save memory, these values are stored in a packed format. Position, principal curvatures and the outlyingness are stored in 16-bit floats per component. Since the normal and PCD are of unit length, we can store these vectors as a 32-bit unsigned integer. Here, the x and $y-$component are stored in the first 16 bits and in the next 15 bits, respectively. The sign of the $z-$component is stored in the last bit. These values are later reconstructed in a shader program. In total, 20 bytes are consumed per vertex. As a result, we end up with N meshes for each isovalue, where $N \in \mathbb{N}$ denotes the number of ensemble members, meaning that $N \cdot |I|$ meshes are generated in total. See the results section for an analysis of the memory requirement.

5.3.2 Clustering

For clustering, we compute a similarity matrix of size $N \times N$, where at each entry (i, j) the difference between member i and j is stored. We compute the difference as
$$d(i, j) = \sum_x \left(\sqrt{|f_i(x) - \mu|} - \sqrt{|f_j(x) - \mu|} \right)^2 / m,$$
where μ and m denote the current isovalue and the number of grid points, respectively. By taking the square root, changes in values closer to the isovalue are considered of greater importance.

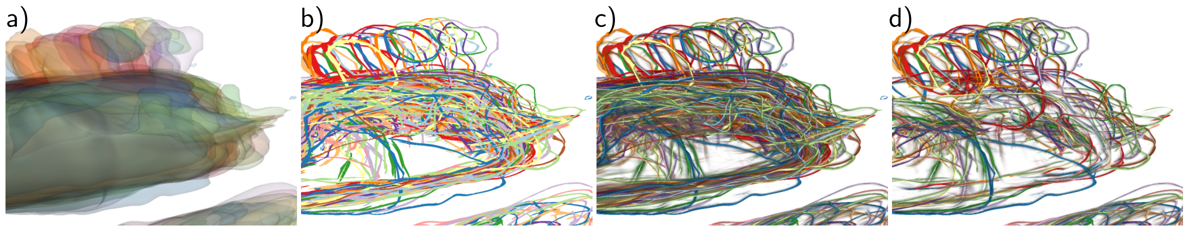


Figure 5.2: Visualization using different rendering techniques. a) Semi-transparent surfaces. b) 3D spaghetti plots of silhouettes. c) Shaded silhouettes. d) Shaded silhouettes with density-based removal.

Clusters are then generated based on the similarity matrix by using an agglomerative hierarchical clustering method [Jai10]. We begin with N clusters, where each member is assigned to one unique cluster. Then, we proceed by merging a pair of clusters of minimal distance. This process is repeated until one cluster remains. Now, for each step we store the respective cluster distribution. Here, distance is computed in the following way. Suppose, each cluster contains exactly one member. Then, we obtain the distance as the corresponding entry in the similarity matrix. Otherwise, we compute the distance by using a linkage criterion. For instance, the average linkage criterion, defined as

$$d\left(A,B\right)=\sum_{a\in A,b\in B}d\left(a,b\right)/\left(\left|A\right|\left|B\right|\right),$$

yields suitable results in our implementation [Bij73]. Again, this process is carried out for each isovalue.

5.4 Rendering Shaded Silhouettes

In this section, we explain the proposed rendering technique for isosurface ensembles based on the precomputed per-vertex attributes and clusters. For a user-selected isovalue and each selected member, the corresponding mesh is rendered. Note that rendering solid isosurfaces suffers from occlusion effects, in particular if multiple members are rendered simultaneously. Although drawing isosurfaces with transparency reduces occlusion effects, perceptually multiple layers of transparency cannot easily be distinguished, and quickly becomes infeasible when too many surfaces are overlaid. This can be observed in Figure 5.2a. Therefore, we propose a different approach based on rendering only silhouettes instead of whole surfaces. Silhouette rendering is not a novel technique, yet to the best of our knowledge, it has not been used for ensemble visualization so far. Figure 5.1 shows a weather forecast ensemble containing 50 members that is rendered using our method. In this view, the gray isosurface represents the mean of all members, which was generated by computing the average value at each grid point. Clearly, the members can be distinguished and the spatial context is preserved. By rotating the camera, a further improved recognition of the spatial structures is achieved.

5.4.1 Drawing Silhouettes in the Fragment Stage

Once the attributes are reconstructed in the vertex shader, rendering silhouettes is done in the fragment stage. By using the normal n, the curvature along view direction κ and the view vector v, we compute the silhouette coefficient as

$$\sigma = \left| v \cdot n^{\mathsf{T}} \right| / \sqrt{\tau\kappa \left(2 - \tau\kappa\right)}.$$

Here, τ denotes the silhouette thickness, such that a silhouette is present iff $\sigma \leq 1$. Let us refer to [KWTM03] for further details on the derivation. To efficiently compute κ in the fragment shader, we make use of the precomputed principal curvatures $\kappa_{1,2}$ and the PCD p as introduced in the previous section. First, we obtain the angle φ between the PCD and the view direction, projected onto the plane aligned with the normal at the fragment position, as

$$\cos\varphi = \langle v - \langle n, v \rangle n, p \rangle .$$

Now, we calculate the curvature in view direction as

$$\kappa = \kappa_1 \cdot \cos^2\varphi + \kappa_2 \cdot \sin^2\varphi.$$

This equation holds, because PCDs are always aligned orthogonal to each other. Note that it is not necessary to actually compute φ, since $\cos^2\varphi + \sin^2\varphi = 1$. By drawing fragments with $\sigma \leq 1$ as opaque using a solid color, and discarding all other fragments, we obtain 3D spaghetti plots as can be seen in Figure 5.2b. However, this approach suffers from the drawback that distinguishing different members of the same color is infeasible without rotating the camera. To overcome this limitation, we make use of shading and transparency effects. Given an RGB-color value c and silhouette coefficient σ, we return the fragment's RGBA color $c^\star$ as

$$c^\star = \begin{cases} (c, 1) & \text{if } \sigma < \frac{1}{2} \\ \left(c \cdot \frac{1}{2}, 1 - \left(\sigma - \frac{1}{2}\right) / \frac{1}{2}\right) & \text{if } \frac{1}{2} \leq \sigma \leq 1 \\ \text{discard} & \text{else.} \end{cases}$$

The result is depicted in Figure 5.2c.

5.4.2 Density-Based Removal

To reduce the amount of visual clutter, consider the observation that there are some regions where the density of silhouettes is greater. Hence, we can eliminate a number of silhouettes in such regions without losing important information, since all these silhouettes share a similar shape. In doing so, unnecessary data are removed from the visualization, thus making it easier to grasp for the user.

To identify silhouettes that are suitable for removal, we resort to the outlyingness o as explained in the previous section. This value indicates how much difference there is between the value of the current member and all other members at a given position with respect to the selected isovalue. Thus, lower values imply a greater density of isosurfaces and therefore a greater density of silhouettes. This gives rise to the following algorithm, implemented in the fragment shader. Let i_M, s_G denote the member index and group size. Here, the group size determines the number of nuances for silhouette removal. Then, we obtain the group index by

$$i_G = i_M \pmod{s_G}.$$

Let $o^*_{Min}, o^*_{Max} \in \mathbb{R}$ denote the global minimum and maximum outlyingness threshold. Now, we compute the thresholds for group index i_G as

$$\begin{aligned} o_{Min}(i_G) &= o^*_{Min} + i_G \cdot (o^*_{Max} - o^*_{Min})/s_G \\ o_{Max}(i_G) &= o^*_{Min} + (i_G + 1) \cdot (o^*_{Max} - o^*_{Min})/s_G. \end{aligned}$$

Finally, we compute the outlyingness-based opacity as

$$\alpha(i_G, o) = \operatorname{clamp}\left(\frac{o - o_{Min}(i_G)}{o_{Max}(i_G) - o_{Min}(i_G)}, 0, 1\right).$$

If the result equals 0, the fragment is discarded. Otherwise, the alpha-component of the fragment shader output is multiplied by $\alpha(i_G, o)$.

To identify regions where most members agree, the user can start out with a lower degree of silhouette removal and then gradually increase this threshold to focus on finer structures. This can be done by dynamically adjusting o^*_{Min} and o^*_{Max}. The result is shown in Figure 5.2d. As an alternative to altering the opacity, we could also resort to color-coding the silhouette density or adjusting the thickness, although this is not shown here. Note, that only parts of silhouettes are removed, meaning that silhouettes in sparser areas remain unaffected.

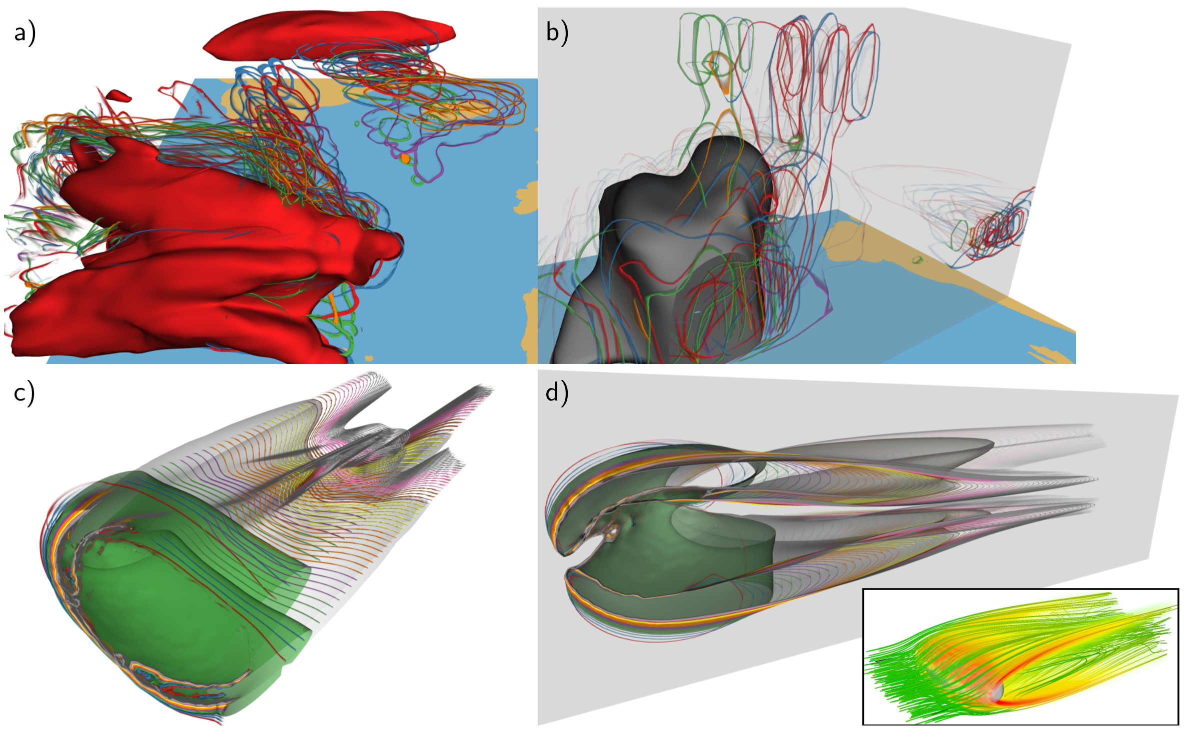


Figure 5.3: Results from two ensemble data sets. a) An outlier was selected in the ECMWF data set (bottom left) to get a better understanding of its shape. b) A cutting plane allows us to analyze a feature in more detail. c) Overview of a Navier-Stokes flow simulation comprising 56 runs. d) A cutting plane reveals the inner structures. The underlying flow is depicted at the bottom right.

5.5 Visualizing Details on Demand

In line with the information seeking mantra "Overview first, zoom and filter, then details-on-demand" proposed by Shneiderman [Shn96], we have implemented several mechanisms to visualize user-selected details, namely, cutting planes, clustering, picking and brushing, and animation. We will now discuss each of these mechanisms in more detail.

5.5.1 Cutting Planes

We have integrated cutting planes into our implementation that can be interactively moved by the user. In this way, we enable the user to gain more detailed insights at specific locations of interest. This process is illustrated in Figure 5.3b,d. Here, the cutting plane is placed such that a selected feature is revealed by displaying the intersection curve between the isosurface and the plane. Thus, the user is able to gain more specific insights with respect to the structures between the ensemble members, as this information cannot be gathered by analyzing the silhouettes alone.

Rendering cutting planes is accomplished in the fragment stage. To begin with, we locate the relative position of the current fragment to the cutting plane with respect to the camera position. For this, we compute the distance d from the fragment's position to the cutting plane along the view direction v. To prevent occlusion, fragments lying in front of the plane, i.e., $d < 0$, are discarded. In order to render intersection curves of approximately equal thickness ϵ, we consider a fragment as lying on the plane iff the following conditions are met: a) The distance along the view direction, projected onto the surface normal, is within the plane thickness, i.e.,

$$\langle v, n \rangle \, d \leq \epsilon;$$

b) the distance along the view direction is within the plane thickness, along the view direction, projected onto the plane normal n_P, i.e.,

$$d \leq \langle v, n_P \rangle \, \epsilon.$$

In this case, the fragment is rendered by using the term $\langle v, n \rangle \, d$ to compute shading and opacity in the same way as explained for the silhouette coefficient. Otherwise, we continue with rendering silhouettes. To enhance the visual perception, we also render the cutting plane itself as a semi-transparent rectangle.

5.5.2 Clustering

We utilize hierarchical clustering algorithms to enable the user to group members according to their similarity with respect to the current isovalue. After the clusters have been computed in the preprocess, they can be selected interactively in the running visualization. This is depicted in Figure 5.1, where members are clustered into five groups. Color is used to indicate each cluster. By showing or hiding clusters separately, the user can focus on cluster-specific features without being distracted by the presence of other members. In this way, outliers can quickly be identified and trends can be analyzed in more detail.

5.5.3 Picking and Brushing

We have developed picking and brushing functionality to provide means to view selected members of interest as solid or semi-transparent isosurfaces. This improves the recognition of selected members and at the same time can serve as a point of reference to facilitate the spatial perception of the silhouettes. This can be seen in Figure 5.1, where the mean surface is visualized in gray. Note, however, that this method is only effective for displaying one or a few members as otherwise occlusion effects prevail. In our implementation, we obtain the currently picked member by checking the mouse coordinates against the screen space coordinates in the fragment stage.

5.5.4 Animation

To quickly discover differences and similarities between a selected subset of members of interest, we provide the user with the ability to animate over members. In this process, the user first selects a subset of members by picking and brushing. Then, during the animation, one member after another is visualized by a solid isosurface. To make the transitions easier to understand, members are sorted in the following way. First, members are sorted by their cluster index. Second, within each cluster, the order is such that more similar members are closer in the sequence. For this, we make use of the precomputed similarity matrix and then sort members in a greedy fashion.

5.6 Results

We now discuss some results to demonstrate the potential of our visualization technique. To begin with, let us consider an ensemble comprising wind velocities of a weather forecast simulation by the ECMWF Ensemble Prediction System [LP08]. This ensemble consists of 50 members of resolution $128 \times 256 \times 16$. In Figure 5.1, for an isovalue of $30\,\frac{m}{s}$ the resulting isosurfaces are visualized as silhouettes. An additional mean member is included, as explained before. To store the data for visualizing all 51 members (including the mean), roughly 16 MiB are consumed per isovalue. On a standard desktop PC, the process of precomputation takes $\approx$ 3 minutes for generating all meshes and another $\approx$ 2 minutes for computing the similarity matrix. The time for computing the clusterization based on the similarity matrix is negligible. Note that the computational cost is proportional to the grid resolution and the number of members. Rendering the image in Figure 5.1 took 3.6 ms measured on an NVIDIA GeForce GTX 580 graphics card at a viewport resolution of 1900×1200. This means that our implementation enables the user to interactively analyze an ensemble of isosurfaces.

By observing the visualization in Figure 5.1, we can spot two regions, where most, if not all, simulation runs agree on a wind velocity of $30\,\frac{m}{s}$. One such region is located above the Baltic Sea and the other region above the North Atlantic Ocean south of Greenland. Note that we have also determined that the wind velocities within these regions are greater than $30\,\frac{m}{s}$ by studying other isovalues, although this is not shown here. Moreover, there is another region located in the middle where only some members exhibit the aforementioned wind velocity. By clustering the members into five groups and using color to indicate the cluster membership, we can discover the main trends arising at that region. Here, we can also identify an outlier and investigate the shape of that member's isosurface by picking the respective silhouette. This is shown in Figure 5.3a. Due to the preprocess, the user can change the number of clusters interactively.

Let us now focus again on the second region. Here, we find a structure of curls on the top, where only some members agree as this area is clearly separated from the mean isosurface. If we place a cutting plane as shown in Figure 5.3b, we can analyze this feature in more detail and discover the inner structures. By moving the plane, the spatial perception is enhanced, and ellipsoid zones of higher wind velocity are revealed, indicating isolated air turbulences predicted by some simulation runs.

As a second example, let us consider an ensemble featuring an incompressible fluid flow evolving around an ellipsoid obstacle. 56 Navier-Stokes simulation runs were performed with different viscosities on a Cartesian grid of resolution $145 \times 49 \times 49$. After a given simulation time, the vorticity magnitude was saved as the scalar field ensemble. The result is presented in Figure 5.3c using our visualization technique. Clearly, it is easily possible to visually distinguish members as only minimal occlusion effects occur. By inserting a cutting plane, we can also gain insight into the inner regions and study the behavior of the flow simulation in more detail as shown in Figure 5.3d. In this data set, the use of animation proved to be helpful, since the shape changes gradually along the sequence of all members.

5.7 Conclusion and Future Work

In this chapter, we have presented a novel visualization technique for 3D scalar field ensembles. Our approach is based on the idea of rendering silhouettes instead of solid isosurfaces. Hence, occlusion artifacts are minimized without losing spatial coherence. We have also implemented several ways to provide a more detailed analysis of the ensemble. That is, by enabling the user to place cutting planes in the 3D view and by providing means of clustering and animation, the ensemble can be investigated according to different criteria. To study individual members in the context of the whole ensemble, the user can resort to picking and brushing functionality. Moreover, by precomputing all computationally involved parts in a preprocess, we have accomplished rendering at interactive rates. We have demonstrated the effectiveness of our approach by visualizing and analyzing two ensembles of very different characteristics. In both cases, our method was able to reveal relevant features and supported the user to gain insight into the spatial structures.

The work presented in this chapter opens up different aspects that can be investigated in future research: By extending our method to time-varying ensembles, the concept of animation might prove as an effective tool. Integrating mechanisms to automatically detect relevant features would allow placing cutting planes or other visual clues accordingly. By utilizing the GPU, the preprocess could be significantly sped up, although we do not consider this as a critical issue, as it has no effect on the rendering performance. Finally, a user study in collaboration with domain experts would be an effective way of evaluating the practical benefits of our method.

6 Ray-Casting Based on Vector-to-Closest-Point Octrees

In this chapter, we propose a novel improvement of voxel-based ray-casting that, unlike classical voxel-based approaches, does not produce block artifacts. We use a hierarchical Vector-to-Closest-Point representation, which yields a smooth approximation of the surface. This method also serves as a foundation of the shape-based visualization technique presented in the next chapter. This chapter is largely based on our publication:

Demir I., Westermann R.: Vector-to-Closest-Point Octree for Surface Ray-Casting. In *Vision, Modeling & Visualization* (2015), Bommes D., Ritschel T., Schultz T., (Eds.), The Eurographics Association. `doi:10.2312/vmv.20151259`. [DW15]. © 2015 EUROGRAPHICS.

6.1 Introduction and Related Work

With the rising capabilities of modern GPUs, ray tracing algorithms have become more popular for real-time rendering applications. Since ray tracing is a technique based on finding intersections between rays and the objects of the scene, it is crucial to employ efficient methods to achieve interactive frame rates. Voxel-based surface ray-casting on the GPU [GMIG08, CNLE09, LK10a] has been introduced as an interesting alternative to rasterization-based polygon rendering and triangle ray tracing. It inherits the benefits of ray-based GPU rendering techniques [PBMH02, CHH02, EVG04, FS05, CHCH06, GPSS07, PGSS07, HSHH07, AL09] to effectively exploit the GPU's massively parallel design and use many processing units simultaneously. In addition, it can effectively perform fine-granularity occlusion culling on the ray level. Moreover, the regular voxel grid gives rise to efficient ray traversal schemes, and it allows generating levels of detail with prescribed (screen-space) error in a very simple and efficient way.

A limitation of voxel-based techniques is that they produce visual artifacts, when the size of the voxels exceeds the pixel size in screen space. In this case, the underlying voxel grid becomes visible in the form of block structures. As a consequence, classical voxel-based techniques need to build very deep trees, so that even for extreme zoom-ins, the voxel size can still match the pixel size. It is worth noting that the high-resolution representation is also required in areas where the initial surface is smooth.

Different solutions have been proposed to remedy this problem. "Far Voxels" have been introduced as an efficient LOD structure for polygonal models by Gobbetti and Marton [GM05]. They render the polygon structure, where the details are high, and render a voxel-based approximation of polygon clusters at coarser resolution levels. Laine et al. [LK10b] introduce a sparse voxel octree (SVO), where the voxels are supplemented by piecewise linear contour representations, which are then used during rendering to avoid block artifacts and generate smooth object silhouettes. Their method was later enhanced by Kämpe et al. [KSA13] using directed acyclic graphs (DAG) instead of octrees to reduce the memory consumption when encoding identical regions. Similar to "Far Voxels", Reichl et al. [RCBW12] employ a hybrid rendering method, which combines rasterization and ray-casting. Rasterization is used in such cases where ray-casting voxel blocks would lead to visual artifacts, i.e., when in an adaptively constructed octree the finest voxel resolution has been reached. Heitz et al. [HN12] have presented a method for rendering signed distance field octrees with Gaussian descriptors of the surface within the voxels. "Sphere Tracing" is a rendering technique proposed by Hart for implicit surfaces defined by signed distance functions [Har96], which was later improved upon by Keinert et al. [KSK*14]. Bastos et al. presented a method built on Sphere Tracing to efficiently render Adaptively Sampled Distance Fields (ADFs) on the GPU [BC08]. While their method is restricted to signed distance isosurfaces, our method is able to handle arbitrary surfaces. "Enhanced Sphere Tracing" is a method proposed by Keinert et al. [KSK*14], which builds upon sphere tracing, a rendering method for signed distance bound geometry.

Already 17 years ago, a different methodology has been proposed by Gibson [Gib98], and used for voxel-based surface rendering on fixed-function graphics hardware by Westermann et al. [WSE99]. Gibson proposed to make use of a distance field to achieve a higher order interpolant of a level-set surface in an intensity volume. Westermann et al. used this idea to voxelize a polygon surface into a 3D scalar distance field and render the implicit surface via slice-based volume rendering. For a thorough overview of algorithms for computing signed distance fields starting from polygon models as well as a number of applications of such procedures, let us refer to Jones et al. and Frisken et al. [JBS06, FPRJ00]. In the context of fluid simulation, Auer et al. suggested a rendering technique for an equidistant, regular low-resolution closest-point simulation grid [AMT*12].

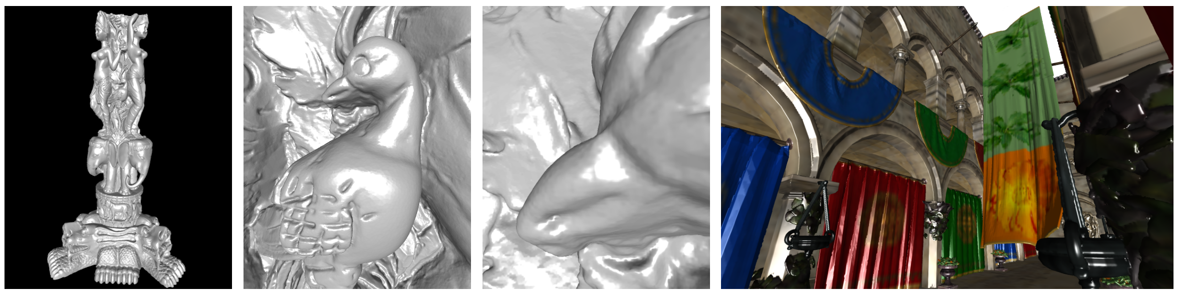


Figure 6.1: Rendering from a hierarchical Vector-to-Closest-Point (VCP) grid at different zoom steps. Left: The Thai statue (10 million polygons) renders in less than 10 ms per frame at a maximum grid resolution of $2048 \times 1024 \times 1024$. Right: For the Sponza model the hierarchy is 3 levels shallower than for a sparse voxel hierarchy in order to achieve roughly the same rendering quality (for the Thai statue the gain is between 1 (left) and 5 levels (right)).

Contribution. In this chapter, we present a novel approach based on previous works to overcome the limitations of voxel-based surface rendering. Inspired by the distance-to-closest-surface representation by Gibson [Gib98], we present the hierarchical *Vector-to-Closest-Point* representation (VCP) to enhance the surface approximation quality of a grid-based representation. As demonstrated in Figure 6.1, as compared to a standard voxel hierarchy, our VCP representation results in a significantly shallower tree, and it reduces the memory consumption and tree traversal costs during rendering. The surface can be determined as the zero-level-set of the VCP function, and smooth normals can be computed by interpolating VCP vectors. Our rendering specific contributions are:

- An algorithm to construct a grid structure from a given arbitrary polygon mesh. Broadly speaking, we start with a coarse grid and store at every grid point the VCP with respect to the mesh. The grid is then refined in an octree-fashion, where it is required for precise rendering, i.e., in close vicinity to the mesh. As a result, we obtain a hierarchical VCP representation that is then used by the GPU.
- A method for efficiently ray tracing the given geometry by utilizing the described data representation. This specifically includes a method of traversing through the VCP octree data on the GPU as well as a technique for interpolating between VCPs, which is largely consistent to the original mesh-based geometry.
- A significantly shorter model hierarchy compared to a classical voxel representation.
- The option to render a smooth surface at arbitrary zoom-ins from a VCP grid.
- The ability to include per-grid point attributes such as color that could also be utilized in the context of texture coordinates, normal maps, and similar attributes.

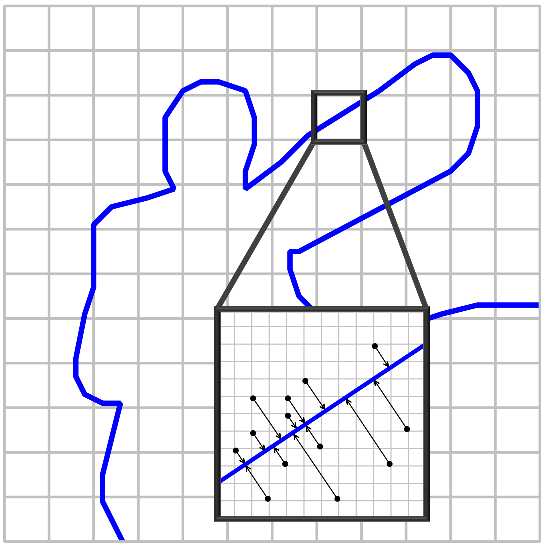


Figure 6.2: Illustration of the VCP representation in 2D. At each grid point, a vector to the point closest to the original geometry (blue) is stored; this is visualized by a few such vectors (black). The grid is adaptively refined in the vicinity of the object.

- A dynamic GPU memory management system that loads more refined subsets of the given geometry into GPU memory in places, which are located close to the camera and are visible in the current viewport. To maintain interactive frame rates, this process is done in the background by utilizing multiple concurrent threads.

The remainder of this chapter is organized as follows. First, we discuss the construction of the proposed VCP grid from high-resolution triangle meshes. In the next section, we discuss our method to ray-cast the surface encoded in a hierarchical VCP grid on the GPU. Then, we shed light on the embedding of the proposed rendering technique into a scalable out-of-core system. We analyze the performance and memory consumption of our technique, compare our results to those generated by alternative rendering techniques, and, finally, conclude the chapter.

6.2 VCP Representation

To begin with, let us consider a geometric object, represented by a polygonal surface, which is embedded in three-dimensional Cartesian space. The space is discretized via a uniform Cartesian grid. This is illustrated in Figure 6.2 in two dimensions. Now, we compute for each grid point the VCP vector with respect to the given geometry, i.e., the vector pointing to the closest surface point. An algorithm to efficiently compute a VCP grid for very large

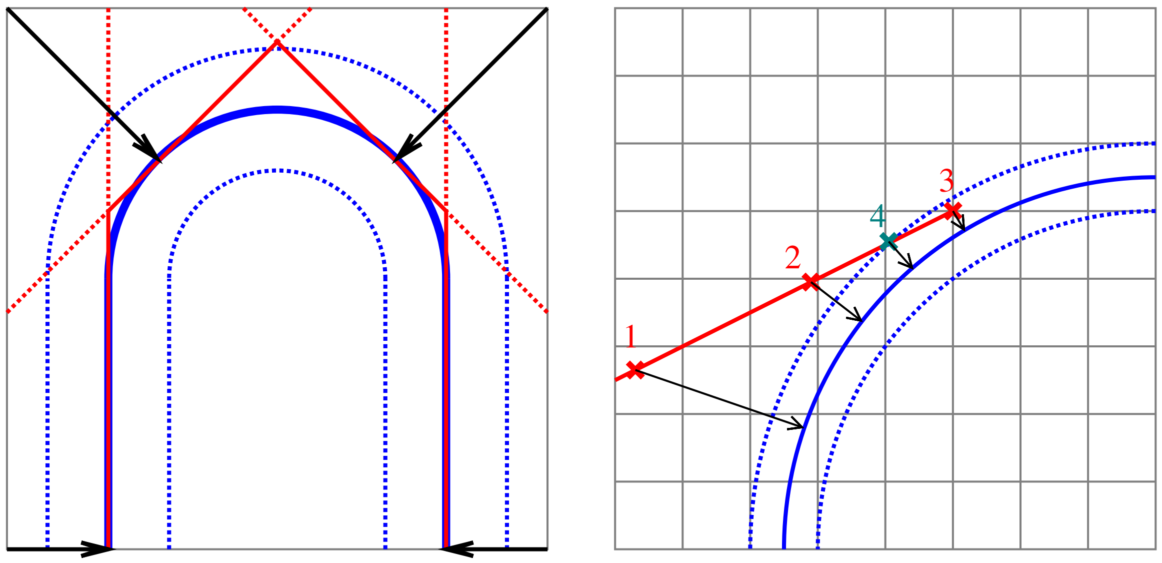


Figure 6.3: Left: If only distance information is stored at the grid points, the feature (blue) is blurred or lost during linear interpolation, since it is located within one grid cell. Our interpolation scheme based on the VCP representation (black arrows), however, conserves the feature to a better degree as illustrated by the red lines. Hence, we can choose a smaller ϵ-band leading to more precise rendering results. Right: Ray-casting through the VCP grid. At each step (marks 1, 2, 3) along the ray (red), the VCP distance to the object (blue) is obtained. Once it is less than ϵ, i.e., the distance from the dotted blue curve to the feature, this is considered as a hit (mark 3). Interpolating between the current and the previous step results in a point on the ray with a VCP distance of about ϵ (mark 4).

polygon meshes is presented below. Note that storing the full grid at a high resolution would waste a significant amount of memory, as during rendering, the VCP data is only required in a narrow ϵ-band around the object's surface. Therefore, our approach uses an additional regular grid, which comprises *blocks* of 2^3 cubical cells. A VCP representation is computed only for those (non-empty) blocks, which overlap the narrow band, while for all other (empty) blocks, we store one single scalar value indicating the minimum distance of all vertices of this block to the surface. These values are used during ray-casting to adaptively vary the step-size, i.e., a step as large as the stored minimum distance can never cross the surface. We will later describe how to employ a hierarchical representation to adaptively prune empty space comprising multiple empty blocks.

In principle, to implicitly encode the surface, one could also use a scalar distance field, which stores at every grid point the shortest distance to the surface. Such a representation, however, requires the use of signed distance values in order to accurately determine the distance-0 isosurface. Computing signed distances, on the other hand, is non-trivial and generally speaking not even possible, e.g., if the surface is non-orientable or not closed. By contrast, our technique can be applied to arbitrary geometry. In addition, the distance values in a discrete grid approximate the surface with an error that is linear in grid spacing, while each closest point defines the exact position of a surface point. Hence, a better approximation quality is achieved with a VCP representation, as illustrated in Figure 6.3 (left).

Moreover, to enable lighting and shading computations, when working with a distance field, normal vectors have to be stored, or obtained by sampling the gradient of the scalar field, which is computationally expensive. As we will demonstrate, storing normals is not necessary when working with a VCP representation, which saves a considerable amount of memory.

6.2.1 Ray-Casting

The VCP grid is rendered via parallel GPU ray-casting. For each ray, we simultaneously compute the first and last intersection point with the object's bounding box, and we let the rays march through the VCP grid using varying step-sizes. This process is illustrated in Figure 6.3 (right). At each sampling point, the allowed size of the next step is computed, either by reading the scalar distance, d, value from the hit (empty) block, or by using the length of the interpolated VCP vector in case a non-empty block is hit. If this distance is less than ϵ, a ray-surface intersection point is assumed, otherwise the value is used as the step size for the next sampling point. In particular, since we know that there is no object within the distance d, we can safely set the step size to $\max\{d - \epsilon, \epsilon/2\}$. In fact, the step size can be chosen even greater in practice, as will be explained later on.

To determine the exact intersection point, the location between the current and the previous sampling point at which the distance is exactly ϵ is interpolated. By doing this for every ray, a smooth surface being a constant distance away from the exact surface is rendered. It is clear that by decreasing ϵ, we can come arbitrarily close to this surface. Once a ray-surface intersection is found, a local lighting model using the surface normal vector at the intersection point is evaluated. The normal vector is given by the normalized VCP vector, since it is always perpendicular to the surface.

At this point, the procedure could be extended to compute shadows by sending another ray from the hit position to the light source. Likewise, additional non-camera rays could be sent from the hit point to render reflection and refraction effects. This can be implemented in a straightforward way, since our grid structure is independent of the camera position. Moreover, we can also look up other attributes such as color at this point and use them to enhance the visual appearance.

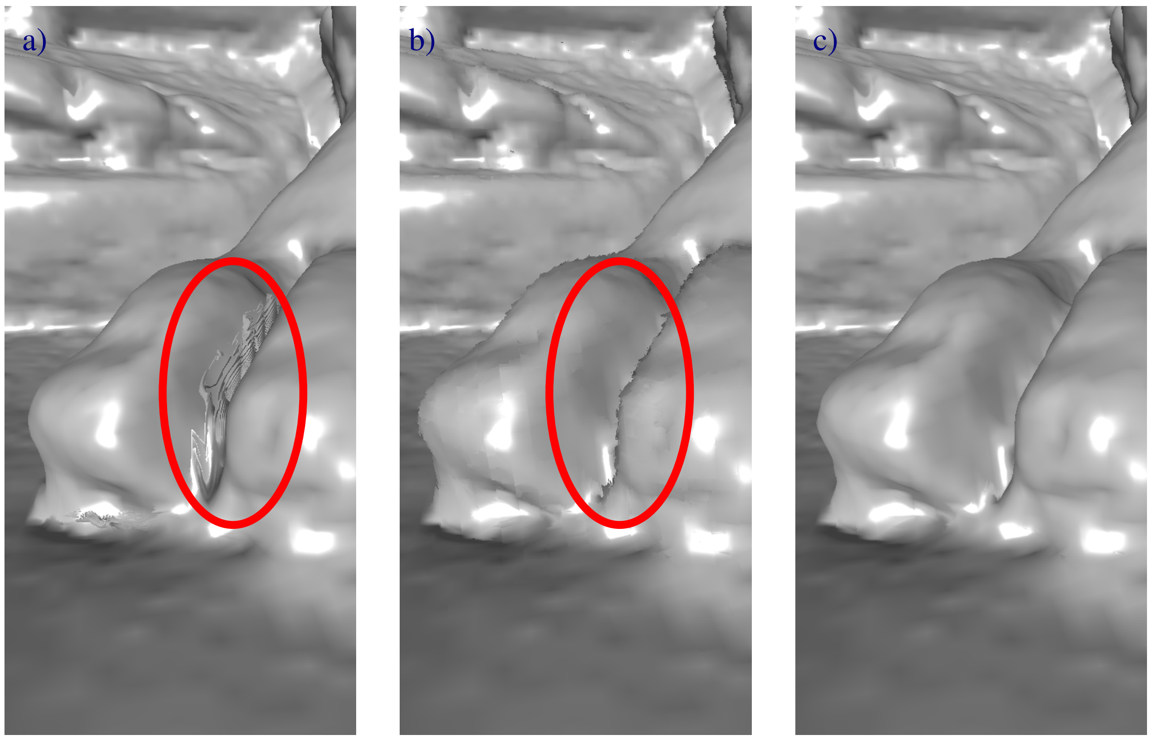


Figure 6.4: Comparison of three interpolation schemes. a) Trilinear interpolation produces artifacts since adjacent VCP vectors pointing in opposite directions are canceled out. b) Our intermediary scheme also leads to artifacts since triangles are substituted by planes, which have no boundary. c) Combining both methods avoids artifacts of either case and is computationally cheap.

6.2.2 VCP Interpolation

To obtain the VCP at an arbitrary location, we interpolate between the 8 given vectors at the vertices of the respective cell. For this, it is crucial to have an interpolation scheme, which is consistent with the VCP geometry. A straightforward way would be to simply use trilinear interpolation. However, as shown in Figure 6.4a, this method produces artifacts: Suppose, we have VCP vectors $(-\epsilon, 0, 0)$ and $(\epsilon, 0, 0)$ at two adjacent grid points, meaning that they are pointing to regions that are close in space but distant with respect to the surface. Now, linearly interpolating in the middle of these two points would yield the vector $(0, 0, 0)$ and hence result in an intersection point during ray-casting. To remedy this problem, we propose a specially designed interpolation scheme, which fits the purpose of interpolating VCP data consistently. This method is illustrated in Figure 6.3 (left).

When interpolating at p', we iterate over all 8 grid points p_i and think of each VCP as describing a plane. More precisely, each vector vcp_i describes a (unique) plane P_i such that

vcp_i starting from p_i is a perpendicular on P_i, i.e., $vcp_i \perp P_i$ and $p_i + vcp_i \in P_i$. We now obtain the interpolated VCP with respect to p_i as the perpendicular from p' to P_i, i.e.,

$$vcp_i^\star = \left(\left(vcp_i \cdot (p_i + vcp_i - p')\right) / \left(vcp_i \cdot vcp_i\right)\right) \cdot vcp_i .$$

Finally, the shortest vector is returned as the result, i.e.,

$$vcp^\star = \arg\min_{vcp_i^\star} \left\{ \| vcp_i^\star \| \right\} .$$

However, this approach also leads to rendering artifacts as can be seen in Figure 6.4b. This comes from the fact that by substituting planes for a mesh, which originally consisted of triangles, the boundaries of the triangles are lost. As a costly solution to this problem, one could not only store VCP vectors but additionally the triangles to which they refer. A much more feasible solution is to combine our method with the trilinear interpolation scheme, such that the VCP with the greater length is returned as the final result. In this case, artifacts of either case are avoided leading to a smooth rendering result. This is depicted in Figure 6.4c.

6.3 VCP Octree

The concept underlying our construction of a hierarchical octree structure is illustrated in Figure 6.5. The octree is built bottom-up from the VCP grid outlined before. Each block is initially considered as a leaf. For each block, we compute the minimum distance d_{min}, as either the minimum of all scalar per-vertex distances of empty blocks or the minimum of all VCP distances of the vertices of non-empty blocks. Whenever the minimum distance is greater than a certain threshold, i.e., $d_{min} > t$, we mark the corresponding block as empty. In this way we ensure that VCP information is only stored in the close vicinity of the surface, thus lowering the memory requirement. To maximize the number of empty cells we minimize t such that ray-casting produces no visual artifacts. For this, we set t equal to 2 times the distance of two diagonally neighboring grid points with respect to the finest resolution. This ensures that an intersection with the object is never lost during the hierarchy construction. The octree is then constructed according to the following merging principles, where we consider groups of 2^3 blocks:

- If all nodes are marked as empty, they are combined to an empty leaf in the next octree level (Figure 6.5b, nodes with only green child nodes).
- If all nodes are leaves but some of them are non-empty, we check, whether it is possible to simplify that block to one single cell. For this, we test how well the VCP at the vertices of the non-empty blocks can be interpolated from the 2^3 vertices of the single

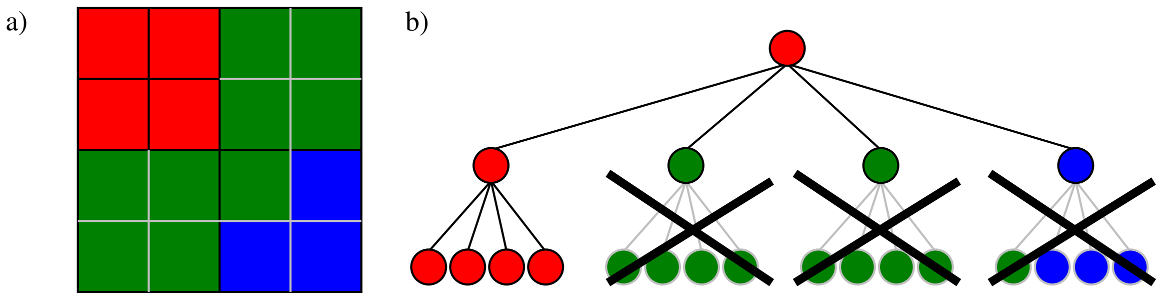


Figure 6.5: a) In this 2D-illustration, each cell is surrounded by 4 neighboring grid points, where a VCP is given at each grid point. Green cells correspond to empty leaves. Blue / red cells correspond to non-empty leaves that can / cannot be merged at the next level. b) The resulting octree after eliminating all nodes that can be combined.

cell. We do so by computing the Euclidean distances d_i between the interpolated and the exact VCP vectors: We iterate over all grid points p_i contained by the current block and compute the Euclidian distance d_i between the interpolated VCP according to the block's vertices $vcp^{\star}(p_i)$ and the original VCP given at the finest resolution $vcp(p_i)$. If the maximum distance falls below a certain threshold, i.e.,

$$\max \{\|vcp\,(p_i) - vcp^*\,(p_i)\|\} < t,$$

the nodes are merged to a non-empty leaf in the next level. If the nodes cannot be merged, an internal node is inserted at the next level pointing to corresponding leaves (Figure 6.5b, nodes with only green and blue child nodes)

- Otherwise an internal node is inserted at the next level pointing to these nodes (Figure 6.5b, nodes with red child nodes).

6.3.1 Mesh-based Generation

In this section, we describe, how a full VCP grid is constructed from a given triangle mesh. To begin with, let us consider the simplified case of computing the VCP from an arbitrary point p in space to a single triangle given by points p_1, p_2, p_3. This can be done efficiently by calculating the barycentric coordinates of the (perpendicular) projection p' from p onto the triangle as proposed by Heidrich [Hei05]. Let

$$u = p_2 - p_1, v = p_3 - p1, n = u \times v, c = n \cdot n, w = p - p_1.$$

Then, we obtain the barycentric coordinates as

$$\begin{aligned}\gamma &= ((u \times w) \cdot n) / c \\ \beta &= ((w \times v) \cdot n) / c \\ \alpha &= 1 - \beta - \gamma.\end{aligned}$$

If we have $0 \leq \alpha, \beta, \gamma \leq 1$, then p' lies inside the triangle and equals also the VCP, i.e., $vcp = p'$. This holds because the VCP vector to a plane is necessarily perpendicular. If p' is outside of the triangle, the VCP must coincide with an edge (or a vertex). For this, we compute the closest point vector to each edge and return the vector such that its length minimal. To compute the closest point vector from p to a line segment given by p_1, p_2, let $u = p_2 - p_1$ and

$$\lambda = \frac{(p - p_1) \cdot d}{d \cdot d}.$$

Let $\lambda^\star = \operatorname{clamp}(\lambda, 0, 1)$. We finally obtain the VCP as $vcp = p_1 + \lambda^\star \cdot d$. Note that it is also possible to store other attributes such as color or texture coordinates at this point.

We now extend our approach to arbitrary meshes consisting of multiple triangles. Note that in the case of polygon meshes our algorithm could be applied after decomposing each polygon into a set of triangles (e.g. [dBCvKO08]). Instead of iterating over all triangles and store the VCP with the least distance at each grid point, which is infeasible for any non-trivial mesh, we propose an algorithm that consists of two stages.

Starting with an empty grid, we insert each triangle into the grid in the following manner. First, we compute the bounding box for the current triangle and then update each grid point within this box. More precisely, each grid point is updated, if it is currently empty or it is pointing to another triangle with a greater VCP distance. Next, to expand the VCP data computed so far, we utilize a region-growing algorithm. We iterate k times over all grid points and compare the current grid point p with each surrounding grid point q in the following way. If p is empty, i.e., points to no triangle, let p point to the same triangle as q. Otherwise, if p, q point to triangles T_p, T_q, update p, such that it points to T_q if the VCP distance from p to T_q is less than the current distance, i.e., to T_p. The number k depends linearly on the aforementioned value ϵ.

For technical reasons, namely to enable multi-threading, this process cannot be done in-place, meaning that we need to maintain an input and an output grid and swap their contents after each region growing iteration.

6.4 GPU Implementation Issues

To reduce the number of memory indirections on the GPU, the octree is not constructed up to one single root node. Instead, the construction process is stopped at a certain level, where the memory requirement falls below a given threshold. The data generated so far constitutes the indirection pool of the octree. To traverse it efficiently on the GPU, the following information is encoded at each node as a 32-bit signed integer. For internal nodes, a pointer to the node index of its first child is stored; all child nodes are then stored at subsequent locations in memory (up to index `0x40000000 - 1`). An empty leaf is encoded by an integer greater than or equal to `0x40000000` $= 2^{30}$, which represents the distance d_{min} to the nearest VCP with respect to all grid points on the boundary of the spatial region referred by that node. We compute this value by the formula $\lfloor \min\{d_{min}/D, 1\} \cdot (2^{30} - 1) + 2^{30} \rfloor$, where D represents a certain cutoff-value. Finally, non-empty leaves are stored as negative integers `-(leafIdx + 1)`, where `leafIdx` points to the index in the VCP leaf data structure, which is constructed as given in the next paragraph.

6.4.1 VCP Leaf Data

For each non-empty leaf, 2^3 VCP vectors have to be stored. Note that leaves can overlap if they belong to adjacent blocks in space as shown in Figure 6.6a. In this case, they share 4 VCP vectors, which allows us to reduce the memory requirement. For this, we sort the leaves by using the following algorithm. First, leaves are ordered by their corresponding octree level. Second, when leaves are on the same level (only in this case the possibility of adjacency arises), they are sorted in xyz-order, where without loss of generality the x-axis has the greatest length. Now, we construct the leaf data as a 3D texture in such a way that cells share their VCP vectors, wherever it is possible as illustrated in Figure 6.6b. Finally, the leaf indices are assigned to the indirection pool. For technical reasons, the cells in the texture are arranged in a cuboid-like fashion rather than linearly, i.e., roughly $\sqrt[3]{n} \times \sqrt[3]{n} \times \sqrt[3]{n}$ cells instead of $n \times 1 \times 1$ cells. Otherwise, access on the GPU by texture coordinates would suffer from insufficient floating point precision.

Another important aspect in achieving memory efficiency is the data type used to store VCP vectors. To reduce the memory requirement, we propose a custom data type, which consumes only 32 bit per VCP vector. We begin with transforming the VCP vector into spherical coordinates (r, θ, φ) given as the radius, azimuthal angle, $\theta \in [-\pi/2, \pi/2]$, and polar angle, $\varphi \in [0, 2\pi]$. We obtain the radius by normalizing the vector length with respect to a fixed maximum radius, i.e., $r = min\{\|pcs\| / r_{max}, 1\}$. To improve the precision for small radii, we apply the square root to the radius, i.e., $r \leftarrow \sqrt{r}$. Now the three components are stored as

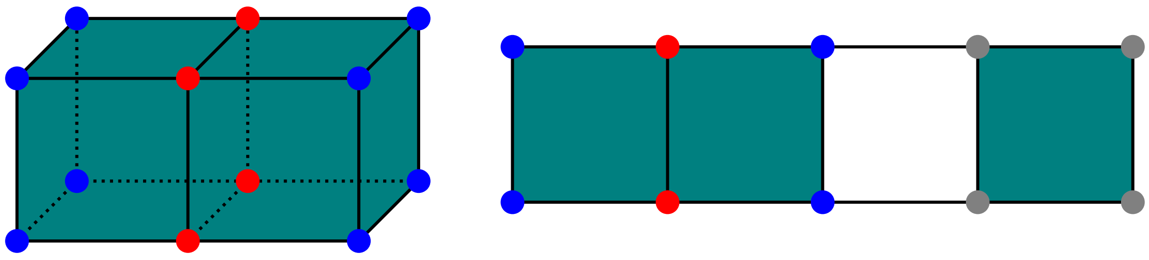


Figure 6.6: Left: Two adjacent blocks overlap at 4 grid points (red) meaning that they share the corresponding 4 VCP vectors, which allows us to reduce the required amount of memory. Right: Cells with overlapping points are placed next to each other. Between non-overlapping cells, there are empty regions (white background) which remain unused. Note however that by construction unused points never arise.

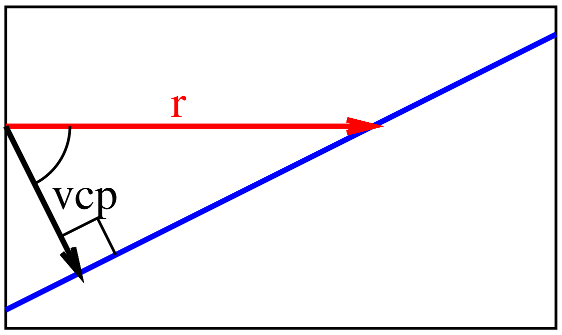


Figure 6.7: The step size can be increased, when considering the surface (blue) as locally planar. Given the angle between the vector *vcp* (black) and the ray direction r (red), the distance to surface along the ray can then be computed by applying the law of cosines.

normalized unsigned integers into 32 bit where r occupies the first 13 bits, θ the next 9 bits and φ the remaining 10 bits.

6.4.2 Reducing Run-time Memory Traffic

It is not necessary to traverse the octree at each step along the ray beginning from the root. Instead, we can simply remember the node index directly before the leaf. In many cases, we can reuse this index as starting point, in particular, when the step size is small, which is the real bottleneck of our technique. To further optimize the number of global memory accesses, we store at each non-empty leaf the 8 VCP values in local memory. This enables us to avoid reaccessing the same values in global memory, as long as we stay in the same cell, while marching along the ray, i.e., when the step size is very small and would hence require

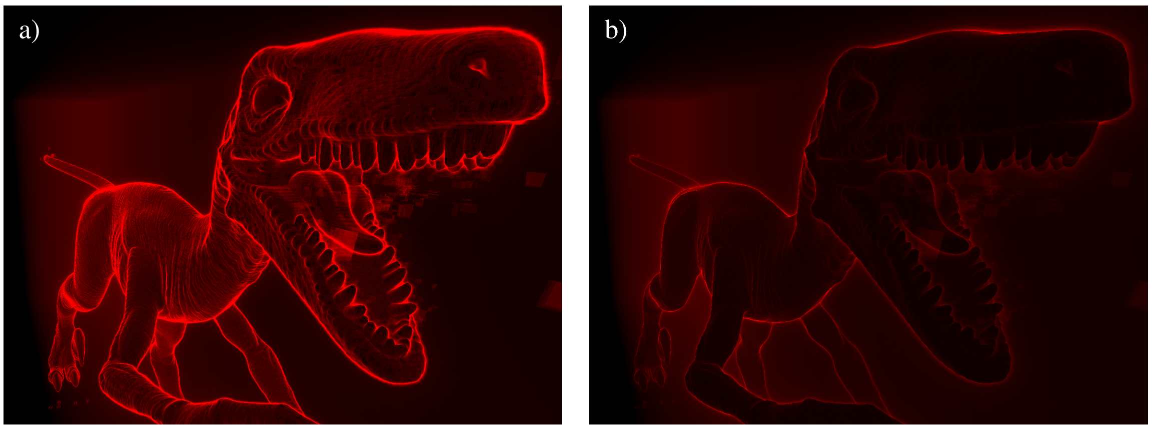


Figure 6.8: Comparison of the unoptimized (a) vs. the optimized approach (b). Here the number of accesses to global memory is encoded by color intensity. Clearly, the optimized version is much more efficient. Note that this also results in significantly greater frame rates.

a lot of memory look-ups. Finally, as illustrated in Figure 6.7, we can increase the step size. Considering the surface locally as planar would allow us to pick a step size of

$$s = d/\left(\frac{vcp}{d} \cdot r\right),$$

where r denotes the normalized ray direction. In practice, however, this leads to rendering artifacts because the surface is not planar on a larger scale. Therefore, we restrict the step size optimization by a threshold, in our case 2ϵ. Figure 6.8 shows a comparison of the optimized versus the unoptimized approach that clearly demonstrates its advantage. A similar algorithm was employed by Hart in the context of sphere tracing [Har96].

6.4.3 Dynamic Memory Management

To improve the rendering quality, we have developed a system that dynamically loads more refined subsets of the given geometry into GPU memory in places, which are located close to the camera and visible in the current viewport. For this, we divide the bounding box of the entire scene into subvolumes. Then, we consider each subvolume as a unique object and construct the VCP hierarchy by restricting the VCP grid on the respective region. During the rendering process, we continually compute the required set of subvolumes and load them into memory. Loading is done in the background, which exploits the multithreading capability of DirectX 11 and does not interrupt the rendering process. While ray-casting, we check at each step, whether there is a refined VCP hierarchy available in GPU memory as long as

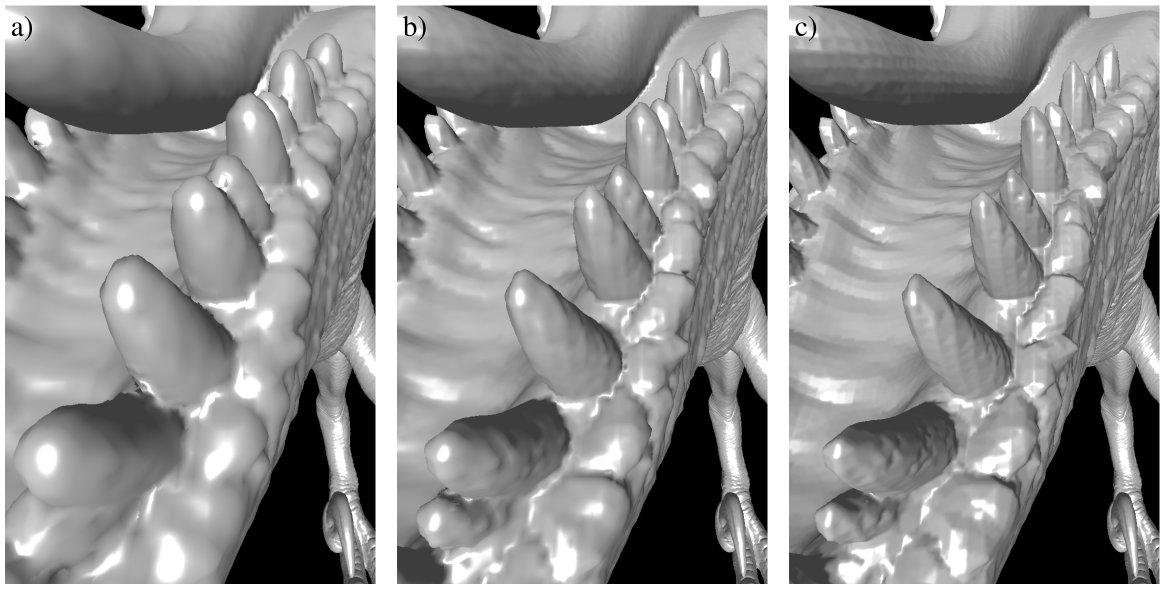


Figure 6.9: By dynamically loading more refined VCP hierarchies belonging to regions close to the camera, the rendering quality is significantly enhanced. Rendering was performed a) at a coarser resolution (to demonstrate the effect more clearly), b) at a finer resolution, c) including further refined subvolumes.

the distance to the camera is less than a threshold, in our case 1/4 of the diameter of the object's bounding box. In this case, we perform a look-up in the refined structure in the same way as explained in the previous section. The effect is demonstrated in Figure 6.9. As a minor drawback of this approach, one can sometimes spot popping artifacts upon switching to different subvolume levels. However, we regard this as acceptable, since our method dramatically reduces the runtime memory requirement on the GPU, thus enabling a significantly finer resolution. Note that it is also be possible to extend this approach to multiple levels of subvolumes, i.e., to further dividing subvolumes, although we have not implemented this method.

6.4.4 GPU-CPU Upstreaming

To determine, which subvolumes are currently needed, we identify the subvolumes hit by rays at early steps. While ray-casting, we compute at each step, which subvolume is currently hit. Then, we increase a counter associated with that particular subvolume by the number $1/(a+\epsilon)$, where a is the accumulated step size. Afterwards, the counters are ordered decreasingly, which results in a priority list. Finally, the associated subvolumes are loaded into GPU memory according to their priority. To minimize the number of interlocked accesses to global memory

Object	Triangles	VCP Res/Mem	Sgn Dist. Res/Mem	Voxel Res/Mem	SVO Res/Mem
Thai	10M	$2K \times 1K^2$ 1.05 GiB	$2K \times 1K^2$ 1.05 GiB	$4K \times 2K^2$ 1.07 GiB	$2K \times 1K^2$ 860 MiB
David	960M	$2K \times 1K^2$ 980 MiB	$2K \times 1K^2$ 980 MiB	$4K \times 2K^2$ 960 MiB	$2K \times 1K^2$ 770 MiB
Sponza	150K	$1K \times 512^2$ 290 MiB	$1K \times 512^2$ 290 MiB	$1K \times 512^2$ 350 MiB	N/A

Table 6.1: For different objects, the following data is shown: the number of triangles of the original mesh; resolution and memory consumption for different techniques.

(required for the counter to be consistent), we run a separate rendering process for this issue. Here, it is sufficient to use a very small texture as rendering target, in our implementation of size 24×16 pixels.

6.4.5 Interruption-Free Loading

Also, notice that the set of required subvolumes to be loaded into GPU memory varies at a significantly lower frequency than the frame rate. This is because the user typically moves the camera around only at a certain pace. We can exploit this fact by updating the set of required data significantly less frequently than regular rendering. However, even with this improvement, there still occur some interruptions that can be notified by the user from time to time. We address this issue by using the following strategy. For each frame, we measure the rendering time and only if it is less than two-thirds the average time in the last n frames, or after a certain timeout period is over, the process of gathering the required subvolumes is carried out. In our implementation, n equals the number of frames rendered in the last 2 seconds.

6.5 Results

In this section, we show several results of our method. Then, we compare our approach by performance and quality against ray-casting (1) on signed distance octrees, (2) on voxel octrees, i.e., binary voxel grids organized as octrees, and (3) the sparse voxel octree (SVO) implementation by Laine et al. [LK10b]. All measurements were performed on a desktop PC equipped with an Intel Xeon X5675 CPU at 3 GHz and an NVIDIA GeForce GTX 580 graphics adapter with 3 GiB of memory. A viewport of resolution 1920×1200 was used for all renderings. In Figure 6.10 renderings of the utilized objects are depicted. Table 6.1

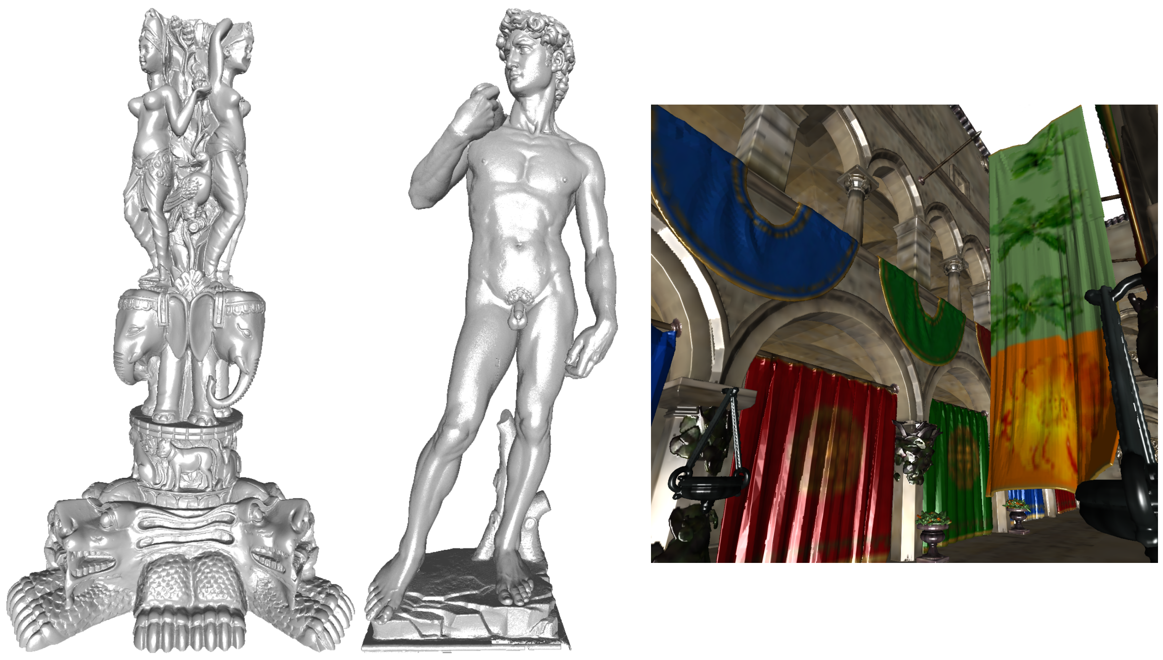


Figure 6.10: Objects used in our test cases. From left to right: Thai Statue, David, and Sponza Atrium. All images were rendered with the implementation of our method.

shows relevant statistical properties about our test scenarios. Note that the resolution for all techniques is chosen such that roughly the same amount of memory is consumed.

A comparison of the performance is presented in Table 6.2. Note that similar frame rates are achieved in all scenarios, which we attribute to the fact the same geometry is rendered by a ray-casting technique with roughly the same memory requirement.

Next, we study the different techniques with respect to the rendering quality they achieve. The results are shown in Figure 6.11 with close-ups in the bottom row. Our method preserves fine details better than the signed distances field for the previously mentioned reasons. Moreover, the use of voxel ray-casting produces clearly recognizable artifacts due to the underlying block structure of the voxel grid. The SVO method avoids such artifacts; however, in generally smoother regions finer details appear somewhat blurred. It should be noted that the quality of all techniques could be significantly increased by using a greater resolution.

6.6 Conclusion and Future Work

In this chapter, we have presented a novel rendering technique built upon a data representation based on Vectors-to-Closest-Points with respect to the geometry of the scene. By using

Scene	VCP	Sgn. Dist.	Voxel	SVO
Thai 1	8.6	8.2	8.0	8.1
Thai 2	11.7	11.1	10.6	11.0
Thai 3	20.4	19.8	19.4	18.2
David	9.1	8.7	8.2	8.9
Sponza	18.3	17.7	18.7	N/A
Raptor (Top)	16.7	16.2	15.9	14.2
Raptor (Bottom)	13.2	12.9	12.7	16.8

Table 6.2: Comparison of rendering times per frame. Each value is given in milliseconds. The Thai scenes belong to Figure 6.1 (from left to right). David and Sponza are (monochrome) renderings of the objects shown in Figure 6.10. The raptor scenes correspond to Figure 6.11.

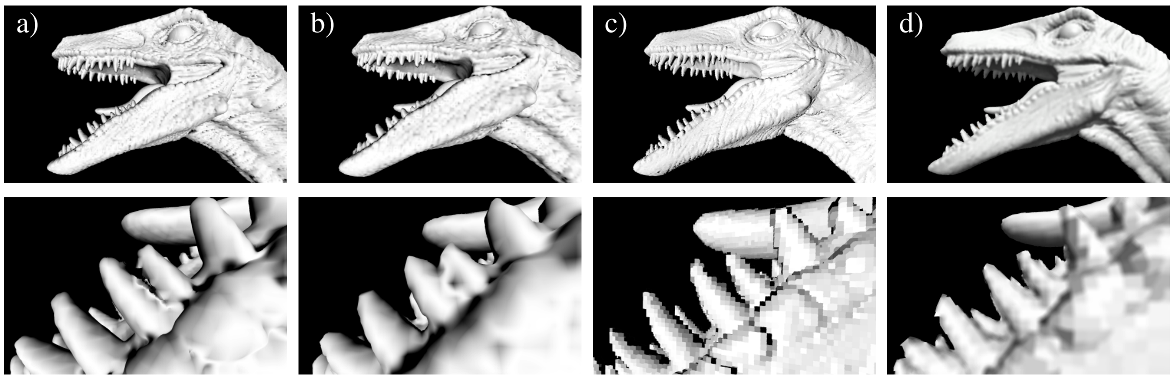


Figure 6.11: Comparison of rendering quality for different ray-casting techniques: a) VCP b) signed distance field c) voxel grid d) SVO. To demonstrate the differences, the resolution was deliberately chosen at a lower level, with the constraint that the memory consumption is roughly equal for each technique.

a regular grid as the underlying structure, we achieve an efficient GPU-implementation of ray-casting. Since the grid is organized in an octree-like fashion, the memory consumption is minimized. Furthermore, by relying on VCP vectors rather than employing voxel grids, our technique has the advantage of avoiding artifacts, which are inherent to all voxel-based methods. We have also presented an algorithm to generate VCP hierarchies, i.e., octree-representations of VCP grids, from polygon meshes in a computationally efficient manner. Our algorithm can be used for arbitrary large meshes by processing their faces sequentially, as we have demonstrated for the David statue. We have also proposed and implemented several ways to significantly enhance the performance of our ray-casting algorithm by exploiting the underlying VCP representation. To circumvent the memory restrictions on the GPU, we have implemented an algorithm that dynamically loads only a certain subset of the whole

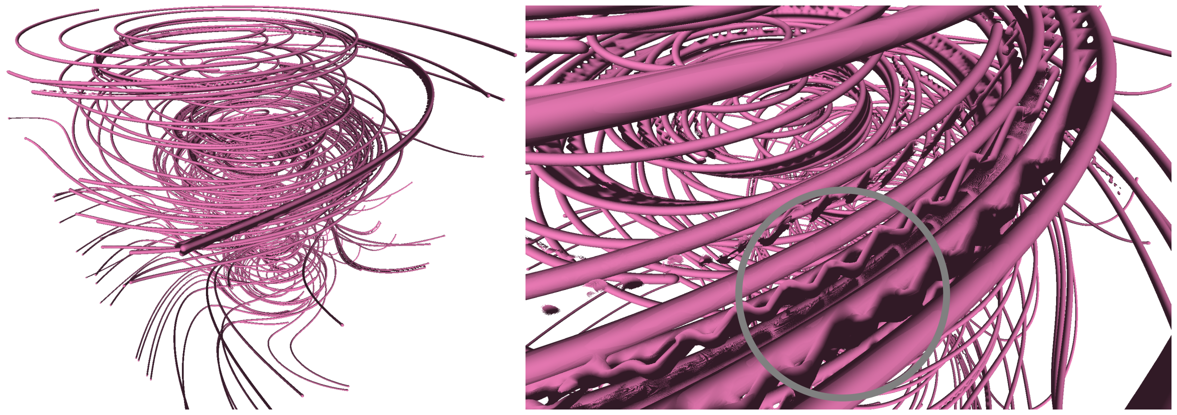


Figure 6.12: Left: Visualization of line fields by utilizing an underlying regular VCP grid. Right: Phantom lines occur, when VCPs at adjacent grid points exhibit a too high geodetic distance on the line set.

geometry at a finer resolution into GPU memory. This subset is determined by analyzing, which parts of the scene in the current viewport lie in close vicinity to the camera. Finally, we have presented various results that clearly demonstrate the potential of our approach in the context of interactively rendering large meshes.

The work presented in this chapter gives rise to various future research directions. Firstly, by implementing a DAG structure—inspired by Sparse Voxel DAGs [KSA13]—identical VCP regions could be reorganized. Thus, we can expect a significantly lower memory requirement, in particular for scenes exhibiting a high degree of self-similarity. Moreover, global illumination effects can be incorporated. For instance, by integrating non-camera rays, shadows and specular reflections can be rendered.

Our method might also be of interest for rendering semi-transparent structures, as the use of ray-casting renders computationally expensive sorting mechanisms in the fragment stage redundant. For instance, consider the case of visualizing dense line fields with varying (importance-based) opacity, as described by Günther et al. [GRT13]. Here, our approach might prove as helpful, in particular as the underlying regular VCP grid structure gives rise to fast opacity optimization strategies. An example of our approach in this context is shown in Figure 6.12 (left). However, there also exist some issues that have to be overcome, before this method can be further developed: For instance, when VCPs at adjacent grid points exhibit a too high geodetic distance on the underlying structure, phantom features occur even when using our interpolation scheme. This is shown in Figure 6.12 (right), where artifacts occur between two lines, thus giving the user a false impression about the real data set.

7 Visualization of Shape-Based Ensembles

In this chapter, we propose a novel approach for visualizing the central tendency of ensembles of shapes by building upon the vector-to-closest-point representation, which allows us to determine the most central shape, to quantify the region-wise centrality of the shapes, and to compute a locally most representative shape. This chapter is largely based on our publication:

Demir I., Jarema M., Westermann R.: Visualizing the Central Tendency of Ensembles of Shapes. In *Proc. SIGGRAPH Asia 2016 Symposium on Visualization* (2016). `doi: 10.1145/3002151.3002165`. [DJW16]. © 2016 Copyright is held by the owner/author(s). Publication rights licensed to ACM.

7.1 Introduction

Ensembles that are generated by repeatedly running simulations are commonly used to estimate the uncertainty in the prediction of physical quantities: First, a representative set of possible states of the quantity, which could evolve out of perturbed initial conditions and different models, is provided, i.e., the ensemble. Then, based on the variability of the ensemble members at a given point in time, the uncertainty of the current prediction can be estimated. By identifying a prediction that is a best representative of the whole ensemble, a central tendency can be communicated to the user. Spaghetti plots based on iso-contours are often used as a means to investigate the variability of a scalar field ensemble with regard to specific features in the data. In this scenario, the contours of all ensemble members at a particular time step are shown simultaneously by drawing overlaid spaghetti plot. Spaghetti plots, on the other hand, are subject to visual clutter when there are too many overlapping features, and they cannot express major trends and statistical attributes of the feature distribution

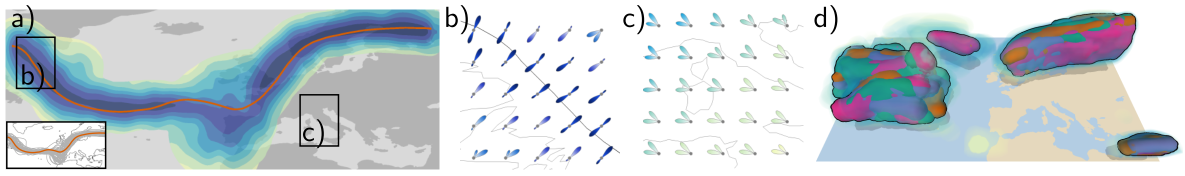


Figure 7.1: a) Visualization of a 2D ensemble of wind velocity fields. The median iso-contour determined by our method is shown in red, and the local centrality with respect to the ensemble is color coded from dark blue (high centrality) to yellow (low centrality). For the marked regions, (b) and (c) show the directional distributions of vectors to closest points that were used to classify the centrality. d) For a 3D scalar ensemble field, the locally best matching median surfaces are combined in one single shape. Colors indicate different ensemble members.

in an intuitive manner. To circumvent these restrictions, descriptive statistics has been utilized to illustrate the main features of contours, and to create abstract representations that condense these features effectively. Contour boxplots [WMK13] are built upon the model of statistical data depth to quantify the centrality of a contour surrounded by a set of contours. Contour variability plots [FKRW16] provide a statistical model of the distribution of contours and yield confidence intervals that stress their Euclidean spread.

Contour boxplots and variability plots generate indicator fields that allow a classification of locations with respect to the contours. However, to this end, they require contours to be closed and orientable. While these requirements are usually fulfilled by iso-contours, they are generally speaking violated for arbitrary curves and surfaces—and sometimes even for iso-surfaces. This can result in conflicting classifications, which can severely disturb the derived statistical abstractions.

In this chapter, we propose an alternative method for visualizing and studying the central tendency of shape-based ensembles, in particular, of ensembles of curves and surfaces, as illustrated in Figure 7.1. Our approach is motivated by the technique of Ferstl et al. [FKRW16], where they use signed distance fields as indicator fields. Closed shapes are defined implicitly as zero level-sets in these fields. In contrast, however, our method is built upon the vector-to-closest-point representation. In this representation, every location is characterized by a distance and direction to the closest point. Then, a shape is defined implicitly as the zero-vector level-set. We use such representations as indicator fields, which allows us to process arbitrary shapes and acquire an enhanced classification scheme of domain points with regard to an ensemble of shapes.

Our specific contributions in this chapter are:

- A classification scheme for domain points with respect to the distribution of vectors to closest points. We study ensembles of vector-to-closest-point fields and classify point-wise directional distributions by using mixtures of von Mises-Fisher distributions as an underlying probabilistic model.

- Finding the globally most representative shape in a statistical sense. By computing the classifications over the shape for each ensemble member, we determine the shape that is most central to the entire sample in a least-squares fashion.
- Region-wise quantification of the centrality of the shapes: Incorporating the directional distribution information into the visualization process allows us to classify the representative quality of the local central tendency.
- Construction of a locally most central shape. In addition to the globally most central ensemble member, we use the vector-to-closest-point distributions to determine the locally best matching shape. These piecewise defined sections are then combined to form a new shape.

We believe that our technique has great potential of enhancing and accelerating the investigation of shape-based ensembles, and that it opens up new prospects in the field of ensemble visualization. For instance, our method can be applied to ensembles of stream surfaces or deforming flow surfaces with immensely dissimilar topologies and also to ensembles of surfaces of material or anatomical structures, in order to generate atlas illustrations. For closed and consistently oriented contours, we show that our technique results in the same or very similar outcomes as existing approaches, and we further demonstrate that our approach is highly effective for exploring ensembles of 3D shapes and non-orientable shapes.

7.2 Related Work

Our method belongs to the class of uncertainty visualization. For recent surveys on this topic, let us refer to the summary article by Potter et al. [PRJ12] and the overview by Bonneau et al. [BHJ*14]. As we have mentioned before, when visualizing ensembles, the assumption is that the uncertainty is characterized by a set of possible data occurrences, rather than by a fully specified probabilistic model of uncertainty. Obermaier and Joy [OJ14] categorize ensemble visualization techniques into two approaches, based either on features or on locations. Our method can be understood as a hybrid approach, as we are using location-based summary statistics to explore the feature-based variability of specific shapes.

Spaghetti plots of iso-contours are a popular approach for conveying the uncertainty inherent in ensembles of 2D scalar fields [PWB*09a, Wil11, SZD*10]. Since the resulting visualizations are often prone to visual clutter, various simplifications and visual abstractions have been suggested. Sanyal et al. [SZD*10] introduced glyphs and graduated ribbons to express the uncertainty along iso-contours. Several kinds of confidence bands were proposed by Whitaker et al. [WMK13] and Ferstl et al. [FKRW16]. Their approaches are predicated on showing regions that exhibit the Euclidean spread of a set of iso-contours. Both methods use indicator fields to categorize locations in space with respect to an ensemble of contours. While the approach of Whitaker et al. [WMK13] is based on binary fields indicating inside and outside locations, Ferstl et al. [FKRW16] establish their method on the basis of probability theory, by utilizing the concept of standard deviation of signed distance functions. It is worth noting that both approaches require closed and consistently oriented contours.

Mirzargar et al. have extended the concept behind Contour boxplots to parametric curves [MWK14]. By using Principal Component Analysis (PCA), Ferstl et al. [FBW16] have proposed a method to extract the main geometric tendencies in ensembles of curves for the same set of curves.

Visualizing the uncertainty with regard to position and structure of iso-surfaces was previously achieved by drawing confidence envelopes [PWL97, ZWK10], surface displacements [GR04], as well as by animating over a sequence of surfaces [Bro04, LLPY07]. Poethkow and Hege have introduced the concept of numerical condition for the visualization of the variability of iso-surfaces implicitly contained in uncertain scalar fields [PH11]. A probabilistic approach based on level crossing for uncertain iso-surfaces that includes global correlations was suggested by Pfaffelmoser et al. [PRW11], and was integrated into techniques for extracting surfaces from ensembles unveiling local correlation structures [PWH11].

Ferstl et al. have used representations based on signed distance functions for the purpose of ensemble-wise classification of domain points [FKRW16]. Initially, such representations were proposed by Gibson to obtain a higher-order interpolant of surfaces that are embedded in 3D scalar fields [Gib98]. For a comprehensive overview of applications of signed distance transformations and algorithms for computing such functions from polygon models, refer to Jones et al. and Frisken et al. [JBS06, FPRJ00]. By building upon signed distance functions, Bruckner and Möller [BM10a] have developed a metric for the comparison of iso-contours in a scalar field. Rathi et al. [RDT06] and Leventon et al. [LGF00], respectively, use PCA on distance functions for analyzing shapes and for the construction of statistical shape models. By applying dimensionality reduction techniques to ensembles of distance functions, Fofonov et al. represent contour time series as trajectories in a feature space of lower dimension [FML16].

Closest point representations, where a surface is implicitly encoded in a grid structure via vectors to closest surface points, have been used in various approaches for solving differential equations on surfaces [RM08, MR08]. In the context of fluid simulation, Auer et al. [AMT*12] have proposed a volume rendering technique based on ray-casting directly on a uniform closest point grid. In Chapter 6, we improved upon this technique by introducing an hierarchical

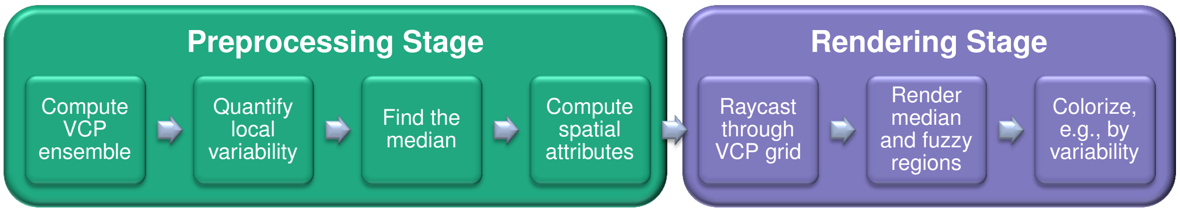

Preprocessing Stage
Compute VCP ensemble
Quantify local variability
Find the median
Compute spatial attributes
Rendering Stage
Raycast through VCP grid
Render median and fuzzy regions
Colorize, e.g., by variability

Figure 7.2: Overview of our method.

closest point representation and a tailored interpolation scheme that considers the closest point geometry in order to intersect rays with a zero-vector level-set of closest point vectors more precisely.

7.3 Method Overview

Our method starts with an ensemble $\{\mathbf{s}_1, ..., \mathbf{s}_N\}$ of N 2D or 3D shapes, the ensemble members. Without loss of generality, we assume that shapes are given either as polygonal curves or surfaces. Nevertheless, any other representation is possible, as long as for a given location we can compute the point closest to that location on the shape. In addition, notice, that each ensemble member can be comprised of multiple shapes, i.e., one ensemble member can be represented by multiple disjoint shapes. Such data sets can be generated by extracting isosurfaces or isocontours from scalar fields, although we do not require the shapes to be closed or orientable.

Starting with the initial ensemble, our method proceeds in two stages: The preprocessing stage and the rendering stage (see Figure 7.2).

In the preprocessing stage, we first generate the bounding volume enclosing all shapes, and we discretize this volume using a Cartesian grid structure. If the shapes were extracted from values on a Cartesian grid, e.g., the shapes are level-sets in scalar fields given on such a grid, we use the resolution of the initial grid to discretize the volume. Otherwise, we try to match the resolution of the input shapes, i.e., we adapt the grid resolution to the smallest features represented by the shapes.

For each shape and at each grid vertex, the vector to the closest point on this shape is computed. This results in an ensemble of vectors to closest points at each vertex. We subsequently call this grid the vector-to-closest-point ensemble (VCPE) grid. For the computation of the VCPE grid, we basically utilize the algorithm presented in Chapter 6 and follow the GPU implementation by Auer et al. [AMT*12], with few adaptations to account for the fact that the closest point computations are not restricted to a narrow band around the surface. Therefore, the shape is first partitioned using a regular grid. One GPU thread then computes the closest point of exactly one grid vertex, by sequentially going over the cells of this grid in increasing distance of the cell centers to the vertex position. For each cell, the closest point to the shape contained in this cell is computed, for instance, by iterating over all lines or triangles of the polygonal structure. Processing of cells is stopped once the current closest point is closer than the closest corner of the next cell. Since many grid vertices can be processed in parallel on the GPU, the computation time was always below 5 seconds even for the largest shapes and grid resolutions.

The ensembles of vectors at each grid vertex are now analyzed regarding their local directional variability. Based on a statistical model describing the directional distribution of these vector ensembles, we quantify the local variability at each vertex, and, in particular, we derive a measure quantifying how central the position of the respective grid vertex is to the surrounding closest points. Intuitively, a vertex is most central, i.e., it is suitable as a median point, if its position coincides with the geometric center of the surrounding closest points and it is possible to order these points along a single direction. The rationale underlying this approach is to compute a measure of centrality that depends on both the distance to the closest points and the modality of the directional distribution. For instance, a bimodal distribution with an equal number of vectors captured by each mode and the modes being oriented inversely to each other indicates perfect centrality. Figure 7.3 shows different cases that are relevant in this regard. We now identify the most central shape by minimizing its overall centrality to the other members. Finally, we generate local statistical attributes to convey the spatial distribution of local centrality to the user.

Optionally, we now utilize a clustering algorithm to group members by their similarity. This is particularly helpful, if the ensemble exhibits many different facets, such that identifying a single member as a most central shape is impracticable, or results in a too high degree of uncertainty, respectively.

The precomputed data are then visualized interactively using GPU volume ray-casting, by either sampling the VCPE grid along the rays of sight, or performing a single lookup in case of 2D shapes. We provide options to render the median shape by testing for intersections with the most central shape, or fuzzy structures surrounding the median to convey the local centrality. By using color to encode the centrality information, the user is able to discover uncertainty and to identify outliers.

7.4 Generating the Vector-to-Closest-Point Ensemble

Let us consider an ensemble of shapes, specified by an index set $I = \{1, \ldots, N\}$. Now, we construct an ensemble of VCP volumes, where each shape results in one volume. As our implementation is based on the Vector-to-Closest-Point (VCP) octree representation, we only provide a short recap of that method, which was introduced in Chapter 6.

- Starting with an empty Cartesian grid at a lower resolution, we iterate over all primitives, e.g., faces, of the given geometry and update all VCPs within its domain. That is, at each grid point, we compute the VCP to the current primitive and store it, if its distance is less than the previously computed VCP distance or if the grid point is still marked as empty. Next, we utilize a region-growing algorithm. By repeatedly traversing over all grid points, we update each point by comparing its distance to the distances to primitives belonging to the surrounding grid points. The process is finished, when no more changes occur. As a result, we end up with a completely filled VCP grid, which constitutes the root level.
- We now construct the tree structure, i.e., an octree or quadtree, by refining the VCP grid in a top-down fashion. Each grid cell is subdivided, if interpolating at one of its inner grid points at the next finer level deviates more than a given threshold, t, from the true VCP vector. Note that quality and memory consumption are inversely proportional to the value of t. The fundamental idea behind the interpolation scheme for VCP vectors is to consider the underlying geometry implicitly given by the closest points. That is, at the interpolation point, we compute the VCP to all planes spanned by the VCPs at the surrounding grid points. The shortest vector is then compared to a linearly interpolated VCP vector, and of these two vectors the one with the greater length is returned as the result.

7.5 Modeling Vectors to Closest Points

Given a VCPE grid, the vectors to closest points at each grid point are used in two different ways: Firstly, their directional distribution is modeled statistically to enable quantifying the local directional variability and, thus, deriving indicators of the centrality of each grid point with respect to the ensemble of shapes. Secondly, the derived models are used to determine the most central member, i.e., the median. Unlike generating a statistical mean member, our algorithm guarantees that the median is an existing member of the initial ensemble. Thus, our technique conveys information to the user, which truly exists in the original data set.

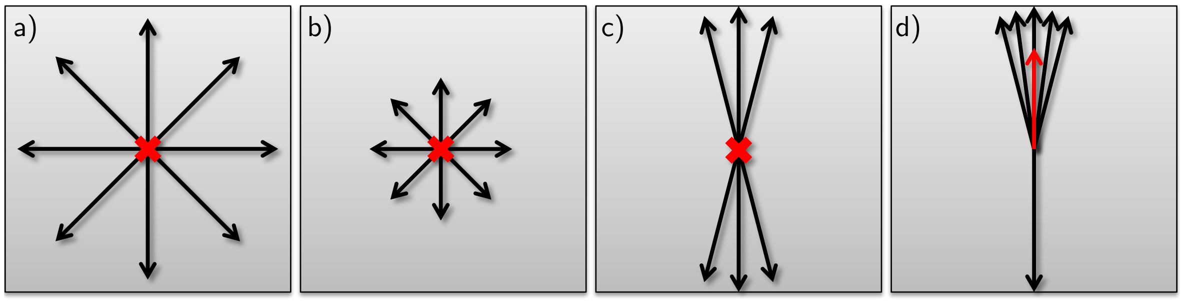


Figure 7.3: Different cases relevant for centrality quantification. a) With a uniform distribution, placing closest points along the same direction is not possible, resulting in a low degree of centrality. b) Points are lying closer together, which implies that the deviation decreases, and thus centrality is greater. c) Here, ordering along a single direction and finding a suitable choice as the median is possible. d) Although these closest point vectors can be ordered, the current point represents a poor choice, as it deviates significantly from the mean vector to closest point (colored in red).

7.5.1 Quantifying the Local Centrality

At each grid vertex, we compute a value representing the local centrality. This value quantifies, how well the respective point in space is suitable as a median point. Intuitively, this happens, if the point coincides with the geometric center of the surrounding closest points, and if it is possible to order these points along a single direction. Figure 7.3 shows different cases that are relevant in this regard.

Let $\{vcp_i(p) : i \in I\}$ denote a set of $N = |I|$ closest point vectors at a grid vertex p. Moreover, let $d^{\star}_{\text{Cutoff}}$ denote a global cutoff-value with respect to the length of all closest point vectors, which is used to scale the vectors to unit length. Now, we consider three criteria indicating whether p can be deemed suitable for a point of the most central shape, each giving a value ranging between 0 (least suitable) and 1 (most suitable):

Bimodality Angle

The point is likely to be representative, if the directions of the closest point vectors can be modeled by two clusters having means with approximately the same direction, but contrary orientation. To find out, if this is the case, we model the distributions of VCPs using mixtures of probability density functions (PDFs). In doing so, we can characterize the directional distributions with relatively few parameters: For each mixture component, its mean, the variation around this mean, and the weight of the component. A suitable mixture model for spherical data is a mixture of von Mises-Fisher (vMF) components. In 2D, this reduces to the von Mises distribution on the circle. In the following we restrict the discussion to the 3D case. For the 2D case, let us refer to Fischer [Fis95].

A unit vector v follows a mixture of vMFs if its PDF is given by

$$f(v) = \sum_{i=1}^{M} \alpha_i \frac{\kappa_i}{4\pi \sinh(\kappa_i)} \exp\left(\kappa_i \mu_i^T \mathrm{v}\right), \quad \alpha_i > 0, \ \sum_{i=1}^{M} \alpha_i = 1,$$

where M is the number of vMF components, the unit vectors μ_i are their mean directions, κ_i the concentration parameters, and α_i the weights of the components. The concentration parameter defines the shape of the distribution, with greater values indicating stronger concentrations in the vicinity of the mean direction. The parameters of a mixture of vMFs are estimated using an implementation of the EM soft-moVMF algorithm introduced by Banerjee et al. [BDGS05].

To qualitatively estimate the bimodality value, we strictly fit two vMF components. For clearly unimodal distributions, one of the two weights α_i will be 0; hence, the bimodality value is also 0 and no further calculations are performed. If this is not the case, then we compute the bimodality value as the smallest angle between the confidence cones of the two components, normalized by π. Uniform distributions are also fitted using two components, but the wide confidence cones result in very small bimodality values. To obtain a confidence cone around each mean direction μ, we use two approaches [FLE87], depending on the size n of each component. For $n < 25$, we need to apply a bootstrap technique. When $n \geq 25$, a simpler method is available. Calculating the mean resultant length $\overline{R}$ and estimated spherical standard error $\tilde{\sigma}$ of the sample mean direction

$$\tilde{\sigma}^2 = d / \left(n\overline{R}^2\right), \text{ where } d = 1 - \frac{1}{n} \sum_{i=1}^{n} (\mu \cdot vcp_i)^2,$$

a 95% confidence cone for μ has a semi-vertical angle equal to $q = \arcsin(1.7308\tilde{\sigma})$.

Visualizing the directional distributions, as can be seen in Figure 7.1bc and Figure 7.7a, is achieved by drawing oriented glyphs, as proposed by Jarema et al. [JDKW15]. At each grid point, the corresponding mixture of vMFs is represented by at most two lobes (one lobe per component), where the mean direction, weight, and confidence cone of each component are mapped to the orientation, length, and opening angle of the corresponding lobe.

Length of Mean VCP

For this quantity, we first determine the mean closest point vector

$$\overline{vcp}\,(p) = \frac{1}{N} \sum_{j=1}^{N} vcp_j\,(p)\,.$$

Note that the geometric center of any given set of points $x_i \subset \mathbb{R}^D, i \in I, D \in \mathbb{N}$ in Euclidean space, is obtained as

$$c = \frac{1}{N} \sum_{j=1}^{N} x_i.$$

That is, the mean VCP points to the geometric center with respect to the closest surface points of all members at any given position. Consequently, we can compute the distance from p to the closest point on a locally optimal median shape as the length of the mean closest point vector. This holds, first, since the geometric center minimizes the mean squared Euclidean distance to each point. Second, because the closest point vector of the potential median member vanishes at all of its surface points. Third, because this vector's length is the shortest distance to its underlying shape. We now compute the result as the scaled length of the mean closest point vector. The Saturate function is here used to restrict the value to the range $[0, 1]$.

$$\hat{\mu}\,(p) = \text{Saturate}\left(\frac{1}{d^{\star}_{\text{Cutoff}}} \cdot \|\overline{vcp}\,(p)\|\right)$$

We consider this value in order to decide to which extent the point p can be regarded as good representation of the geometric center. Here, smaller values correspond to better degrees. Consequently, we use the value $1 - \hat{\mu}$ to determine the suitability as median.

Maximum Length of VCPs

This value represents the scaled length of the closest point vector with the greatest Euclidean length, namely,

$$\hat{d}_{\text{Max}}\,(p) = \text{Saturate}\left(\frac{\max\left\{\|vcp_i\,(p)\| : i \in I\right\}}{d^{\star}_{\text{Cutoff}}}\right).$$

This value indicates how closely together the closest points of all members are located. Again, smaller values stand for better degrees, because this implies that the maximum deviation between all members decreases. In particular, if the maximum length is equal to 0, all members are locally identical, meaning that any member is a perfectly good choice as the median. Hence, we use the value $1 - \hat{d}_{\text{Max}}$ to determine the suitability as median.

Criterion	**Variable**	**a)**	**b)**	**c)**	**d)**
Bimodality	φ	0.0	0.0	0.9	0.95
Mean length	$1-\hat{\mu}$	1.0	1.0	1.0	0.2
Maximum length	$1-\hat{d}_{\text{Max}}$	0.0	0.7	0.0	0.0
Centrality	σ	0.0	0.7	0.95	0.44

Table 7.1: For the cases in Figure 7.3, the respective values of the different criteria are shown. The last row contains the resulting centrality as computed by our method. In all cases, our method coincides with the intuitive understanding of a good median choice.

We compute the local centrality based on the aforementioned criteria as

$$\sigma = \max\left\{1-\hat{d}_{\text{Max}}, \sqrt{\hat{\varphi}\cdot(1-\hat{\mu})}\right\}$$

This formula accounts for all criteria and turned out to produce meaningful results in our test cases. The values corresponding to the cases in Figure 7.3, as well as the resulting qualities are shown in Table 7.1.

7.5.2 Finding the Median

To find the median, we calculate a global deviation with respect to the whole ensemble for each member. The member with the smallest deviation in a least-square sense is then selected as the median. For each ensemble member $i \in I$, we first select a sufficiently large set of points, P_i. For polygonal meshes or line strips, we use the original vertex positions for that purpose. Now, we iterate over all ensemble members $j \in I$ and over all surface points $p_i \in P_i$. For each run, we gather the vectors $vcp_j\left(p_i\right)$, i.e., the closest point vectors at all given surface points with respect to all members. Here, $vcp_i\left(p\right)$ denotes the closest point vector at point p for member i. Gathering is performed by bi- and tri-linear interpolation in the VCPE grid. Based on the set of interpolated vectors, we quantify the local centrality, $\sigma\left(p_i\right)$, described in the previous section.

We then obtain the global median deviation Δ_i as the expected squared local centrality at a uniformly randomly selected position on the shape's surface. Note that it is not sufficient to simply average over the centrality values, since the points p_i are not necessarily distributed uniformly, which would result in assigning unequal weights to regions that are sampled at a denser or less dense resolution than the average. As an example, consider a shape given as an adaptively resolved triangle mesh, where the vertex positions are used as the set of surface points. Even if derived from a uniform Cartesian grid, the surface points need not be resolved uniformly, as depending on the underlying geometry, distant closest point vectors

Algorithm 1: Finding the Median

```
Input: Ensemble of $N = |I|$ vector-to-closest-point grids $vcp_i$
Result: Median index $i \in I$
/* Select surface points                                              */
for $i \in I$ do
    $P_i \leftarrow$ surface points of member $i$
end
/* Iterate over all members                                           */
for $i \in I$ do
    /* Compute centrality                                             */
    for $p_i \in P_i$ do
        $\sigma(p_i) \leftarrow \text{Centrality}\{vcp_j(p_i) : j \in I\}$
    end
    /* Insert into uniform grid                                       */
    Choose resolution $r \in \mathbb{N}^d$
    $C_i \leftarrow \emptyset^r$
    for $p_i \in P_i$ do
        Let $c_i \in C_i$, such that $c_i$ contains $p_i$
        $c_i \leftarrow c_i \cup \sigma(p_i)$
    end
    /* Compute global deviation                                       */
    $\Delta_i \leftarrow 0$
    $weight(\Delta_i) \leftarrow 0$
    for $\emptyset \neq c_i \in C_i$ do
        $\Delta_i \leftarrow \Delta_i + \sum_{\sigma \in c_i} \sigma / |c_i|$ $weight(\Delta_i) \leftarrow weight(\Delta_i) + 1$
    end
    $\Delta_i \leftarrow \Delta_i / weight(\Delta_i)$
end
return $\arg\min\{\Delta_i : i \in I\}$
```

can point to locations close to each other. To overcome this issue efficiently, we proceed in the following way. First, we insert the squared centrality values into a Cartesian grid of lower resolution, such that each cell c_i contains the average of all values $\sigma(p_i)^2$, for all p_i belonging to c_i. Second, we compute the average over all cells containing at least one value C_i, and return the result as the global deviation with respect to the ensemble, i.e.,

$$\Delta_i = \frac{1}{|C_i|} \sum_{c_i \in C_i} \sigma(c_i).$$

Here, the key idea is to choose the grid resolution such that both the distribution of points within each cell and the distribution of the cells near the initial shape's surface are roughly uniform. We achieve this by picking the finest resolution such that the largest primitive of the original geometry fits into one cell, with the restriction that it is at most as fine as the underlying VCPE grid. Alternatively, we could utilize a binary search between 1 and the grid resolution in each dimension. Finally, we obtain the median index as the member with the least global deviation, i.e.,

$$\text{Median}\left(I\right) = \arg\min\left\{\Delta_i : i \in I\right\}.$$

A complete overview is shown in Algorithm 1.

7.5.3 Spatial Attributes

We now discuss the generation of statistical spatial attributes, which are presented to the user as fuzzy structures in addition to the median shape. In our implementation, the data are stored in the VCPE grid in addition to the index of the median shape and the local centrality data. This allows us to render simultaneously the shape and the fuzzy centrality volume during ray-casting. We begin with computing a fuzziness at each point of the VCPE grid. This value determines during rendering, whether a fuzzy or crisp structure should be drawn at a respective point, as well as its opacity. For this purpose, we compute two different values.

The minimal distance is given as the minimum of the Euclidean distances to all ensemble members from a given point in space. Note that we can easily obtain this distance as the length of the corresponding closest point vector, and thus we have

$$d_{\text{Min}}\left(x\right) = \min\left\{\left\|vcp_i\left(x\right)\right\| : i \in I\right\}.$$

Using the minimal distance as the fuzziness allows the user to see all regions where at least one member resides.

Using the minimal distance enables the user to identify all regions of potential interest. However, it does not reveal how many members cross a certain region, i.e., the density of shapes at a given point in space. To compensate for this, we propose a weighted distance value, given as an adjusted mean of all distances,

$$d_{\text{Weighted}}\left(x\right) = \frac{\sum_{i=1}^{N}\left(\left\|vcp_i\left(x\right)\right\| \cdot \exp\left(-\alpha \cdot \left\|vcp_i\left(x\right)\right\|\right)\right)}{\sum_{i=1}^{N}\exp\left(-\alpha \cdot \left\|vcp_i\left(x\right)\right\|\right)},$$

where $\alpha \in \mathbb{R}^+$ denotes an importance parameter giving additional weight to smaller distances. This allows us to retain regions containing outliers, whereas choosing α too small would result in smoothing out fine structures distant to most other members. On the other hand, choosing α too large would produce results similar to the minimal distance measure. In our test cases,

$\alpha \approx 3$ yielded good results. Using the weighted distance criterion enables the user to quickly identify outliers, as these are regions characterized by a relatively low fuzziness.

To finally store this data together with the VCPE grid, i.e., to extend the VCPE grid, we extend the concept of closest point vectors by three scalar components, each encoded by 8 bits in our implementation. The minimum and weighted distances are stored in the first two components. The local variability at the respective point in space is stored in the third component. This value is later used to colorize the cloud accordingly.

7.6 Clustering

As the shapes are given in two or more dimensions, it is often impracticable to single out a shape that can be selected as a most representative member. To remedy this problem, we utilize a hierarchical clustering algorithm, where members are first grouped into clusters and after that, we pick a median for each cluster. In the process of rendering, clusters are overlaid, such that the spatial context is preserved. By using color and transparency, the user can easily distinguish between different clusters. Note that the number of clusters has to be limited to avoid occlusion artifacts. Therefore, the optimal number of clusters is a trade-off between minimizing the global deviation and the visualization's perceivability.

We will now give a short overview of the clustering method used in our implementation. For further details, refer to Jain [Jai10]. For N members, we first generate a similarity matrix of size $N \times N$, where the similarity with respect to the underlying shapes i and j is stored at entry $(i, j), i, j \in \{1 \ldots N\}$. We compute the similarity of two members by comparing their VCPs at a uniform grid. For each grid point p, we sum up the squared distance between the two vectors and return the total value as the similarity measure, namely

$$d(i, j) = \sum_p \|cp_i(p) - cp_j(p)\|^2 .$$

We now create for each member a new cluster and proceed by merging pairs of clusters until only one cluster remains. In each step, two clusters of minimal distance are combined. For this, we determine the distance in the following way. If each cluster contains a single member, we use the distance from the similarity matrix. Otherwise, we calculate the distance according to a linkage criterion, see [Bij73]. For our purpose, complete linkage clustering, defined as

$$D(I, J) = \max\{d(i, j) : i \in I, j \in J\},$$

and average linkage clustering, defined as

$$D(I,J) = \frac{1}{|I| \cdot |J|} \sum_{i \in I, j \in J} d(i,j),$$

turned out to produce useful results with respect to minimizing the variability of each cluster. As this process is carried out as a precomputation step, the user can interactively choose between different numbers of clusters and linkage criteria.

For each cluster distribution, we compute the corresponding medians. Here, we proceed as explained above, with the exception that computing the variability is restricted to the members of the respective cluster. The distribution with the least total deviation is then preselected in the visualization, although it can be changed by the user at any time.

7.7 Ensemble Visualization

To visualize the median shape in combination with the local centrality information, we use volume ray-casting to sample the extended VCPE grid along the view rays. Since both the closest point vectors and centrality values are stored in the same grid, visualizing both structures can be performed simultaneously in one rendering pass.

Ray-casting the VCPE grid is performed in essentially the same way as proposed in Chapter 6. Thus, we give only a brief overview of this approach and explain the differences to this implementation. To render the median shape via ray-casting, we march along the view rays through the VCPE grid and interpolate the closest point vectors corresponding to the median shape at the sampling points. For this, we use the same interpolation scheme as described before. Now, we consider the length of the closest point vector. If it is less than a given threshold $\varepsilon > 0$, we assume a hit with the shape and compute the color by a local illumination model. At this point, the approach presented in the previous chapter terminates the ray traversal. However, as we include transparency effects to visualize the centrality information, we continue the process until the accumulated opacity reaches a prescribed threshold or the ray position lies outside of the VCPE grid. In addition, the following visualization options are provided.

7.7.1 Local Best Median

While traversing the VCPE grid along the rays, we test for the maximum centrality value of all ensemble shapes. If this value is greater than a user-selected threshold, the shape for which the value is determined is visualized, i.e., a unique color assigned to this shape is rendered and the ray is terminated. In this way we obtain a shape composed of many different shapes, depending on which of these shapes represent the median best locally. This combined shape is one that does not exist in the initial ensemble, but allows determining local trends and conveying the ensemble member that best follows the local trend.

7.7.2 Fuzzy Regions

To visualize the ensemble centrality around the median, we render a semi-transparent fuzzy structure, similar to the approach by Grigoryan and Rheingans [GR04]. The fuzziness, mapped to opacity, is computed according to a user-selected criterion: Either the minimal distance, d_{Min}, or the weighted distance, d_{Weighted}, can be picked by the user. In the first case, we choose a threshold $d^{\star}_{\text{Min}}$ such that the structure is rendered only if the minimal distance is less than the diameter of a grid cell. For this, we set the fuzziness to 1, if $d_{\text{Min}} < d^{\star}_{\text{Min}}$, and to 0, otherwise. In doing so, we ensure that fuzzy structures only occur where initial members exist. By determining the fuzziness according to d_{Weighted}, the user is able to explore the density of shapes in each region. Again, to avoid visualizing non-existent data, we combine both the weighted and the minimal distance in this case, i.e., we set the fuzziness to d_{Weighted}, if $d_{\text{Min}} < d^{\star}_{\text{Min}}$, and to 0 otherwise. For coloring the fuzzy structures, we use the local centrality, mapped from dark blue (high centrality) to yellow (low centrality).

7.7.3 2D Ensembles

We provide the option to visualize 2D VCPE grids. Therefore, a grid of depth 1 is generated, and one single lookup per ray retrieves all the necessary information to determine, whether the median shape is hit and what the local centrality at the sampling points is. This information is then color-coded.

7.7.4 Color

We enhance the visualization by using color to encode additional information about the original data set. For this purpose, we make use of the precomputed values stored together with the closest point vectors. For coloring the fuzzy structures, we can use either the local centrality or the weighted distance, depending on the user's interest. By using the weighted distance, the user immediately perceives the density of members in each region. Coloring the shape is done by using the centrality, which enables the user to identify regions where the median is a poor representation. Note that such regions can exist, as we always use an existing member from the original data set as the median. Alternatively, we could also color the shape by a single color, which is particularly helpful if more than one shape should be drawn, e.g., if medians of different clusters should be visualized at once. Finally, we apply a local illumination model, i.e., Lambertian lighting, to enhance the spatial perception.

7.7.5 Transparency

In addition to using color, we have integrated transparency effects in our visualization, by using front-to-back α-compositing. Transparency is mainly used to render the fuzzy structures, where the value of opacity corresponds to the fuzziness as explained, scaled by a user-selected value. In doing so, relevant information about the spatial distribution of all members is conveyed to the user without distracting her by visual clutter or occlusion artifacts. We can also assign transparency to the shapes, which is useful if multiple shapes are rendered or if one single shape contains interior structures. Note, however, that this is limited to a small number of shapes as otherwise occlusion effects would prevail. We also allow the use of transparency as a means to visualize the shape's centrality. In this case, regions of lower centrality are drawn with a lower opacity, allowing the user to focus specifically on regions that are representative for the whole ensemble.

7.7.6 Silhouettes

To enhance the user's recognition of the shapes, especially if fuzzy structures are visualized in front of them, we extend our rendering technique by adding silhouettes to the shape. They are blended over the shapes, such that they are never obscured by other structures. To detect silhouettes, we extend the ray-casting algorithm by a second threshold $\varepsilon^\star$, given by

$$\varepsilon^\star = \varepsilon \cdot (1 + \tau \cdot \lambda) \,,$$

where τ denotes the user-selected silhouette thickness and λ the distance traveled along the ray, starting from the camera position. We now proceed as follows. If, while marching along the ray, no silhouette was hit so far, and the current closest point length is less than $\varepsilon^\star$, we consider this a silhouette hit and store the distance to the camera, i.e., λ_0. If a surface hit occurs in the following steps, we check whether the current distance to the camera is close enough to the silhouette hit position, i.e., $\lambda < \lambda_0 + \varepsilon$. In this case, the silhouette is discarded, since it does not lie on the boundary of the shape. By letting the silhouette threshold depend on λ, we ensure that all silhouettes are of roughly equal thickness, regardless of their distance to the viewer.

7.8 Results

To demonstrate the practical application of the proposed method, we discuss two synthetic data sets, two real-world weather forecasting cases that occurred during an atmospheric research campaign, and an ensemble of 3D fluid simulations. The first synthetic data set is comprised of 50 Möbius bands, each represented by a 5K triangle mesh, which were slightly perturbed to simulate an ensemble with changing geometry. The second synthetic ensemble comprises 50 polygonal spheres of 4K triangles each, which have exactly the same geometry in most regions, but turn inward and outward with varying strength and shape in one of the hemispheres. Both synthetic data sets were discretized on a VCPE grid of resolution $128 \times 128 \times 128$. The 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. Our example region covers the North Atlantic and Europe, encompasses $335 \times 135 \times 62$ voxels, and comprises geopotential height, a typical measurement variable in weather forecasting. 2D and 3D polygonal iso-contours were first extracted from the physical fields, resulting in up to 16K triangles per surface, and these contours were then used as input to our approach. Our last ensemble features a Navier-Stokes fluid simulation around an ellipsoid obstacle. Here, 56 simulation runs with different viscosities were performed at a grid resolution of $145 \times 49 \times 49$. The vorticity magnitude of each run was written out as a scalar field, and polygonal iso-surfaces were extracted from each ensemble member.

All presented results 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 the VCPE grids, including the statistical modeling of the directional distributions per vertex, took less than 15 seconds on the GPU in all of our experiments. This time scales linearly in the number of grid points and the number of lines or triangles to which closest points have to be computed.

In Figure 7.4, we demonstrate the use of our method to find a median surface in an ensemble of non-orientable 3D surfaces, and to visualize the local centrality over the given surfaces with respect to the median. As can be deduced from the surface coloring, the computed median has very high centrality across the entire shape, while the other shapes show locally low centrality, which is indicated by the color shift towards yellow.

In Figure 7.5, we use our method to analyze the central tendency in the ensemble of spheres. Note that in this ensemble, roughly $^3/_4$ of all spheres were perturbed outward, and the remaining spheres were perturbed inward. Firstly, we show a rendering of the fuzzy centrality volume to indicate the different shapes of the ensemble members. It can be seen that, in

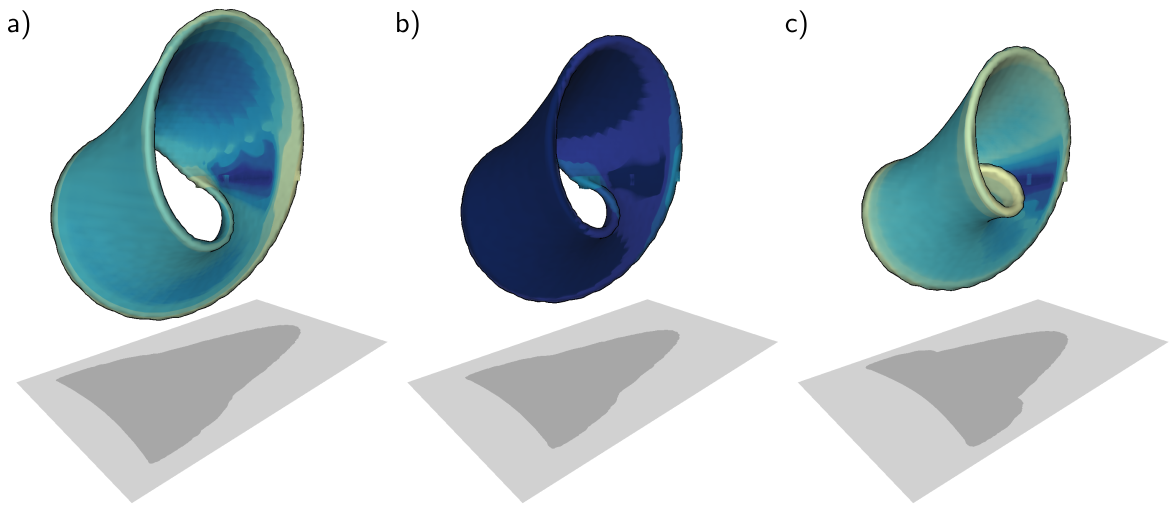


Figure 7.4: From an ensemble of 50 slightly perturbed Möbius bands, the median surface (middle) is computed using our method. The left and right surfaces show the two ensemble member with the highest overall deviation from the median. Color encodes centrality, ranging from dark blue (high) to yellow (low).

one hemisphere, all members have exactly the same geometry, while a high variability in the shapes can be observed in the frontal region. Secondly, the median shape is shown, color coded with local centrality. In the last two images, two members exhibiting low centrality are shown, which is indicated by the color shift towards yellow.

In Figure 7.6 (left), we show the computed median surface for one of the 3D weather forecast ensembles. Right, for the same ensemble, the locally best matching median surface is visualized, by using color to indicate, which of the ensemble members provides the best median surface locally. It can be seen that the surface in the bottom image shows details not present in the median shape, i.e., meaning that locally the median is not always the best representative. The same experiment is shown in the top image in Figure 7.7a, and in Figure 7.7b for the fluid ensemble. The visualization of the ECMWF data set reveals certain features that become immediately apparent to the user. For instance, one can discover the major regions where higher wind velocities are predicted, namely above the Baltic Sea and to the south of Greenland. Note that by analyzing other iso-values, we can verify that higher velocities occur within these regions, although this is not shown here. By studying the fuzzy structures, we can spot a region located northwest of Africa exhibiting a lower degree of centrality. This indicates the existence of outliers at that region. In the fluid ensemble, three representative shapes extracted from disjunct subsets of members are shown on the top. The visualization of a piecewise median is depicted at the bottom. By volume rendering the fuzzy structure, the user gets an overview of the distribution regarding the remaining ensemble members.

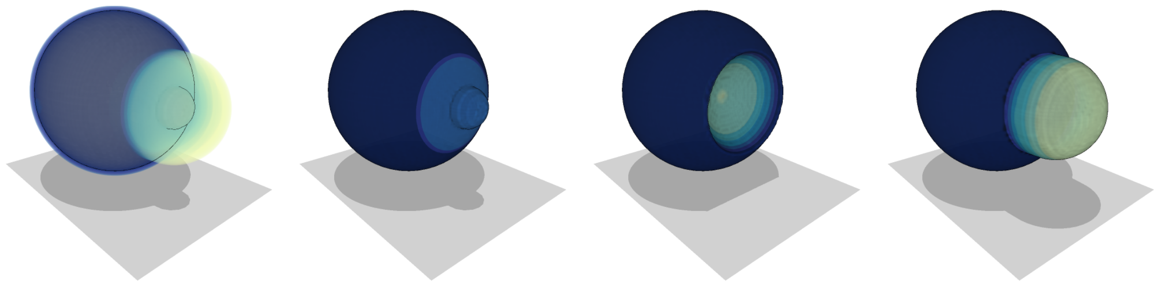


Figure 7.5: Visualization of an ensemble of locally perturbed 3D spheres. From left to right: The fuzzy volume indicating local variability of the ensemble. The median surface; color encodes centrality, ranging from dark blue (high) to yellow (low). Two members of the ensemble being less central than the median.

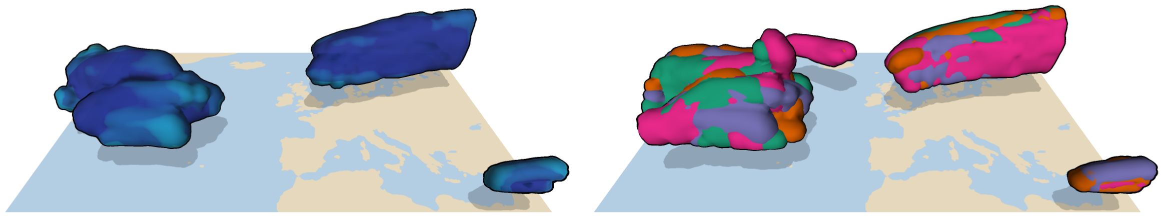


Figure 7.6: Visualization of an ensemble of 3D iso-surfaces in wind velocity fields. Left: The median surface. Color encodes centrality, ranging from dark blue (high) to yellow (low). Right: The locally best matching medians. Colors indicate different ensemble members.

Figure 7.1abc and the bottom images in Figure 7.7a show 2D iso-contours in the ECMWF ensembles. In both figures, spaghetti plots of all considered contours are shown, and the computed median contours are highlighted in red. Here it is worth noting that our approach yields exactly the same median contour as the contour boxplots [WMK13] do. In addition, we show for selected regions the directional variability of vectors to closest points, which was used to classify the local centrality. It can be seen that the directional variability is a good indicator for classifying the domain locally. While in central regions we see a clear bimodality with opposing directions, the distributions are tending towards unimodality or uniformity in the more de-central regions.

7.9 Conclusion and Future Work

In this chapter, we have presented a novel approach for visualizing shape-based ensembles in two or three dimensions while taking into account the spatial uncertainty that such an

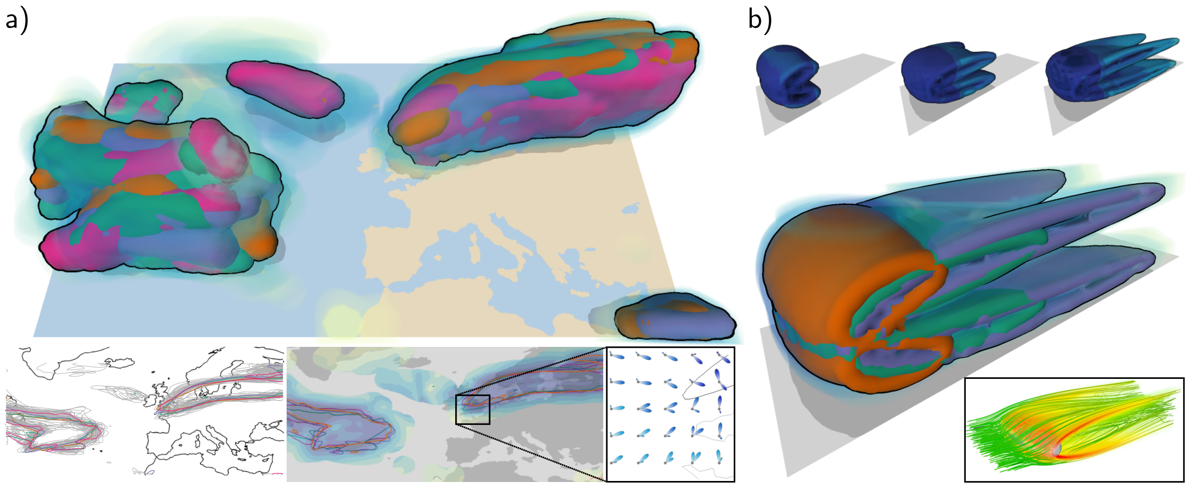


Figure 7.7: a) Top: 3D weather forecast ensemble. The coloring indicates the locally best matching median surface. Bottom: 2D slice from the same ensemble. First, a spaghetti plot of all iso-contours for a selected threshold is shown. The contours contributing to the median are highlighted. Next, the domain is colored according to local centrality, and the distribution of vectors-to-closest-points is shown for a selected region. b) Median and fuzzy regions are visualized in the same view for an ensemble featuring vorticity magnitude of a fluid simulation. On the bottom right a single member is rendered using streamlines.

ensemble entails. By combining the vector-to-closest-point representation with techniques from probability theory, we have proposed a statistical model for the resulting directional distributions. Based on this model, we classify regions and shapes with respect to central tendencies and uncertainty. We validated our technique by testing it with ensembles of 2D curves and comparing our results to those achieved by alternative approaches. In doing so, we discovered a high degree of similarity. An outstanding benefit of our approach lies in the fact that no assumptions about the input shapes are necessary and that our method can be extended to three dimensions in a straightforward way. By exploiting a GPU-accelerated ray-casting technique based on closest point grids, we provide the user with a tool for visually guided analysis of multi-dimensional shape-based ensembles at interactive rates.

Our approach opens up different aspects that can be investigated in the future: Firstly, analyzing and comparing the shape medians that are computed by our and other approaches, in particular with respect to local representativeness, can provide deeper insight into the effectiveness of our technique and give rise to future improvements. According to our observations, computing medians is a process that is susceptible to changes with respect to the VCPE grid resolution and the weighting of variability values in the mean square integration over the shapes. This issue can be addressed by pursuing investigations, where ground truth medians are available. Lastly, a more extensive analysis of the locally best representative median, in the context of the application, where it is used, is paramount. This special shape is assembled from sections of many different shapes, and it might be worthwhile to examine, to which degree the resulting shape can communicate application-specific features with regard to local tendencies to the domain expert.

8 Conclusion and Future Work

"Knowledge is an unending adventure at the edge of uncertainty."

– Jacob Bronowski

In this thesis, we have presented several visualization techniques for uncertain multidimensional data. Our approaches are predicated on showcasing visual abstractions that are derived from given data sets: By hiding expendable information, our goal is to allow the domain expert to focus the analysis on relevant structures. Abstracting from raw data in such a manner and developing visualization schemes that communicate these results to the user is a challenging task, especially, if uncertainty has to be considered. Due to the inherent nature of uncertainty in all real-word scenarios, it is an important research topic that concerns many disciplines in science and engineering. We have provided substantial contributions to this field of research. To conclude this thesis, we give a short summary of the methods that we have proposed, followed by an outlook over possible future research directions inspired by our work.

For visualizing multidimensional data, we have proposed a novel approach that is based on the Kriging interpolation scheme. As Kriging can handle scattered sample positions, generates smooth surfaces, and induces a measurement of uncertainty, our method is suitable for manifold real-world data sets. To visualize multidimensional data, we provide the user with multiple views, each consisting of two dimensions represented by a surface. To this end, we have implemented two strategies: Projecting the sample onto each surface, where relatively few sample points suffice to obtain a general overview; HyperSlice, where interpolation is carried out in multidimensional space and then cutting planes are inserted at a user-selected focus point. Our novel progressive GPU implementation allows the user to see results immediately, while the sample can be arbitrary large.

We have developed different techniques for visualizing ensemble data: By bidirectional linking between volumetric visualization and an abstract view, we enable the user to analyze 3D scalar field ensembles at the data level. In the abstract view, we draw multi-charts, which can be understood as an extension of bar and line charts that linearize 3D points along a space-filling curve. Statistical information, such as histograms, is encoded in the bars, where individual members are represented by overlaid line charts. As the abstract view does not suffer from occlusion effects, distributions and spatial dependencies are effectively revealed to the expert. Brushing and linking functionality in combination with volume visualization enables the user to explore the whole ensemble space.

Isosurfaces are commonly used for visualizing 3D scalar fields. We have extended this method to ensembles of scalar fields, built upon the essential idea of rendering silhouettes instead of solid surfaces. In this way, occlusion effects are minimized and the spatial coherence is still preserved. For an in-depth analysis, we provide several means to inspect the data set, such as cutting planes that can freely be placed by the user, picking and brushing, clustering and animation. We have demonstrated that our method is capable of revealing relevant features and spatial dependencies between different members.

Moreover, we have proposed a novel rendering technique based on a Vector-to-Closest-Point (VCP) representation. By using a regular grid, organized in an octree-like fashion, we achieve a fast and memory efficient ray-casting implementation that takes full advantage of the GPU's capabilities. We avoid rendering block artifacts that are inherent to classical voxel-based methods by smoothly interpolating between VCPs. To transform polygonal meshes into VCP grids, we have developed an algorithm capable of processing even the largest meshes, we had on hand, in a computationally efficient manner. We have demonstrated this for the David statue, a mesh consisting of 960 million triangles. By generating the VCP structure at different resolutions, and dynamically loading only subsets of the finest level into GPU memory, we circumvent memory restrictions and still maintain very fine details from the original data.

Although the applications of this approach are not restricted to the area of visualization, we have developed a visualization technique for shape-based ensembles based upon VCP representations. By combining the VCP representation with statistical models for directional distributions, we are able to classify regions and shapes according to centrality and uncertainty. Due to the underlying VCP structure, our approach does not make any assumptions about the given shapes, unlike similar alternatives that can be found in the literature, e.g., shapes do not need to be closed or orientable. We have implemented our solution for two and three-dimensional shapes, and by integrating our GPU-based ray-casting method, we have provided an interactive tool for visual exploration of arbitrary shapes.

We have demonstrated the effectiveness of all our approaches with a couple of real-world data sets. As we have shown, various features were revealed by using our visualization schemes. In the future, our methods might have the potential of providing insight into data sets from various domains of application. Moreover, there are several future research directions based on our work.

To discover shape-based feature indicators in higher dimensional data, our approach of rendering progressive response surfaces could be analyzed with functions that have known extreme points. Knowledge about topological features discovered in this way might then be applied to real-world data sets. In combination with multi-charts, our approaches could be combined and extended to visualizing higher dimensional data, while still maintaining the spatial coherence to a good degree. This is due to the fact that space-filling curves can be extended to higher dimensions in a straightforward way. Likewise, an extension to time-varying data might turn out to yield useful results, either by interpreting time as another multi-chart dimension, or by utilizing animation. Time-varying data might also be of interest in the context of our silhouette-based isosurface rendering technique. Here, animation is already used to iterate over individual members and it might prove as equally effective for iterating over different time steps.

Although not being a visualization technique by itself, our VCP ray-casting approach can serve as a basis for advanced visualization methods that rely on high resolutions or incorporate complex transparency effects. In addition, integrating secondary rays might support the user in better understanding spatial context, for instance, by casting shadows. As mentioned before, we have also developed a visualization technique based on VCP grids. Here, it might be worthwhile to compare the medians found by using the VCP representation to medians that are computed by other approaches. Moreover, the concept behind the locally best matching median has to be validated more thoroughly, in particular in the context of application-specific information concerning local trends.

Bibliography

[AL09] Aila T., Laine S.: Understanding the efficiency of ray traversal on gpus. In *Proc. High-Performance Graphics 2009* (2009).

[All10] Allen T.: *Introduction to engineering statistics and lean sigma : statistical quality control and design of experiments and systems.* Springer, London New York, 2010.

[AMT*12] Auer S., Macdonald C. B., Treib M., Schneider J., Westermann R.: Real-time fluid effects on surfaces using the closest point method. *Computer Graphics Forum 31*, 6 (2012).

[AOB08] Allendes Osorio R., Brodlie K. W.: Contouring with uncertainty. *Theory and Practice of Computer Graphics 2008. Proceedings.* (2008), 59–66.

[ASE16] Athawale T., Sakhaee E., Entezari A.: Isosurface visualization of data with nonparametric models for uncertainty. *IEEE transactions on visualization and computer graphics 22*, 1 (2016), 777–786.

[Asi85] Asimov D.: The grand tour: a tool for viewing multidimensional data. *SIAM J. Sci. Stat. Comput. 6*, 1 (Jan. 1985), 128–143.

[Aut93] Auto mpg data set. `http://archive.ics.uci.edu/ml/datasets/Auto+MPG`, 1993. UCI Machine Learning Repository.

[AWH*12] Alabi O. S., Wu X., Harter J. M., Phadke M., Pinto L., Petersen H., Bass S., Keifer M., Zhong S., Healey C., Taylor II R. M.: Comparative visualization of ensembles using ensemble surface slicing. In *IS&T/SPIE*

Electronic Imaging (2012), International Society for Optics and Photonics, pp. 82940U–82940U.

[Ban37] BANACHIEWICZ T.: Zur Berechnung der Determinanten, wie auch der Inversen, und zur darauf basierten Auflösung der Systeme linearer Gleichungen. *Acta Astronomica, Serie C*, 3 (1937), 41–67.

[BBF*11] BUSKING S., BOTHA C., FERRARINI L., MILLES J., POST F.: Image-based rendering of intersecting surfaces for dynamic comparative visualization. *The Visual Computer 27* (2011), 347–363.

[BC87] BECKER R. A., CLEVELAND W. S.: Brushing scatterplots. *Technometrics 29*, 2 (1987), 127–142.

[BC08] BASTOS T., CELES W.: Gpu-accelerated adaptively sampled distance fields. In *Shape Modeling and Applications, 2008. SMI 2008. IEEE International Conference on* (June 2008), pp. 171–178.

[BDGS05] BANERJEE A., DHILLON I. S., GHOSH J., SRA S.: Clustering on the unit hypersphere using von mises-fisher distributions. *J. Mach. Learn. Res. 6* (Dec. 2005), 1345–1382.

[BHGK14] BEHAM M., HERZNER W., GRÖLLER M. E., KEHRER J.: Cupid: Cluster-based exploration of geometry generators with parallel coordinates and radial trees. *IEEE TVCG 20*, 12 (2014), 1693–1702.

[BHJ*14] BONNEAU G.-P., HEGE H.-C., JOHNSON C. R., OLIVEIRA M. M., POTTER K., RHEINGANS P., SCHULTZ T.: Overview and state-of-the-art of uncertainty visualization. In *Scientific Visualization*. Springer, 2014, pp. 3–27.

[Bij73] BIJNEN E. J.: *Cluster analysis – Survey and evaluation of techniques*. Springer Netherlands, Dordrecht, 1973.

[Bil98] BILMES J.: *A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models*. Tech. rep., 1998.

[BK05] BOTSCH M., KOBBELT L.: Real-time shape editing using radial basis functions. In *Computer Graphics Forum* (2005), pp. 611–621.

[BKS04] BORDOLOI U. D., KAO D. L., SHEN H.-W.: Visualization techniques for spatial probability density function data. *Data Science Journal 3* (2004), 153–162.

[Blo50] Blomqvist N.: On a measure of dependence between two random variables. *The Annals of Mathematical Statistics 21*, 4 (1950), 593–600.

[BM10a] Bruckner S., Möller T.: Isosurface similarity maps. *Computer Graphics Forum 29*, 3 (2010), 773–782.

[BM10b] Bruckner S., Möller T.: Result-driven exploration of simulation parameter spaces for visual effects design. *IEEE TVCG 16*, 6 (2010), 1468–1476.

[BOL12] Brodlie K., Osorio R. A., Lopes A.: A review of uncertainty in data visualization. In *Expanding the frontiers of visual analytics and visualization.* Springer, 2012, pp. 81–109.

[BPFG11] Berger W., Piringer H., Filzmoser P., Gröller E.: Uncertainty-aware exploration of continuous parameter spaces using multivariate prediction. *Published in Computer Graphics Forum 30*, 3 (2011), pp. 911 – 920.

[Bro04] Brown R.: Animated visual vibrations as an uncertainty visualisation technique. In *Proc. GRAPHITE* (2004), pp. 84–89.

[Buh03] Buhmann M.: *Radial basis functions theory and implementations.* Cambridge University Press, Cambridge New York, 2003.

[BWE05] Botchen R. P., Weiskopf D., Ertl T.: Texture-based visualization of uncertainty in flow fields. In *VIS 05. IEEE Visualization, 2005.* (2005), IEEE, pp. 647–654.

[Cas02] Casella G.: *Statistical Inference.* Thomson Learning, Australia Pacific Grove, CA, 2002.

[CBC*01] Carr J. C., Beatson R. K., Cherrie J. B., Mitchell T. J., Fright W. R., McCallum B. C., Evans T. R.: Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), SIGGRAPH '01, ACM, pp. 67–76.

[CBDT11] Coninx A., Bonneau G.-P., Droulez J., Thibault G.: Visualization of uncertain scalar data fields using color scales and perceptually adapted noise. In *Proceedings of the ACM SIGGRAPH Symposium on Applied Perception in Graphics and Visualization* (2011), ACM, pp. 59–66.

[CGL83] Chan T. F., Golub G. H., LeVeque R. J.: Algorithms for computing the sample variance: Analysis and recommendations. *The American Statistician 37*, 3 (1983), 242–247.

[Cha83] Chatfield C.: *Statistics for technology : a course in applied statistics.* Chapman and Hall, London New York, 1983.

[CHCH06] Carr N. A., Hoberock J., Crane K., Hart J. C.: Fast GPU ray tracing of dynamic meshes using geometry images. In *Proc. Graphics Interface* (2006), pp. 203–209.

[CHH02] Carr N. A., Hall J. D., Hart J. C.: The ray engine. In *HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (2002), pp. 37–46.

[CJ08] Cressie N., Johannesson G.: Fixed rank kriging for very large spatial data sets. *Journal of the Royal Statistical Society: Statistical Methodology 70*, 1 (2008), 209–226.

[Cle85] Cleveland W. S.: *The elements of graphing data.* Wadsworth Publ. Co., Belmont, CA, USA, 1985.

[CNLE09] Crassin C., Neyret F., Lefebvre S., Eisemann E.: Gigavoxels : Ray-guided streaming for efficient and detailed voxel rendering, feb 2009. to appear.

[Cre93] Cressie N.: *Statistics for Spatial Data.* Wiley Series in Probability and Statistics. Wiley-Interscience, Jan. 1993.

[CSD*09] Cole F., Sanik K., Decarlo D., Finkelstein A., Funkhouser T., Rusinkiewicz S., Singh M.: How well do line drawings depict shape. *ACM Trans. on Graph. (Proc. of SIGGRAPH)* (2009).

[dBCvKO08] de Berg M., Cheong O., van Kreveld M., Overmars M.: *Delaunay Triangulations*, 3rd ed. ed. Springer-Verlag TELOS, Santa Clara, CA, USA, 2008, pp. 191–218.

[DDW14] Demir I., Dick C., Westermann R.: Multi-Charts for Comparative 3D Ensemble Visualization. *IEEE Transactions on Visualization and Computer Graphics 20*, 12 (Dec. 2014). `doi:10.1109/TVCG.2014.2346448`.

[Dey10] Dey D.: *Essential Bayesian models : a derivative of Handbook of statistics: Bayesian thinking - modeling and computation, vol. 25.* North-Holland, Amsterdam, 2010.

[DGBW08] Dick C., Georgii J., Burgkart R., Westermann R.: Computational steering for patient-specific implant planning in orthopedics. In *Proceedings of the First Eurographics conference on Visual Computing for Biomedicine* (2008), pp. 83–92.

[DH02] Doleisch H., Hauser H.: Smooth brushing for focus+context visualization of simulation data in 3D. *Journal of WSCG 10*, 1–3 (2002), 147–154.

[DJW16] Demir I., Jarema M., Westermann R.: Visualizing the central tendency of ensembles of shapes. In *SIGGRAPH Asia 2016 Symposium on Visualization* (New York, NY, USA, 2016), SA '16, ACM. `doi:10.1145/3002151.3002165`.

[DKD09] Der Kiureghian A., Ditlevsen O.: Aleatory or epistemic? Does it matter? *Structural Safety 31*, 2 (2009), 105–112.

[DKLP01] Djurcilov S., Kim K., Lermusiaux P. F., Pang A.: Volume rendering data with uncertainty information. In *Data Visualization 2001*. Springer, 2001, pp. 243–252.

[DKLP02] Djurcilov S., Kim K., Lermusiaux P., Pang A.: Visualizing scalar volumetric data with uncertainty. *Computers & Graphics 26*, 2 (2002), 239–248.

[DKW16] Demir I., Kehrer J., Westermann R.: Screen-space Silhouettes for Visualizing Ensembles of 3D Isosurfaces. In *Proc. IEEE Pacific Visualization Symp. (Visualization Notes)* (2016). `doi:10.1109/PACIFICVIS.2016.7465271`.

[Dol07] Doleisch H.: SimVis: Interactive visual analysis of large and time-dependent 3D simulation data. In *Proc. Winter Simulation Conference* (2007), pp. 712–720.

[DW13] Demir I., Westermann R.: Progressive High-Quality Response Surfaces for Visually Guided Sensitivity Analysis. *Computer Graphics Forum (Proceedings of EuroVis 2013) 32*, 3 (2013), 21–30. `doi:10.1111/cgf.12089`.

[DW15] Demir I., Westermann R.: Vector-to-Closest-Point Octree for Surface Ray-Casting. In *Vision, Modeling & Visualization* (2015), Bommes D., Ritschel T., Schultz T., (Eds.), The Eurographics Association. `doi:10.2312/vmv.20151259`.

[EB03] Esty W. W., Banfield J. D.: The box-percentile plot. *Journal of Statistical Software 8*, 17 (2003), 1–14.

[EQOR11] Ezzatti P., Quintana-Ortí E., Remon A.: High performance matrix inversion on a multi-core platform with several gpus. In *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on* (2011), pp. 87 –93.

[EVG04] Ernst M., Vogelgsang C., Greiner G.: Stack implementation on programmable graphics hardware. In *Vision Modeling and Visualization* (2004), pp. 255–262.

[FB94] Furnas G. W., Buja A.: Prosection views: Dimensional inference through sections and projections. *Journal of Computational and Graphical Statistics 3* (1994), 323–385.

[FBW16] Ferstl F., Bürger K., Westermann R.: Streamline variability plots for characterizing the uncertainty in vector field ensembles. *IEEE Transactions on Visualization and Computer Graphics 22*, 1 (Jan 2016), 767–776.

[Fis95] Fisher N.: *Statistical Analysis of Circular Data.* Cambridge Univ. Press, 1995.

[FKLT10] Feng D., Kwock L., Lee Y., Taylor R.: Matching visual saliency to confidence in plots of uncertain data. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 980–989.

[FKRW16] Ferstl F., Kanzler M., Rautenhaus M., Westermann R.: Visual analysis of spatial variability and global correlations in ensembles of iso-contours. In *Computer Graphics Forum (Proc. EuroVis)* (2016). to appear.

[FLE87] Fisher N., Lewis T., Embleton B.: *Statistical analysis of spherical data.* Cambridge Univ. Press, 1987.

[FML16] Fofonov A., Molchanov V., Linsen L.: Visual analysis of multi-run spatio-temporal simulations using isocontour similarity for projected views. *IEEE Trans. Visual. and Comp. Graphics, to appear* (2016).

[FN91] Franke R., Nielson G. M.: Scattered data interpolation and applications: A tutorial and survey. In *Geometric Modelling: Methods and Their Applications*, Hagen H., Roller D., (Eds.). Springer, 1991, pp. 131–160.

[FPRJ00] Frisken S. F., Perry R. N., Rockwood A. P., Jones T. R.: Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (2000), SIGGRAPH '00, pp. 249–254.

[Fri97] Fristedt B.: *A Modern Approach to Probability Theory.* Birkhäuser, Boston, 1997.

[FS05] Foley T., Sugerman J.: Kd-tree acceleration structures for a GPU raytracer. In *HWWS '05: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (2005), pp. 15–22.

[FT74] Friedman J. H., Tukey J. W.: A Projection Pursuit Algorithm for Exploratory Data Analysis. *IEEE Trans. Comput. 23*, 9 (1974), 881–890.

[GAW*11] Gleicher M., Albers D., Walker R., Jusufi I., Hansen C. D., Roberts J. C.: Visual comparison for information visualization. *Information Visualization 10*, 4 (2011), 289–309.

[GBS*99] Goel A., Baker C., Shaffer C. A., Grossman B., Haftka R. T., Mason W. H., Watson L. T.: VizCraft: A multidimensional visualization tool for aircraft configuration design. In *Proc. IEEE Visualization* (1999), pp. 425–428.

[GC11] Gupta M. R., Chen Y.: *Theory and use of the EM algorithm.* Now Publishers Inc, 2011.

[Ger92] Gershon N. D.: Visualization of fuzzy data using generalized animation. In *Visualization, 1992. Visualization'92, Proceedings., IEEE Conference on* (1992), IEEE, pp. 268–273.

[GH97] Garland M., Heckbert P. S.: Surface simplification using quadric error metrics. In *Proc. SIGGRAPH '97* (1997), pp. 209–216.

[Gib02] Gibbs J.: *Elementary Principles in Statistical Mechanics: Developed with Especial Reference to the Rational Foundations of Thermodynamics.* C. Scribner's sons, 1902.

[Gib98] Gibson S. F. F.: Using distance maps for accurate surface representation in sampled volumes. In *Proceedings of the 1998 IEEE Symposium on Volume Visualization* (1998), VVS '98, pp. 23–30.

[GLLL09] Ge Y., Li S., Lakhan V. C., Lucieer A.: Exploring uncertainty in remotely sensed data with parallel coordinate plots. *International Journal of Applied Earth Observation and Geoinformation 11*, 6 (2009), 413–422.

[GM05] Gobbetti E., Marton F.: Far voxels: a multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms. *ACM Transactions on Graphics 24*, 3 (2005), 878–885.

[GMIG08] Gobbetti E., Marton F., Iglesias Guitián J. A.: A single-pass gpu ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *Vis. Comput. 24*, 7 (July 2008), 797–806.

[GPSS07] Günther J., Popov S., Seidel H.-P., Slusallek P.: Realtime ray tracing on GPU with BVH-based packet traversal. In *Proceedings of the IEEE/Eurographics Symposium on Interactive Ray Tracing 2007* (Sept. 2007), pp. 113–118.

[GR02] Grigoryan G., Rheingans P.: Probabilistic surfaces: Point based primitives to show surface uncertainty. In *Proceedings of the conference on Visualization'02* (2002), IEEE Computer Society, pp. 147–154.

[GR04] Grigoryan G., Rheingans P.: Point-based probabilistic surfaces to show surface uncertainty. *IEEE TVCG 10*, 5 (2004), 564–573.

[GRT13] Günther T., Rössl C., Theisel H.: Opacity optimization for 3d line fields. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 120.

[GRW*00] Gresh D. L., Rogowitz B. E., Winslow R. L., Scollan D. F., Yung C. K.: WEAVE: A system for visually linking 3-D and statistical visualizations, applied to cardiac simulation and measurement data. In *Proc. IEEE Visualization* (2000), pp. 489–492.

[GS06] Griethe H., Schumann H.: The visualization of uncertain data: Methods and problems. In *Proc. SimVis* (2006), pp. 143–156.

[GTC01] Grinstein G., Trutschl M., Cvek U.: High-dimensional visualizations. In *Workshop on Visual Data Mining* (2001), 7th Conf. on Knowledge Discovery and Data Mining (KDD), pp. 77–87.

[GXY12] Guo H., Xiao H., Yuan X.: Scalable multivariate volume visualization and analysis based on dimension projection and parallel coordinates. *IEEE TVCG 18*, 9 (2012), 1397–1410.

[Haa95] Haas T. C.: Local prediction of a spatio temporal process with an application to wet sulfate deposition. *American statistical Association 90*, 432 (1995), 1189–1199.

[Han05] Hansen C.: *The visualization handbook.* Elsevier Butterworth-Heinemann, Burlington, MA, 2005.

[Har96] Hart J. C.: Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer 12* (1996), 527–545.

[Har07] Harris M.: Optimizing parallel reduction in cuda, 2007. NVIDIA Developer Technology.

[Hau05] Hauser H.: Generalizing focus+context visualization. In *Scientific Visualization: The Visual Extraction of Knowledge from Data*. Springer, 2005, pp. 305–327.

[HBG*12] Heinzl C., Bruckner S., Gröller M. E., Pang A., Hege H.-C., Potter K., Westermann R., Pfaffelmoser T., Möller T.: Uncertainty and parameter space analysis in visualization. IEEE VisWeek Tutorial, 2012.

[HCL*11] Huang L., Chen K., Lai Y., Chang P., Song S.: Geological visualization system with gpu-based interpolation. *AGU Fall Meeting Abstracts* (Dec. 2011), B1600.

[Hei05] Heidrich W.: Computing the barycentric coordinates of a projected point. *Journal of Graphics, GPU, and Game Tools 10*, 3 (2005), 9–12.

[Hen03] Hengl T.: Visualisation of uncertainty using the hsi colour model: computations with colours. In *7th International Conference on GeoComputation* (2003).

[HHB16] Hao L., Healey C., Bass S.: Effective visualization of temporal ensembles. *IEEE TVCG 22*, 1 (Jan 2016), 787–796.

[HJ61] Hooke R., Jeeves T. A.: "Direct Search" Solution of Numerical and Statistical Problems. *Journal of the ACM (JACM) 8*, 2 (1961), 212–229.

[HLD02] Hauser H., Ledermann F., Doleisch H.: Angular brushing of extended parallel coordinates. In *Proc. IEEE Symposium on Information Visualization* (2002), pp. 127–137.

[HM90] Haber R. B., Mcnabb D. A.: Visualization Idioms: A Conceptual Model for Scientific Visualization Systems. In *Visualization in Scientific Computing*. 1990.

[HMC*13] Höllt T., Magdy A., Chen G., Gopalakrishnan G., Hoteit I., Hansen C., Hadwiger M.: Visual analysis of uncertainties in ocean forecasts for planning and operation of off-shore structures. In *Proc. IEEE Pacific Visualization Symposium* (2013), pp. 185–192.

[HMZ*14] Höllt T., Magdy A., Zhan P., Chen G., Gopalakrishnan G., Hoteit I., Hansen C. D., Hadwiger M.: Ovis: A framework for visual analysis of ocean forecast ensembles. *IEEE TVCG PP*, 99 (2014).

[HN98] Hintze J. L., Nelson R. D.: Violin plots: A box plot-density trace synergism. *The American Statistician 52*, 2 (1998), 181–184.

[HN12] Heitz E., Neyret F.: Representing appearance and pre-filtering subpixel data in sparse voxel octrees. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics* (2012), EGGH-HPG'12, pp. 125–134.

[HRC08] Hamilton C. H., Rau-Chaplin A.: Compact Hilbert indices: Space-filling curves for domains with unequal side lengths. *Information Processing Letters 105*, 5 (2008), 155–163.

[HSHH07] Horn D. R., Sugerman J., Houston M., Hanrahan P.: Interactive k-d tree GPU raytracing. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games* (2007), pp. 167–174.

[Ins85] Inselberg A.: The plane with parallel coordinates. *The Visual Computer 1*, 2 (Aug. 1985), 69–91.

[Jai10] Jain A. K.: Data clustering: 50 years beyond k-means. *Pattern recognition letters 31*, 8 (2010), 651–666.

[JBS06] Jones M. W., Baerentzen J. A., Sramek M.: 3d distance fields: A survey of techniques and applications. *IEEE Transactions on Visualization and Computer Graphics 12*, 4 (July 2006), 581–599.

[JDKW15] Jarema M., Demir I., Kehrer J., Westermann R.: Comparative Visual Analysis of Vector Field Ensembles. In *Visual Analytics Science and Technology (VAST), 2015 IEEE Conference on* (Oct 2015), pp. 81–88. doi:10.1109/VAST.2015.7347634.

[JKLE09] Jasche J., Kitaura F. S., Li C., Ensslin T. A.: Mapping of the universe beyond the known. http://www.mpa-garching.mpg.de/mpa/research/current_research/hl2009-12/hl2009-12-en.html, 2009.

[JKW16] Jarema M., Kehrer J., Westermann R.: Comparative visual analysis of transport variability in flow ensembles. *Journal of WSCG 24*, 1 (2016), 25–34.

[JLRP99] Jospeh A. J., Lodha S. K., Renteria J. C., Pang A.: Uisurf: Visualizing uncertainty in isosurfaces. In *Proceedings of the Computer Graphics and Imaging* (1999), pp. 184–191.

[JPGJ12] Jiao F., Phillips J. M., Gur Y., Johnson C. R.: Uncertainty visualization in hardi based on ensembles of odfs. In *Visualization Symposium (PacificVis), 2012 IEEE Pacific* (2012), IEEE, pp. 193–200.

[JPS*10] Jiao F., Phillips J. M., Stinstra J., Krger J., Varma R., Hsu E., Korenberg J., Johnson C. R.: Metrics for uncertainty analysis and visualization of diffusion tensor images. In *International Workshop on Medical Imaging and Virtual Reality* (2010), Springer, pp. 179–190.

[JS03] Johnson C. R., Sanderson A. R.: A next step: Visualizing errors and uncertainty. *IEEE Computer Graphics and Applications 23*, 5 (2003), 6–10.

[KBH04] Kosara R., Bendix F., Hauser H.: TimeHistograms for large, time-dependent data. In *Proc. Joint Eurographics - IEEE TCVG Symposium on Visualization* (2004), pp. 45–54.

[Kei02] Keim D. A.: Information visualization and visual data mining. *IEEE TVCG 8*, 1 (2002), 1–8.

[KFH10] Kehrer J., Filzmoser P., Hauser H.: Brushing moments in interactive visual analysis. *Computer Graphics Forum 29*, 3 (2010), 813–822.

[KH10] Kirk D., Hwu W.-m.: *Programming massively parallel processors : a hands-on approach.* Morgan Kaufmann Publishers, Burlington, MA, 2010.

[KH13] Kehrer J., Hauser H.: Visualization and visual analysis of multi-faceted scientific data: A survey. *IEEE TVCG 19*, 3 (2013), 495–513.

[KHDH02] Keim D. A., Hao M. C., Dayal U., Hsu M.: Pixel bar charts: a visualization technique for very large multi-attribute data sets. *Information Visualization 1*, 1 (2002), 20–34.

[KHDL07] Keim D. A., Hao M. C., Dayal U., Lyons M.: Value-cell bar charts for visualizing large transaction data sets. *IEEE TVCG 13*, 4 (2007), 822–833.

[KLM*08] Kehrer J., Ladstädter F., Muigg P., Doleisch H., Steiner A., Hauser H.: Hypothesis generation in climate research with interactive visual data exploration. *IEEE TVCG 14*, 6 (2008), 1579–1586.

[KMDH11] Kehrer J., Muigg P., Doleisch H., Hauser H.: Interactive visual analysis of heterogeneous scientific data across an interface. *IEEE TVCG 17*, 7 (2011), 934–946.

[KMN98] Kearns M., Mansour Y., Ng A. Y.: An information-theoretic analysis of hard and soft assignment methods for clustering. In *Learning in graphical models*. Springer, 1998, pp. 495–520.

[KO01] Kennedy M. C., O'Hagan A.: Bayesian calibration of computer models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology) 63*, 3 (2001), 425–464.

[KPBG13] Kehrer J., Piringer H., Berger W., Gröller M. E.: A model for structure-based comparison of many categories in small-multiple displays. *IEEE Transactions on Visualization and Computer Graphics 19*, 12 (Dec. 2013), 2287–2296.

[Kri51] Krige D. G.: A statistical approach to some basic mine valuation problems on the witwatersrand. *Journal of the Chemical, Metallurgical and Mining Society of South Africa 52*, 6 (Dec. 1951), 119–139.

[KSA13] Kämpe V., Sintorn E., Assarsson U.: High resolution sparse voxel dags. *ACM Trans. Graph. 32*, 4 (July 2013), 101:1–101:13.

[KSDD14] Kothur P., Sips M., Dobslaw H., Dransch D.: Visual analytics for comparison of ocean model output with reference data: Detecting and analyzing geophysical processes using clustering ensembles. *IEEE TVCG 20*, 12 (2014), 1893–1902.

[KSK*14] Keinert B., Schäfer H., Korndörfer J., Ganse U., Stamminger M.: Enhanced Sphere Tracing. pp. 1–8.

[KVUS*05] Kniss J. M., Van Uitert R., Stephens A., Li G.-S., Tasdizen T., Hansen C.: Statistically quantitative volume visualization. In *VIS 05. IEEE Visualization, 2005.* (2005), IEEE, pp. 287–294.

[KWTM03] Kindlmann G., Whitaker R., Tasdizen T., Moller T.: Curvature-based transfer functions for direct volume rendering: Methods and applications. In *Proc. IEEE Visualization* (2003), pp. 513–520.

[LC87] Lorensen W. E., Cline H. E.: Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Comput. Graph. 21*, 4 (Aug. 1987), 163–169. URL: `http://doi.acm.org/10.1145/37402.37422`, `doi:10.1145/37402.37422`.

[Lev97] Levkowitz H.: *Color theory and modeling for computer graphics, visualization, and multimedia applications.* Kluwer Academic Publishers, Boston, 1997.

[LGF00] Leventon M., Grimson W., Faugeras O.: Statistical shape influence in geodesic active contours. In *Proc. IEEE Computer Vision and Pattern Recognition* (2000), pp. 316–323.

[LHKK79] Lawson C. L., Hanson R. J., Kincaid D. R., Krogh F. T.: Basic linear algebra subprograms for fortran usage. *ACM Trans. Math. Softw. 5*, 3 (Sept. 1979), 308–323. `doi:10.1145/355841.355847`.

[LK10a] Laine S., Karras T.: Efficient sparse voxel octrees. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2010), I3D '10, pp. 55–63.

[LK10b] Laine S., Karras T.: *Efficient Sparse Voxel Octrees – Analysis, Extensions, and Implementation.* NVIDIA Technical Report NVR-2010-001, NVIDIA Corporation, Feb. 2010.

[Llo82] Lloyd S. P.: Least squares quantization in PCM. *IEEE Transactions on Information Theory 28*, 2 (1982), 129–137.

[LLPY07] Lundström C., Ljung P., Persson A., Ynnerman A.: Uncertainty visualization in medical volume rendering using probabilistic animation. *IEEE transactions on visualization and computer graphics 13*, 6 (2007), 1648–1655.

[LMK*15] Liu L., Mirzangar M., Kirby R. M., Whitaker R., House D. H.: Visualizing time-specific hurricane predictions, with uncertainty, from storm path ensembles. *Comput. Graph. Forum 34*, 3 (2015), 371–380.

[LP08] Leutbecher M., Palmer T.: Ensemble forecasting. *Journal of Computational Physics 227*, 7 (2008), 3515–3539.

[Mat63] Matheron G.: Principles of geostatistics. *Economic Geology 58*, 8 (1963), 1246–1266.

[MB62] Matheron G., Blondel F.: Traité de géostatistique appliquée, Tome I. *Memoires du Bureau de Recherches Geologiques et Minieres 14* (1962).

[MCW*08] Mak W.-H., Chan M.-Y., Wu Y., Chung K.-K., Qu H.: VoxelBars: An informative interface for volume visualization. In *Advances in Visual Computing*, vol. 5358 of *Lecture Notes in Computer Science*. Springer, 2008, pp. 161–170.

[MGKH09] Matković K., Gračanin D., Klarin B., Hauser H.: Interactive visual analysis of complex scientific data as families of data surfaces. *IEEE TVCG 15*, 6 (2009), 1351–1358.

[Mic16] Microsoft Corporation: Direct3D (Windows). `https://msdn.microsoft.com/en-us/library/windows/desktop/hh309466(v=vs.85).aspx`, 2016. Accessed: 2016-08-30.

[MM95] Myers R. H., Montgomery D. C.: *Process Improvement with Steepest Ascent, The Analysis of Response Surfaces, Experimental Designs for Fitting Response Surfaces*, 1st ed. John Wiley & Sons, Inc., New York, NY, USA, 1995, pp. 183–351.

[Möl08] Möller T.: *Real-time rendering.* A.K. Peters, Wellesley, Mass, 2008.

[Mon06] Montgomery D. C.: *Response surface method and designs.* John Wiley & Sons, New Jersey, 2006.

[MR08] Macdonald C. B., Ruuth S. J.: Level set equations on surfaces via the Closest Point Method. *J. Sci. Comput. 35*, 2–3 (June 2008), 219–240.

[MRH*05] MacEachren A., Robinson A., Hopper S., Gardner S., Murray R., Gahegan M., Hetzler E.: Visualizing geospatial information uncertainty: What we know and what we need to know. *Cartography and Geographic Information Science 32*, 3 (2005), 139–161.

[MW14] Mihai M., Westermann R.: Visualizing the stability of critical points in uncertain scalar fields. *Computers & Graphics 41*, 0 (2014), 13 – 25. `doi:http://dx.doi.org/10.1016/j.cag.2014.01.007`.

[MWK14] Mirzargar M., Whitaker R. T., Kirby R. M.: Curve boxplot: Generalization of boxplot for ensembles of curves. *IEEE TVCG 20*, 12 (2014), 2654–2663.

[Nea99] Neal R. M.: Regression and classification using Gaussian process priors (with discussion). *Bayesian Statistics 6* (1999), 475–501.

[NVI10] NVIDIA Corporation: NVIDIA CUDA C programming guide. `http://docs.nvidia.com/cuda/cuda-c-programming-guide/`, 2010. Version 7.5. Accessed: 2016-09-01.

[OJ14] Obermaier H., Joy K.: Future challenges for ensemble visualization. *IEEE Comp. Graphics and Applications 34* (2014), 8–11.

[OK78] O'Hagan A., Kingman J. F. C.: Curve fitting and optimal design for prediction. *Journal of the Royal Statistical Society. Series B (Methodological) 40*, 1 (1978), 1–42.

[Opp98] Opper M.: On-line learning in neural networks. 1998, ch. A Bayesian approach to on-line learning, pp. 363–378.

[Par62] Parzen E.: On estimation of a probability density function and mode. *The Annals of Mathematical Statistics 33*, 3 (1962), 1065–1076.

[Par79] Parzen E.: Nonparametric statistical data modeling. *Journal of the American statistical association 74*, 365 (1979), 105–121.

[Pat96] Patel J.: *Handbook of the normal distribution.* Marcel Dekker, New York, 1996.

[PBK10] Piringer H., Berger W., Krasser J.: Hypermoval: Interactive visual validation of regression models for real-time simulation. *Computer Graphics Forum 29*, 3 (2010), 983–992.

[PBMH02] Purcell T. J., Buck I., Mark W. R., Hanrahan P.: Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics 21*, 3 (July 2002), 703–712.

[PGA13] Potter K., Gerber S., Anderson E. W.: Visualization of uncertainty without a mean. *IEEE computer graphics and applications 33*, 1 (2013), 75–79.

[PGSS07] Popov S., Günther J., Seidel H.-P., Slusallek P.: Stackless kd-tree traversal for high performance GPU ray tracing. *Computer Graphics Forum 26*, 3 (2007), 415–424.

[PH11] Pothkow 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.

[PKH04] Piringer H., Kosara R., Hauser H.: Interactive focus+context visualization with linked 2D/3D scatterplots. In *Proc. International Conference on Coordinated & Multiple Views in Exploratory Visualization* (2004), pp. 49–60.

[PKRJ10] Potter K., Kniss J., Riesenfeld R., Johnson C. R.: Visualizing summary statistics and uncertainty. *Computer Graphics Forum 29*, 3 (2010), 823–831.

[PRJ12] Potter K., Rosen P., Johnson C. R.: From quantification to visualization: A taxonomy of uncertainty visualization approaches. In *Uncertainty Quantification in Scientific Computing*. Springer Berlin Heidelberg, 2012, pp. 226–249.

[PRW11] Pfaffelmoser T., Reitinger M., Westermann R.: Visualizing the positional and geometrical variability of isosurfaces in uncertain scalar fields. In *Computer Graphics Forum* (2011), vol. 30, Wiley Online Library, pp. 951–960.

[PW12] Pfaffelmoser T., Westermann R.: Visualization of global correlation structures in uncertain 2D scalar fields. *Computer Graphics Forum 31*, 3 (2012), 1025–1034.

[PW13] Pfaffelmoser T., Westermann R.: Visualizing contour distributions in 2d ensemble data. In *EuroVis-Short Papers* (2013), The Eurographics Association, pp. 55–59.

[PWB*09a] 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 *2009 IEEE International Conference on Data Mining Workshops* (2009), IEEE, pp. 233–240.

[PWB*09b] Potter K., Wilson A., Bremer P.-T., Williams D., Pascucci V., Johnson C.: A flexible approach for the statistical visualization of ensemble data. In *Proc. IEEE ICDM Workshop Knowledge Discovery from Climate Data* (2009).

[PWH11] Pöthkow K., Weber B., Hege H.-C.: Probabilistic marching cubes. *Computer Graphics Forum 30* (2011), 931–940.

[PWL97] Pang A. T., Wittenbrink C. M., Lodha S. K.: Approaches to uncertainty visualization. *The Visual Computer 13*, 8 (1997), 370–390.

[RCBW12] Reichl F., Chajdas M. G., Bürger K., Westermann R.: Hybrid Sample-based Surface Rendering. pp. 47–54.

[RDT06] Rathi Y., Dambreville S., Tannenbaum A.: Statistical shape analysis using kernel PCA. *Proc. SPIE 6064* (2006), 60641B.

[Reg08] Rege A.: An introduction to modern gpu architecture. NVIDIA Corporation, 2008.

[RLBS03] Rhodes P. J., Laramee R. S., Bergeron R. D., Sparr T. M.: Uncertainty visualization methods in isosurface rendering. In *Eurographics* (2003), vol. 2003, pp. 83–88.

[RM78] Robert McGill John W. Tukey W. A. L.: Variations of box plots. *The American Statistician 32*, 1 (1978), 12–16.

[RM08] Ruuth S. J., Merriman B.: A simple embedding method for solving partial differential equations on surfaces. *J. Comput. Phys. 227*, 3 (Jan. 2008), 1943–1961.

[Ros56] Rosenblatt M.: Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics 27*, 3 (1956), 832–837.

[SCJ*10] Swihart B. J., Caffo B., James B. D., Strand M., Schwartz B. S., Punjabi N. M.: Lasagna plots: a saucy alternative to spaghetti plots. *Epidemiology (Cambridge, Mass.) 21*, 5 (2010), 621.

[Sco92] Scott D. W.: *Multivariate Density Estimation: Theory, Practice, and Visualization (Wiley Series in Probability and Statistics)*, 1 ed. Wiley, Sept. 1992.

[Shn96] Shneiderman B.: The eyes have it: A task by data type taxonomy for information visualizations. In *Proc. IEEE Symposium on Visual Languages* (1996), pp. 336–348.

[Shn98] Shneiderman B.: *Designing the User Interface. Strategies for Effective Human-Computer Interaction*, 3rd ed. Addison-Wesley, 1998.

[SKS12] Schlegel S., Korn N., Scheuermann G.: On the interpolation of data with normally distributed uncertainty for visualization. *IEEE Transactions on Visualization and Computer Graphics 18* (2012), 2305–2314.

[SSSSW13] Schultz T., Schlaffke L., Schölkopf B., Schmidt-Wilcke T.: Hifive: a hilbert space embedding of fiber variability estimates for uncertainty modeling and visualization. In *Computer Graphics Forum* (2013), vol. 32, Wiley Online Library, pp. 121–130.

[Sut05] Sutter H.: The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's journal* (2005).

[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 TVCG 16*, 6 (2010), 1421–1430.

[THM*05] Thomson J., Hetzler E., MacEachren A., Gahegan M., Pavel M.: A typology for visualizing uncertainty. In *Proc. SPIE 5669, Visualization and Data Analysis* (2005), pp. 146–157.

[TK09] Taylor B. N., Kuyatt C. E.: *Guidelines for Evaluating and Expressing the Uncertainty of NIST Measurement Results (rev).* DIANE Publishing, 2009.

[TLB*11] Thompson D. C., Levine J. A., Bennett J., Bremer P.-T., Gyulassy A., Pascucci V., Pébay P. P.: Analysis of large-scale scalar data using hixels. In *Proc. IEEE Symposium on Large Data Analysis and Visualization* (2011), pp. 23–30.

[TPM05] Tory M., Potts S., Möller T.: A parallel coordinates style interface for exploratory volume visualization. *IEEE TVCG 11*, 1 (2005), 71–80.

[TSD09] Tory M., Swindells C., Dreezer R.: Comparing dot and landscape spatializations for visual memory differences. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1033–1040.

[TWSM*11] Torsney-Weir T., Saad A., Möller T., Hege H.-C., Weber B., Verbavatz J.-M.: Tuner: principled parameter finding for image segmentation algorithms using visual response surface exploration. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (2011), 1892–1901.

[Upt08] Upton G.: *A dictionary of statistics.* Oxford University Press, Oxford, 2008.

[Van10] Vanmarcke E.: *Random Fields : Analysis and Synthesis.* World Scientific, Singapore Hackensack, NJ, 2010.

[vN93] von Neumann J.: First draft of a report on the edvac. *IEEE Ann. Hist. Comput. 15*, 4 (Oct. 1993), 27–75. doi:10.1109/85.238389.

[vWvL93] van Wijk J. J., van Liere R.: Hyperslice: visualization of scalar functions of many variables. In *Proceedings of the 4th conference on Visualization '93* (1993), pp. 119–125.

[WB97] Wong P. C., Bergeron R. D.: 30 years of multidimensional multivariate visualization. IEEE Computer Society Press, pp. 3–33.

[WBWK00] Wang Baldonado M. Q., Woodruff A., Kuchinsky A.: Guidelines for using multiple views in information visualization. In *Proc. Working Conference on Advanced Visual Interfaces* (2000), pp. 110–119.

[Weg90] Wegman E. J.: Hyperdimensional data analysis using parallel coordinates. *Journal of the American Statistical Association 85* (1990), 664–675.

[Wel62] Welford B. P.: Note on a method for calculating corrected sums of squares and products. *Technometrics 4*, 3 (1962), 419–420. doi:10.1080/00401706.1962.10490022.

[Wen05] Wendland H.: *Scattered data approximation.* Cambridge University Press, Cambridge, UK New York, 2005.

[Wil11] Wilks D. S.: *Statistical Methods in the Atmospheric Sciences.* Academic Press, 2011.

[WLS13] Wei T.-H., Lee T.-Y., Shen H.-W.: Evaluating isosurfaces with level-set-based information maps. *Comput. Graph. Forum 32*, 3 (2013), 1–10.

[WMK13] Whitaker R. T., Mirzargar M., Kirby R. M.: Contour boxplots: A method for characterizing uncertainty in feature sets from simulation ensembles. *IEEE TVCG 19*, 12 (2013), 2713–2722.

[WPL95] Wittenbrink C., Pang A., Lodha S.: *Verity Visualization: Visual Mappings.* Tech. rep., Santa Cruz, CA, USA, 1995.

[WPL02] Wittenbrink C. M., Pang A. T., Lodha S. K.: Glyphs for visualizing uncertainty in vector fields. *IEEE TVCG 2*, 3 (2002), 266–279.

[WS06] Wang C., Shen H.-W.: LOD Map - a visual interface for navigating multiresolution volume visualization. *IEEE TVCG 12*, 5 (2006), 1029–1036.

[WSE99] Westermann R., Sommer O., Ertl T.: Decoupling polygon rendering from geometry using rasterization hardware. In *Proceedings of the 10th Eurographics Conference on Rendering* (1999), EGWR'99, pp. 45–56.

[WZ13] Wu K., Zhang S.: A contour tree based visualization for exploring data with uncertainty. *Int'l. J. Uncertainty Quantification 3*, 3 (2013), 203–223.

[ZCG05] Zuk T., Carpendale M. S. T., Glanzman W. D.: Visualizing temporal uncertainty in 3d virtual reconstructions. In *VAST* (2005), vol. 2005, p. 6th.

[ZMH*09] Zachow S., Muigg P., Hildebrandt T., Doleisch H., Hege H.-C.: Visual exploration of nasal airflow. *IEEE TVCG 16*, 6 (2009), 1407–1414.

[ZSL*16] Zhang C., Schultz T., Lawonn K., Eisemann E., Vilanova A.: Glyph-based comparative visualization for diffusion tensor fields. *IEEE transactions on visualization and computer graphics 22*, 1 (2016), 797–806.

[ZWK10] Zehner B., Watanabe N., Kolditz O.: Visualization of gridded scalar data with uncertainty in geosciences. *Computers & Geosciences 36*, 10 (2010), 1268–1275.