# Multi-Field Visualization on Graphics Processing Units

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Ralf Peter Botchen

aus Frankfurt am Main

| | |
|---|---|
| Hauptberichter: | Prof. Dr. T. Ertl |
| Mitberichter: | Prof. Dr. M. Chen |
| | Prof. Dr. D. Weiskopf |
| Tag der mündlichen Prüfung: | 19.11.2008 |

Institut für Visualisierung und Interaktive Systeme
der Universität Stuttgart

2008

# Contents

# List of Abbreviations and Acronyms

| | |
|---|---|
| API | application programming interface |
| bit | binary digit |
| cf. | confer |
| CCD | Charge-coupled Device |
| CCTV | Closed Circuit Television |
| CFD | Computational Fluid Dynamics |
| Cg | C for Graphics |
| CPU | Central Processing Unit |
| CT | Computer Tomography |
| CTA | Computed Tomography Angiography |
| DES | Detached-Eddy Simulation |
| DNS | Direct Numerical Simulation |
| Dr. rer. nat. | Doctor rerum naturalium |
| DVR | Direct Volume Rendering |
| ECG | electrocardiogram |
| EBF | Elliptical Basis Functions |
| e.g. | exempli gratia |
| et al. | et alii, et aliae, et alia |
| etc. | et cetera |
| FEM | finite element methods |
| fps | frames per second |
| fMRI | functional Magnetic Resonance Imaging |
| GB | gigabyte |
| GHz | gigahertz |
| GLSL | OpenGL Shading Language |
| GPU | Graphics Processing Unit |
| HLSL | High Level Shading Language |
| IA | interrogation areas |
| i.e. | id est |
| LIC | Line Integral Convolution |
| LES | Large-Eddy Simulation |
| MB | megabyte |
| MHD | Magnetohydrodynamics |
| MRI | Magnetic Resonance Imaging |
| OLIC | Oriented Line Integral Convolution |
| PC | Personal Computer |
| PIV | Particle Image Velocimetry |
| pixel | picture element |
| Prof. Dr. | Professor Doctor |
| RAM | Random Access Memory |
| RANS | Reynolds-Averaged Navier-Stokes |
| RBF | Radial Basis Functions |
| RGB | red, green, blue |
| RGBA | red, green, blue, alpha |
| s | second |
| SIMD | Single Instruction, Multiple Data |
| spf | seconds per frame |
| texel | texture element |
| voxel | volume element |
| VPG | Video Perpetuo Gram |
| WTF | goes without saying |

# Abstract and Chapter Summaries

## Abstract

The generation of multi-field data has become commonplace in many scientific disciplines and application areas today. While researchers have produced numerous techniques for analyzing a single scalar, vector, or tensor field over the last years, finding approaches for exploring multi-field datasets still forms one of the significant challenges in visualization and analytics. One crucial aspect for the growing demand of multi-field visualization techniques is the fact that scientists need to explore the interaction of these fields to gain deeper understanding of underlying processes and relationships. This work addresses the challenge of illustrating multi-field data and presents new approaches of visualization techniques for a variety of application areas, with the aim to map these algorithms to graphics hardware architectures to achieve interactive visualization.

In particular, the main contributions of this thesis contain multi-field flow visualization with one focus on integrating an additional flow uncertainty value, based on measurement simulation, into visualization. Therefore, texture based advection techniques are extended for the transport and display of the additional information. The second focus lies on the illustration of multiple fields as one combined characteristic set to minimize memory usage and allow further feature extraction from the new unique representation. New techniques are developed for multi-field volume rendering in the area of medical applications, with the primary challenge to intermix volumetric data that was acquired by different medical imaging modalities. The proposed solutions give implementation details for raycasting and slice-based rendering of multiple overlapping volumes. The third application area is video visualization. This domain is a typical representative for multi-field visualization, as it combines both, flow fields and multi-volume data for illustration. The goal of the introduced video visualization techniques is to extract dynamic or still objects in a scene, detect their individual actions and the relations among each other and to display this filtered information as a continuous stream of signatures for analysis. Another problematic issue in multi-field visualization is the size of the data, which is usually rather large. Yet, data transfer to and memory size on GPUs are two major bottlenecks. To address this issue, throughout the thesis techniques for data reduction by combination and data bricking for continuous streaming are discussed. Finally, multi-field data encoding and visualization techniques are presented that utilize the advantages of radial basis functions to minimize the data size.

## Chapter Summaries

An overview of the thesis is given in this section as chapter summaries.

### Chapter 1: Introduction

The first chapter introduces the topic of this thesis. Multi-field visualization as such is a relatively new area of research that deals with large data sizes and the challenge to map various overlapping fields to the available screen space for illustration. Thus, the focus of this discussion is the demand for novel multi-field visualization techniques in science and engineering disciplines, and the issues that need to be considered for developing new algorithms that fulfill the requirements to advance analysis and understanding of the phenomena occurring in the data.

### Chapter 2: Graphics and Visualization Fundamentals

In Chapter 2 an overview of visualization technologies and methods is given. The chapter starts by an introduction to visualization in general, then giving a description of the fundamental visualization pipeline and its stages. Next, the various types of grids that are used for the data analysis throughout the thesis are described, followed by the interpolation schemes that can be computed on these grids, to obtain values at arbitrary locations from the data given on discrete grid points.

Subsequently, an overview of basic principles on flow visualization is provided. The section starts with a formal definition of flows, before it describes particle based tracing techniques for steady and unsteady flows as representative for a sparse visualization. As counterpart, for a dense visualization, the texture based line integral convolution method is introduced.

Chapter 2 continues with an overview of volume visualization used for visualizing scalar fields. The workflows of indirect and direct volume rendering methods are described, whereby the direct volume rendering techniques are subdivided into raycasting and slice-based volume rendering. For the latter two techniques, the volume rendering integral is described and blending operators by means of image compositing are given. The discussion of the former is restricted to the Marching-Cubes algorithm for isosurface extraction.

The final part of this chapter details operations on a lower abstraction level. It starts with the rendering pipeline that transforms all geometrical primitives to pixels. Since this transformation is usually performed by graphics hardware and all algorithms in this thesis are developed with the aim to be mapped to commodity graphics hardware architectures, the chapter closes with a discussion on the functionality of graphics processing units.

## Chapter 3: Multi-field Flow Visualization

Chapter 3 starts with an introduction to multi-field flow visualization and outlines related work, published in literature. The following sections exemplify the basic idea of fluid dynamics on the Navier-Stokes equation and outline some of the most important simulation and measuring techniques used to obtain the flow data presented in this work. Based on this flow data, the subsequent section gives an overview of flow features and their mathematical definition, as well as an introduction to possible uncertainties and errors that can adhere the data during acquisition. These uncertainties can also be understood as a kind of flow feature and thus, need to be considered for ensuing visualization.

In the following sections, this chapter presents five novel texture-based techniques to visualize uncertainty in time-dependent 2D flow fields. All methods use semi-Lagrangian texture advection as basis to show the flow direction by streaklines. Three of these methods include an additional step that is derived from a generic filtering process, and then incorporated into the traditional texture advection pipeline as pre-processing or post-processing step. From the perception point of view, these methods indirectly map the uncertainty to intensity of the image, by blurring the streaklines. The two remaining methods use a color mapping approach to display the uncertainty extent on a different perceptional channel, i.e., hue. The visualization methods allow for a continuous change of the density of flow representation by adapting the density of particle injection, and can be mapped to efficient GPU implementations. The usefulness of these techniques is demonstrated for examples of simulation and PIV data sets.

Flow data usually contains various features and multi-field data that bears not only a flow field, but also information about pressure, temperature or even uncertainty, tends to be very large and unwieldy for visualization. To prepare this data for interactive visualization, the next section of this chapter describes a framework that exploits first-order fuzzy logic to combine multiple scalar features, extracted from the original data, to one logical combined feature set. The combination of several feature criteria to one characteristic subset can be used to build one single geometric isosurface representation of several features, and thus, significantly reduces the amount of graphical primitives that would be needed to display all features separately. The created subset is also utilized for particle seeding, with the aim to show the behavior of the flow in the surrounding area.

## Chapter 4: Multi-field Volume Visualization

Volume rendering of multiple intersecting volumetric objects is a difficult visualization task, especially if different rendering styles need to be applied to the objects, in order to achieve the desired illustration effect. Chapter 4 describes a generic technique for multi-volume rendering, discusses implementation details for two rendering approaches and verifies the techniques on several medical application examples.

The chapter begins with an overview of related work in multi-volume visualization and outlines the applied medical imaging techniques that have been used to obtain the test datasets. The general challenges of multi-volume rendering are discussed, before technical details for slice-based multi-volume rendering and multi-volume raycasting are given. The focus of this part is the problem of intermixing several overlapping volumes and the different issues that need to be considered for implementing these approaches. Both techniques are implemented in a generic multi-volume rendering framework that is based on a freely configurable rendering graph. The following sections describe the so called render graph framework and explain the evaluation of the graph as a two-pass process for the dynamic generation of shader programs. The chapter is concluded by the demonstration and discussion of various medical application cases that show the flexibility of the system before it gives a comparison of rendering performance for both techniques.

**Chapter 5: Multi-field Video Visualization**

Video visualization is a computation process that extracts meaningful information from original video data sets and conveys the extracted information to users in appropriate visual representations. The goal of Chapter 5 is to presents a broad treatment of the subject, following a typical research pipeline involving concept formulation, system development, a path-finding user study, an expert survey, and a field trial with real application data. The chapter starts with a motivation on video visualization and outlines related work in this field. The next section introduces concepts and definitions for video visualization that are the basis for the following implementations and studies. The main part of this chapter starts with a summary of a number of image processing modules and filters that are used in the video processing stage to extract meaningful information form the video stream. In particular, this section includes the description of a video transfer function to highlight important features in the video stream, an optical flow filter for estimating motion flows in subsequent images, a seed point generation algorithm for placing flow glyphs in the video, an action recognition and classification filter that is based on a motion descriptor and an object relation filter that computes 1:1 relations for all objects appearing in the scene, and assigns these relations to the objects for visualization.

Based on the extracted information of these filters, the next section presents the video visualization framework that was developed to maintain a user study on visual signatures. The system supports four kinds of signatures that are rendered as a combination of volume and flow visualization in a volume bounding box that is bend to a horseshoe shape for better screen space utilization. Further, this system implements a video volume bricking mechanism to allow a continuous streaming and rendering of large video volumes. The effectiveness of this approach is demonstrated with example visualizations constructed from two benchmarking problems in computer vision. The subsequent section provides details about the assembly, execution and evaluation of a major user study on visual signatures that was accomplished with the renderings produced with the visualization framework. The study demonstrates that video visualization is both technically feasible and cost-effective. Further, the study gives evidence that ordinary users can be accustomed to the visual features depicted in video visualizations, and can learn to recognize visual signatures of a variety of motion events.

The results of the user study inspired to develop new video visualization algorithms as mapping techniques that were validated by an expert survey and led to the implementation of an extended video visualization framework. Based on the visualization expert survey, the second part of this chapter proposes a visualization solution, where a video stream is rendered as abstract illustrative visualization that conveys a continuous stream of multi-field information including extracted objects, detected actions and the estimated relationship between objects. This type of video visualization is named as *VideoPerpetuoGram* (VPG). Therefore, the basic visualization system is extended to handle the raw and processed information of the video stream in a multi-field visualization pipeline, incorporating new the visualization techniques that scored best in the survey. The chapter closes with an evaluation of the extended visualization framework and examines the effective means for depicting multi-field information in VPG.

**Chapter 6: Visualization of Encoded Multi-field Data**

An alternative approach that exploits a compact and universal data representation scheme of radial basis functions (RBF) and ellipsoidal basis functions (EBF) to store multi-field data in the local memory of the graphics hardware is given in Chapter 6. Procedural encoding of scattered and unstructured multi-field datasets using RBFs is an active area of research with great potential for compactly representing large datasets. The reduced storage requirement allows the compressed datasets to completely reside in the local memory of the graphics card, thus, enabling accurate and efficient processing and visualization without data transfer problems.

The introduction of this chapter gives a motivation on the use of procedural encoding for the visualization purpose and outlines a variety of related work in this area of research. The fundamentals of spherical and ellipsoidal basis functions and the workflow of functional approximation using these functions is detailed in the first section of Chapter 6. The steps for procedural encoding of scalar and vector data are discussed in the following section. This part includes domain localization, initial parameter approximation, nonlinear optimization and error measurement for the optimization process based on the H1-norm and the L2-norm.

The main section of this chapter presents a framework for multi-field approximation and visualization, using spherical and ellipsoidal Gaussian functions, which provide greater compression for encoding volumetric and vector data of different type and structure, e.g., uniform meshes, tetrahedral meshes, and meshless representations. The proposed system supports new hierarchical techniques that effectively encode data on arbitrary grids including volumetric scalar, vector, and multi-field data and adapts the employed basis functions to graphics hardware rendering. The effectiveness and performance for both spherical Gaussians and ellipsoidal Gaussians is demonstrated by various GPU-based visualization techniques, such as particle tracing, texture advection, cutting planes, isosurfaces, and volume rendering, build on those techniques introduced in the previous chapters. The chapter is then concluded with an evaluation of a variety of encoded application datasets and a comparison of the different encoding techniques for interactive rendering.

**Chapter 7: Multi-field Techniques in Visualization**

The last chapter concludes this thesis with guidelines for the design of multi-field visualization methods. It gives a complete overview on all proposed techniques and a discussion on the ”lessons learned” by accomplishing the studies and during the realization of the approaches. Further, a review on the achievements and gained insights of all individual techniques is given, followed by some thoughts on still existing challenges that need to be considered for the design of new multi-field visualization techniques in future.

## Zusammenfassung

Die Generierung von Multifeld-Daten ist heutzutage in vielen wissenschaftlichen Disziplinen und auch in praktischen Anwendungsgebieten weit verbreitet. Während Wissenschaftler im Verlauf der letzten Jahre zahlreiche Techniken für die Analyse von einzelnen Skalar-, Vektor-, und Tensorfeldern entwickelten, so bergen Lösungsansätze zur Erkundung von Multifeld-Datensätzen eine komplexere Problematik, die die Forschung im Bereich der Visualisierung und der Analytik vor enorme Herausforderungen stellt. Ein entscheidender Aspekt für die steigende Nachfrage an Multifeld-Visualisierungsmethoden ist die Tatsache, dass Wissenschaftler und Analysten die Interaktion zwischen mehreren in Relation zueinander stehenden Feldern erforschen wollen, um weitere Erkenntnisse über die beobachteten Phänomene zu gewinnen, um so ein tieferes Verständnis der zugrunde liegenden Prozesse zu ermöglichen. Diese Arbeit behandelt Problemstellungen, die bei der Visualisierung von Multifeld-Daten auftreten und präsentiert neue Visualisierungskonzepte und Algorithmen für eine Vielzahl von Anwendungsgebieten. Ziel ist weiterhin, diese Algorithmen auf moderne Grafikhardwarearchitekturen abzubilden, um so verstärkt eine interaktive Darstellung zu gewährleisten, die für einen kontinuierlichen Analyseprozess von großer Bedeutung ist.

Im Einzelnen enthalten die Beiträge in dieser Doktorarbeit Konzepte zur Visualisierung von Multifeld-Strömungsdaten, mit Fokus der Integration eines zusätzlichen Ungenauigkeitsparameters in die Darstellung, der bei Messung oder Simulation der Daten entstehen kann. Für die Umsetzung wird eine bildbasierte Strömungsvisualisierungstechnik, die auch als Texturadvektion bekannt ist, implementiert. Diese Technik wird erweitert, um zusätzliche Informationen zu transportieren und darzustellen. Der zweite Schwerpunkt im Bereich der Strömungsvisualisierung basiert auf einer Technik zur Illustration von mehreren Skalarfeldern als ein logisch kombiniertes charakteristisches Skalarfeld. Der Vorteil dieser Technik liegt zum einen in der Minimierung des Speicherbedarfs, führt zu einer vereinfachten Suche und Extraktion von Strömungscharakteristiken auf dem reduzierten Feld und kann zum anderen als Kriterium verwendet werden, um Partikel in die Strömung zu injizieren. Ein weiterer Teil behandelt neue Techniken zur Multifeld-Volumenvisualisierung von mehreren zusammengehörigen skalaren Feldern aus dem medizinischen Kontext. Die primäre Herausforderung liegt hier im geeigneten Vermischen der einzelnen volumetrischen Daten, die durch verschiedene medizinische Bildgebungsverfahren erzeugt wurden. Insbesondere bei der Wahl eines spezifischen Algorithmus zur direkten Volumenvisualisierung sind hier unterschiedliche Aspekte zu beachten. Die präsentierten Lösungen beinhalten Implementierungsdetails für Raycasting sowie für das Schnittebenen-basierte Verfahren zur Visualisierung von Szenen, in denen sich mehrere Volumen überschneiden. Der dritte Kernbereich dieser Dissertation behandelt die Vi-

deovisualisierung. Dieses Anwendungsgebiet ist ein klassischer Stellvertreter für Multifeld-Visualisierung. Hier werden beide bisher besprochenen Gebiete für die Darstellung der Multifeld-Daten kombiniert. Multifeld-Videodaten beinhalten sowohl Volumendaten von extrahierten Objekten als auch Bewegungsinformationen und somit Strömungsdaten dieser Objekte die zur Visualisierung herangezogen werden. Ziel der vorgestellten Videovisualisierungstechniken ist die Extraktion dynamischer und statischer Objekte aus einer Szene und weiterführend die Detektion von individuellen Aktionen und Relationen, die den einzelnen Objekten zugeordnet werden können. Anhand dieser gefilterten Informationen wird dann eine kontinuierliche Endlosdarstellung des Videos generiert, wobei die extrahierten Objekte in dieser Ansicht als raum-zeitliche Signaturen dargestellt werden, die für die Analyse der Daten von großem Wert sind. Eine weitere Problematik, die bei der Visualisierung von Multifeld-Daten auftitt, ist die Tatsache, dass die Existenz von mehreren, in der Regel recht großen Datenfeldern auch unweigerlich zu einem enormen Speicherbedarf führt. Der Transport großer Datenmengen vom Hauptspeicher zum Grafikspeicher ist jedoch ein wesentlicher Flaschenhals, der für die Realisierung von interaktiven Visualisierungstechniken berücksichtigt werden muss. Verschiedene Lösungsvorschläge für diese Problemstellung finden sich in dieser Dissertation in Form einer Unterteilung der Daten in einzelne Blöcke, die dann kontinuierlich zur GPU heruntergeladen werden können. Auch die Kombination und somit eine Fusion von mehreren Datenfeldern zu einem charakteristischen Datenfeld, das dann platzsparend im Grafikspeicher gelagert werden kann, wird hier vorgestellt. Da diese Form von Datenhandhabung nicht in allen Anwendungsgebieten einsetzbar ist, widmet sich das letzte Kapitel ausschließlich der Rekonstruktion von komprimierten Multifeld-Daten. Durch Approximation mittels radialer Basisfunktionen ist es möglich, komplette Multifeld-Datensätze durch eine neue Datenstruktur zu repräsentieren und so als Ganzes im GPU-Speicher zu lagern. Um diese komprimierten Multifeld-Daten darzustellen, präsentiert diese Arbeit GPU-basierte Dekodierungsalgorithmen, die eine interaktive Rekonstruktion und Visualisierung ermöglichen.

## Kapitelzusammenfassungen

Eine Übersicht dieser Dissertation wird in den folgenden Abschnitten durch eine Zusammenfassung jedes einzelnen Kapitels gegeben.

### Kapitel 1: Einleitung

Im erste Kapitel werden die Themen und die Problemstellungen, die in dieser Dissertation abgehandelt werden vorgestellt. Die Visualisierung von Multifeld-Daten als solches ist eine verhältnismäßig neue Forschungsdisziplin, die sich substanziell mit der simultanen Handhabung von mehreren großen Datenfeldern beschäftigt sowie mit der Herausforderung, diese sich überlappenden Daten möglichst effizient in den Bildraum abzubilden. Einen Ansporn für die entwickelten Methoden in dieser Arbeit gibt vor allem die wachsende Nachfrage nach neuen Multifeld-Visualisierungstechniken sowohl für wissenschaftliche Zwecke als auch in vielen Anwendungsgebieten. Weiterhin enthält dieses Kapitel Problematiken und Fragestellungen, die beachtet werden sollten, um neue Algorithmen für die Darstellung und Analyse von Multifeld-Daten zu entwickeln. Neue Visualisierungsalgorithmen auf diesem Gebiet unterstützen den Anwender im Wesentlichen bei der Analyse der Daten und dem Verständnis der zugrunde liegenden Phänomene, die in den Daten auftreten.

## Kapitel 2: Grafik und Visualisierungsgrundlagen

In Kapitel 2 wird ein Überblick der fundamentalen Visualisierungstechnologien und -methoden gegeben. Das Kapitel beginnt mit einer Einleitung in das Gebiet Visualisierung im Allgemeinen und gibt im Anschluss eine Beschreibung der zugrunde liegenden Visualisierungspipeline, die in der Regel durchlaufen wird, um aus Rohdaten eine aussagekräftige Darstellung zu generieren. Im darauf folgenden Unterkapitel werden die verschiedenen Arten von Gittertypen vorgestellt, die in dieser Arbeit zur Datenrepräsentation verwendet werden, gefolgt von den Interpolationsverfahren, die auf den entsprechenden Gittertypen definiert sind. Interpolation wird verwendet, um Datenwerte an beliebigen Positionen zwischen den diskret gegebenen Datenpunkten des Gitters zu berechnen.

Der folgende Abschnitt beinhaltet die Grundprinzipien der Strömungsvisualisierung und beginnt mit formalen Definitionen aus dem Bereich der Visualisierung und Simulation, gefolgt von Partikel-basierten Verfolgungstechniken als Repräsentant für eine dünne Visualisierung, für stationäre sowohl als instationäre Strömungen. Als Gegenstück dazu wird für eine dichte Repräsentation des Strömungsfelds die texturbasierte Linienintegral-Faltungsmethode (line integral convolution) vorgestellt. Kapitel 2 wird fortgesetzt mit einem Überblick der grundlegenden Volumenvisualisierungsmethoden die häufig eingesetzt werden, um volumetrische Skalarfelder darzustellen. Im Einzelnen werden hier die Abläufe für indirekte und direkte Volumenvisualisierungsverfahren beschrieben, wobei die Techniken zur direkten Darstellung von Volumen in Raycasting und Schnittebenen-basierte Verfahren unterteilt sind. Für die beiden letzteren Methoden wird zunächst das Volumenrenderingintegral eingeführt und dann diskretisiert. Für die diskrete Abtastung des Volumens werden Überlagerungsoperatoren vorgestellt. Diese definieren die gewichtete Summierung der einzelnen Abtastpunkte. Die Diskussion über indirekte Volumenvisualisierung beschränkt sich auf das Marching-Cubes-Verfahren, das dazu verwendet wird um anhand eines Isowertes eine geometrische Isofläche aus dem Volumen zu extrahieren.

Der abschließende Teil dieses Kapitels schildert den Ablauf von Operationen auf einer niedrigeren Abstraktionsstufe. Der Abschnitt beginnt mit der Beschreibung der Rendering-Pipeline, die während der Bearbeitung alle geometrischen Primitive rasterisiert und zu Pixeln des Ausgabebildes aufarbeitet. Da diese Umwandlung normalerweise auf der Grafikhardware durchgeführt wird und alle Algorithmen in dieser Dissertation mit dem Ziel entwickelt wurden, sie letztendlich auf handelsübliche Grafikhardwarearchitekturen abzubilden, schließt dieses Kapitel mit einer Diskussion über den Funktionalitätsumfang von modernen Grafikprozessoren.

**Kapitel 3: Multifeld-Strömungsvisualisierung**

Kapitel 3 beginnt mit einer Einleitung in des Gebiet der Multifeld-Strömungsvisualisierung und gibt einen Überblick über verwandte Arbeiten, die in der Literatur zu finden sind. Die folgenden Abschnitte verdeutlichen die Grundidee der Strömungsdynamik anhand der Navier-Stokes-Gleichungen und umschreiben einige der wichtigsten Simulations- und Messtechniken, die von Strömungsmechanikern eingesetzt wurden, um die in dieser Arbeit verwendeten Daten zu erzeugen. Auf der Basis dieser Strömungsdaten gibt der folgende Abschnitt einen Einblick in unterschiedliche Stömungscharakteristiken und ihre mathematische Definition sowie eine Erläuterung zu möglichen Ungenauigkeiten oder gar Fehlern, die den Daten während der Erzeugung durch Messung oder Simulation anhaften können. Diese Ungenauigkeiten können auch als eine Art Strömungscharakteristik verstanden werden, wobei auch für dieses Attribut die Notwendigkeit besteht, während des Visualisierungsprozesses brücksichtigt zu werden.

In den weiterfürenden Abschnitten stellt dieses Kapitel fünf neue texturbasierte Techniken vor, um Strömungsungenauigkeiten in zeitabhängigen zweidimensionalen Strömungsfeldern sichtbar zu machen. Alle Methoden verwenden das Prinzip der semi-Lagrangen Texturadvektion als Basis, um die Strömungsrichtung durch kontinuierliche Strömungslinen darzustellen. Drei dieser Methoden implementieren einen zusätzlichen Schritt, der von einem generischen Filterprozess abgeleitet wurde und je nach Technik als Vorverarbeitungsschritt oder Nachverarbeitungsschritt in den traditionellen Ablauf der Texturadvektion eingebunden wird. Aus Sicht der menschlichen Perzeption bilden diese Methoden die Ungenauigkeit indirekt auf den Intensitätskanal eines Bildes ab, indem sie die Strömungslinien orthogonal zur Strömungsrichtung verwischen und so die Intensität in diesem Bereich minimieren. Die beiden anderen Methoden verwenden den Ansatz einer Farbkodierung, indem sie die Ungenauigkeiten anhand einer Farbtabelle auf einzelne Farbwerte abbilden. Aus perzeptueller Sicht sind Farb- und der Intensitätskanal strikt getrennt und daher die unterschiedlichen Methoden auch in Kombination gut wahrnehmbar. Die texturbasierten Visualisierungsmethoden erlauben einen kontinuierlichen Übergang der Dichterepräsentation von dünner bis hin zu dichter Partikelverteilung zur Strömungsdarstellung, indem sie eine variable Anpassung der Dichte für die Partikelinjektion erlauben. Alle Techniken wurden als leistungsfähige GPU-Implementierung realisiert und ermöglichen so eine interaktive Visualisierung. Die Nützlichkeit dieser Techniken wird anhand mehrerer Anwendungsbeispiele aus der Simulation und echter Messungen, die mit der PIV-Methode durchgeführt wurden, demonstriert.

Strömungsdaten enthalten für gewöhnlich eine Vielzahl verschiedener Charakteristiken. Multifeld-Daten, die nicht nur aus einem Strömungsfeld bestehen, sondern auch Informationen über Druck, Temperatur oder Ungenauigkeiten enthalten, neigen dazu, von der Datenmenge her sehr groß und daher für die Visualisierung recht unhandlich zu sein. Wie diese Datenmengen für eine interaktive Visualisierung vorverarbeitet werden können, beschreibt der folgende Absatz dieses Kapitels. Hier wird ein Framework beschrieben, das durch Anwendung einer Fuzzy-Logik erster Ordnung mehrere skalare Charakteristiken zu einer charakteristischen Menge logisch zu kombiniert. Dieses neu generierte Skalarfeld repräsentiert die charakteristische Menge aller kombinierten Felder und kann dazu genutzt werden, um geometrische Isoflächen anhand eines vordefinierten Isowertes zu extrahieren. Durch diesen Vorverarbeitungsschritt verringert sich der Aufwand für die Visualisierung, da die Anzahl der geometrischen Primitive, die aus dem kombinierten Feld generiert werden, maßgeblich geringer ist als die Anzahl, die entstehen würde, wenn die Extraktion auf jedes einzelne Feld angewendet wird. Die extrahierte geometrische Untermenge wird daraufhin für eine gezielte Partikelinjektion herangezogen, mit dem Ziel, das Verhalten der Strömung in der Umgebung dieser Charakteristischen Isofläche zu zeigen.

## Kapitel 4: Multifeld-Volumenvisualisierung

Die Darstellung von mehreren sich gegenseitig überlappenden volumetrischen Objekten ist eine große Herausforderung für die Volumenvisualisierung, insbesondere mit der Anforderung, unterschiedliche Teilmengen der Objekte mit verschiedenen illustrativen Stilen darzustellen. Kapitel 4 beschreibt eine generische Technik zur Visualisierung von skalaren Multifeld-Daten, bespricht Implementierungsdetails für zwei unterschiedliche Lösungsansätze und validiert diese Techniken anhand einer Vielzahl von medizinischen Anwendungsbeispielen.

Das Kapitel beginnt mit einem Überblick der verwandten Arbeiten im Bereich der Multifeld-Volumenvisualisierung und gibt anschließend einen Einblick in die wichtigsten medizinischen Messverfahren, die zur Datenakquisition der verwendeten Testdaten eingesetzt wurden. Die Herausforderungen zur Entwicklung von Methoden für die Multifeld-Volumenvisualisierung werden im folgenden Abschnitt besprochen, bevor technische Details für Schnittebenen-basiertes Multifeld-Volumenrendering und Multifeld-Volumenraycasting spezifiziert werden. Der Fokus des nächsten Teils ist die Problematik, die beim Vermischens oder dem so genannten ”blending” unterschiedlicher, sich überlappender Entitäten entsteht und die bei der Implementierung der beiden Ansätze beachtet werden muss. Die beiden Techniken sind in ein generisches Framework eingebunden. Dieses basiert auf einem frei konfigurierbaren Graph-Konzept, dem ”Rendergraph”. Die folgenden Abschnitte beschreiben den Aufbau dieses Rendergraph-Frameworks und erklären die Evaluierung des Graphen als einen zweistufigen, dynamischen Prozess, der für die Generierung der Shader Programme verantwortlich ist. Das Kapitel wird durch die Demonstration und die Diskussion von verschiedenen medizinischen Anwendungsszenarien abgeschlossen, die die Flexibilität und die Einsatzvielfalt des Systems zeigen. Beide Visualisierungstechniken werden auch auf ihre Leistungsfähigkeit hin untersucht und verglichen.

**Kapitel 5: Multifeld-Videovisualisierung**

Videovisualisierung ist ein Prozess, in dem sinnvolle Informationen aus den originalen Videodaten extrahiert werden, um sie dann dem Benutzer in einer möglichst effektiven und verständlichen Form darzustellen. Das Ziel von Kapitel 5 ist eine relativ breite Abhandlung dieses Themengebiets und folgt dem Prinzip eines typischen Forschungsablaufes, von der Konzeptformulierung, zur Systementwicklung eines Prototypen über die Durchführung einer weitreichenden Benutzerstudie und einer Expertenbefragung, bis hin zu Feldversuchen mit realen Anwendungsdaten.

Das Kapitel beginnt mit einer Ausführung von Beweggründen, die zur Videovisualisierung geführt haben und beschreibt dann bereits bestehende Arbeiten auf diesem Gebiet. Der folgende Abschnitt stellt wichtige Konzepte und Definitionen der Videovisualisierung vor, die die Basis für folgende Implementierungen und Benutzerstudien bilden. Der Hauptteil dieses Kapitels gliedert sich in eine Zusammenfassung der Bildverarbeitungsmodule, die im System der Videoverarbeitung eingesetzt werden, um Informationen aus den Rohdaten zu extrahieren. Insbesondere umfasst dieser Abschnitt die Beschreibung einer Videotransferfunktion, um wichtige Eigenschaften im Datenstrom hervorzuheben, einen optischen Flussfilter, um Bewegungen von Objekten in aufeinander folgenden Bildern zu berechnen und einen Algorithmus zur Saatpunktgenerierung anhand derer sich Richtungsglyphen im Video platzieren lassen. Weiterhin wird hier ein Aktionserkennungs- und Klassifikationsfilter vorgestellt sowie ein Objekt-Relationsfilter, der $1:1$-Relationen für alle erkannten Objekte berechnet, die in der Szene erscheinen und diesen einen Relationskoeffizient zuweist, der dann zur Darstellung verwendet wird.

Basierend auf den extrahierten Informationen der beschriebenen Filter, stellt der folgende Abschnitt ein Framework zur Videovisualisierung vor, das entwickelt wurde, um eine Benutzerstudie zum Thema visuelle Signaturen in der Videovisualisierung zu ermöglichen. Das System unterstützt vier Arten von Signaturen, die aus einer Kombination von Volumen- und Strömungsvisualisierung erzeugt werden und die in einer zur Hufeisenform gebogenen Box im Bildraum dargestellt werden. Diese besondere Darstellungsform wurde gewählt, da sie den vorhandenen Bildraum am besten ausnutzt. Weiterhin enthält dieses System einen Datenunterteilungsmechanismus, um die Handhabung großer Videodaten zu vereinfachen. Der Mechanismus teilt die Daten in kleinere Stücke auf und ermöglicht somit einen Transfer von kleineren Blöcken zur Grafikhardware für eine kontinuierliche Wiedergabe. Die Effektivität dieses Ansatzes wird anhand verschiedener Beispielvisualisierungen demonstriert, die aus typischen Benchmarking-Referenzdaten, wie sie im Forschungsbereich des Bildverstehens zu finden sind, generiert wurden. Der folgende Abschnitt liefert Details über den Aufbau, die Durchführung und die Auswertung der Benutzerstudien zum Thema visuelle Signaturen, die mit Hilfe des Frameworks erstellt wurden. Die Studien zeigen, dass Videovisualisierung sowohl technisch umsetzbar als auch kosteneffektiv einsetzbar ist. Weiterhin liefert die Studie den Beweis, dass gewöhnliche Benutzer die Interpretation von visuellen Signaturen selbst für eine Vielzahl von Bewegungsereignissen erlernen können.

Die positiven Resultate der Benutzerstudie führten zur Entwicklung von neuen Videovisualisierungstechniken, deren Entwurf im Vorlauf durch eine Befragung von mehreren Experten validiert wurde. Die Ergebnisse sind in einem erweiterten Videovisualisierungsframework implementiert. Basierend auf der Expertenvalidierung, beschreibt der zweite Teil von Kapitel 5 einen Lösungsansatz zur Videovisualisierung, in dem der Videostrom als kontinuierliche, illustrative Signatur dargestellt wird, wobei diese abstrakte Illustration alle extrahierten Multifeld-Daten wie z.B. Objekte, Aktionen und Relationen enthält. Diese Art der Videovisualisierung wird im Folgenden als *VideoPerpetuoGram* (VPG) bezeichnet. Folglich wird das auf einer Multifeld-Visualisierungspipeline bestehende System erweitert, um die Rohdaten des Videostroms zu bearbeiten und die extrahierten Informationen anhand der neuen Techniken darzustellen. Das Kapitel schließt mit einer Auswertung des erweiterten Frameworks und überprüft, wie wirkungsvoll Multifeld-Informationen in einem VPG illustriert werden können.

## Kapitel 6: Visualisierung von Enkodierten Multifeld-Daten

Ein alternativer Ansatz, der eine kompakte und universelle Datenrepräsentation von radialen Basisfunktionen (RBF) und ellipsoidalen Basisfunktionen (EBF) zur Speicherung von Multifeld-Daten ausnutzt, wird in Kapitel 6 vorgestellt. Dieser Ansatz der Datenkodierung ist sehr nützlich, um große Mutlifeld-Daten zu komprimieren und so als Ganzes im lokalen Speicher der Grafikhardware zu halten. Der Einsatz von radialen Basisfunktionen findet in der Forschung verstärkt Anklang und birgt ein großes Potetnial zur Approximation von Daten. Durch die effiziente Kodierung als Vorberechnungsschritt und die Möglichkeit zur echtzeitdekodierung auf der Grafikhardware während der Darstellung ist diese Methode sehr interessant für die Visualisierung, da so Datentransferprobleme und auch der akute Mangel an lokalem Speicher umgangen werden kann.

Die Einleitung dieses Kapitels motiviert den Einsatz dieses prozeduralen Kodierungsverfahrens für die Visualisierung und beschreibt eine Auswahl von verwandten Arbeiten in diesem Forschungsgebiet. Die Grundlagen für radiale und ellipsoidale Basisfunktionen und der Arbeitsablauf der funktionalen Approximation unter Verwendung dieser Basisfunktionen werden im ersten Abschnitt von Kapitel 6 beschrieben, gefolgt von den einzelnen Schritten für die Kodierung von Skalar- und Vektordaten. Dieser Teil umfasst Gebietslokalisierung, Abschätzung der Ausgangsparameter, nichtlineare Optimierung und Bestimmung des Fehlermaßes für den Optimierungsprozess, basierend auf der H1-Norm und der L2-Norm.

Der Hauptteil dieses Kapitels beschreibt ein Framework für die Visualisierung von kodierten Multifeld-Daten, die durch radiale oder ellipsoide Gauss-Funktionen repräsentiert werden. Diese Funktionen bieten im Vergleich zu anderen Basisfunktionen eine bessere Kompression von skalaren und vektoriellen Daten unterschiedlichster Art und Struktur und lassen sich auf eine Vielzahl von Gittertypen anwenden, wobei die resultierende Datenstruktur gänzlich ohne Gitter auskommt. Das vorgestellte System unterstützt weiterhin hierarchische Techniken, die eine effektive Kodierung der unterschiedlichen Daten bezüglich der gewählten Basisfunktion begünstigt sowie eine Oktalbaum-basierte Unterteilung der kodierten Daten, die zur Beschleunigung der Visualisierung mittels Grafikhardware eingesetzt wird. Im letzten Abschnitt dieses Kapitels werden die Effektivität und Leistungsfähigkeit der beiden Ansätze radialer und ellipsoidaler Gauss-Funktionen durch verschiedene GPU-basierte Visualisierungstechniken getestet. Dazu gehören z.B. Partikelverfolgung, Texturadvektion, Schnittebenen, Isoflächen Extraktion und Volumenvisualisierung auf Basis der kodierten Multifeld-Daten. Das Kapitel resümiert mit der Evaluierung aller vorgestellten Kodierungen und Visualisierungstechniken anhand einer Vielzahl von Anwendungsdatensätzen und gibt einen Vergleich der verschiedenen Techniken in Bezug auf eine interaktive Darstellung.

## Kapitel 7: Multifeld-Techniken in der Visualisierung

Das letzte Kapitel vollendet diese Dissertationsschrift und gibt Vorschläge und Richtlinien für den Entwurf von Multifeld-Visualisierungsmethoden. Das Kapitel enthält eine komplette Übersicht über alle vorgestellten Techniken und eine Diskussion über die Lektionen, die während der Durchführung der Studien und der Implementierung der Ansätze gelernt wurden. Weiterhin beschreibt dieses Kapitel die Entwicklung und die gewonnenen Erkenntnisse jeder einzelnen Visualizierungstechnik, gefolgt von einigen Gedanken zur weiterhin existierenden Herausforderung, die für die zukünftige Entwicklung von Multifeld-Visualisierungsmethoden besteht.

# CHAPTER 1 INTRODUCTION

Visualization has become a fundamental tool in scientific research and engineering disciplines. The analysis of complex, multi-dimensional and large-scale data is recognized as an important component in areas including computational fluid dynamics (CFD), medical imaging, weather modeling, computational mechanics, manufacturing industry and chemical engineering. In these areas, the output of specific simulation and measuring techniques can be a single scalar field, or more commonly, a combination of various fields consisting of scalar, vector or tensor data, making the amount of information available to the analyst indefinite. In recent years researchers have concentrated on finding effective ways to visualize a single field variable, like the extraction of isosurfaces from a scalar field or the representation of the flow behavior in a velocity field by arrow glyphs or tracelines. Even though in many cases the visualization of a single field is adequate to satisfy the user needs, it is conceivable that a simultaneous visualization of multiple fields would be useful and of great value for the analysts, to give a better understanding of potential influences between those fields.

As stated by Chris Johnson in 2005, in his visualization viewpoints article [69], there are several issues that should be taken into account in terms of finding effective visualization methods for given problems and for making advances in visualization research, independent of the source of the data and the application area. One of the listed points of research problems is to find new visualization techniques for multi-field data. A crucial aspect for multi-field data visualization is to be aware that both, the number of independent variables (like space and time) and the number of dependent variables (like density, velocity vector or temperature) can be arbitrary. This fact needs to be considered for classification and extraction of features, and the subsequent visualization, which makes it a difficult task. However, the extraction of various features from different fields is becoming very important as the need to show a combination of features and to highlight their correspondence and a possible interaction between those fields is increasing for the purpose of analysis. This includes the identification of important features as well as tracking their signatures evolving in the data over time. Besides the challenges of classifying features and showing correspondence of different features, a major difficulty in developing visualization methods for multiple fields is to find an adequate representation that displays the extent of all involved attributes without cluttering the visualization display or occluding one feature by another.

When a suitable representation for the underlying data is found, regardless if it is a single field or of multi-field type, another aspect for providing a good visualization is to give information about the reliability of the resulting illustration. While it is very common in other fields of science and engineering to show 2D graphs that represent error or uncertainty occurring within experimental or simulation data, visualization research nearly completely ignored this issue and despite from a few exceptions, most visualizations lack to give information about the correctness of representing the underlying data. One reason why this should be done is that most of the extraction and visualization algorithms are not always capable of either analyzing or representing the desired information of the data with high accuracy. These uncertainties or errors can be caused by border conditions in simulations, technical difficulties like a proper calibration for measurements or interpolation inaccuracy of missing data values for visualization, making it essential to include them as additional information for the analyst. This point becomes even more important for multi-field data, as the sources of possible errors increase with every variable added for visualization.

Not only are adequate visual representations of the underlying data and its according reliability of great importance. As multi-field data often has a complex three-dimensional structure, the possibility of real-time interaction with the system has an essential meaning for a fast and effective understanding of the illustrated data. In general, the minimal claim for interactive frame rates lies between five to twenty frames per second, depending on the application. To achieve interactivity, the use of modern graphics hardware and its massive computation power of parallel stream processors is predestinated. The rapid progress in the design of graphics hardware with new functionality and more computation power lead to very flexible, high-performance GPUs which provide freely programmable units, such as the vertex processor, geometry processor and fragment processor. The utilization of these units for the purpose of interactive visualization makes it more efficient and more effective. Efficiency is obtained by accelerating the visualization algorithms using special features supported by modern GPUs. More effectivity results from accelerated visualizations, as they allow interaction and thus, a more productive work during data analysis. Therefore, the presented algorithms are developed with the aim to run on GPUs.

## 1.1 Goal of this Thesis

All algorithms presented in this work have been developed to handle multi-field data, under consideration of the discussion above and with the aim to be mapped to graphics hardware architectures and thus, the main objectives that are of concern can be summarized by four questions:

1. How can multi-field (large-scale) data, which has its origin in a broad range of application fields and which may be acquired from various data sources, be processed effectively in uncompressed or compressed form?
2. What has to be considered to develop novel algorithms to extract important features or signatures from the raw multi-field data and visualize a combination of them most clearly at a time?
3. In which way can errors and uncertainties inhering the data from faulty measurements, algorithmic instabilities or reliability of the algorithms be included into visualization?
4. How can we creatively map new visualization algorithms to graphics hardware, exploiting their computation power to effectively facilitate interactive visualization?

With the intention to give answers to these questions, GPU-based multi-field methods have been designed for volume rendering and flow visualization and applied to data from application areas such as medical imaging, computational fluid dynamics and video surveillance recordings, to cover the application field as broad as possible. In detail, the presented flow visualization methods consider multiple fields with the focus on displaying arbitrary combinations of these fields and showing inherent uncertainties of simulation or measuring techniques. The proposed volume visualization of medical data aims at the combined illustration of multiple data from CT, MRI and fMRI scans, to enhance the pre-operative analysis and shows the difficulties and solutions for a slice-based and a raycasting approach. Video visualization in general is of multi-field nature, due to the resulting information of object extraction and action recognition algorithms and combines volume rendering of 3D object traces and flow visualization of object movement. The goal is to illustrate a maximum amount of information— that is usually annotated by text or labels and displayed in this form, would usually lead to cluttering and occlusion—in combination with the certainty of the extraction algorithms in one final image. Eventually, the use of radial basis functions is presented to compress large-scale multi-filed data. This technique can be applied to reduce the size of the data, before passing it to the limited memory of the graphics hardware and to uncompress it on the GPU either for volume or flow visualization.

This work brings all presented methods in a context, and tries to extract guidelines for the implementation of multi-field visualization techniques on GPUs.

However, the research on multi-field visualization is a young scientific discipline with quite a lot of new interdisciplinary problems that could not all be addressed here, as they are diverse enough to induce the need for several new strategies. Nevertheless, the presented techniques might give other researchers a paradigm and inspiration to develop new multi-field visualization algorithms and help them to solve similar problems.

## 1.2 Acknowledgments

I am most grateful to my advisor Thomas Ertl, who supported and guided my work and provided a great environment to work in. Thanks for giving me the chance to become acquainted with so many interesting fields of research, work with so many professional people and attend all the conferences that have been important for me making progress and developing new ideas. I am more than thankful to my advisor Daniel Weiskopf, who prepared me to take this step into scientific research and for leading my way through the research jungle, spending so much of his valuable time for discussions and explanations, even though he was located on the other side of the planet. Thank you for always being positive and supportive on my work. I am highly indebted to Min Chen for his guidance and support in the video visualization project. With all his invaluable comments and constructive suggestions, not only professionally, he became more than a mentor to me. I specially appreciate all the realistic and honest advices he gave me. Furthermore, I want to thank him for the stay in Swansea and the opportunity to give a talk in London. I am deeply grateful to Rul Gunzenhäuser for his confidence in my work, and the distinction of my diploma thesis, which gave me a lot of motivation.

I would like to express my gratitude to Luis G. Nonato for the good time in Brazil and for his valuable comments and discussions on the on-going work on higher-order data visualization. Unfortunately this work is not included in this thesis. I am very thankful to Ícaro L. L. da Cunha, who helped me a lot with different formalities in São Carlso and showed me around in São Paulo. Many thanks to Valdecir Polizelli-Junior for keeping company and giving me a deep insight into the complex system of Brazilian phone numbers. I am very grateful to João P. Gois for his help organizing my access to the bandejão and especially for picking me up at the bus station in the middle of the night. Further, I want thank the German Academic Exchange Service (DAAD) for providing the financial support for my stay at the University of São Paulo, São Carlos, and for carrying out the good cooperation between Germany and Brazil.

Many thanks to Tobias Schafhitzel for the long constructive discussions; for urging each other in hard times; and for his help on many different matters at the University. Beyond that, thanks for being my climbing fellow all the time. A tribute goes to Friedemann Rößler for our sometimes "painful" work on medical multi-volume rendering that we fortunately both survived in the end; and to Eduardo Tejada for his valuable help on different matters and for our still on-going work on higher-order data visualization.

I express gratitude to Yun Jang, Jingshu Huang, Manfred Weiler, Simon Stegmaier, David S. Ebert and Kelly P. Gaither, who have been the collaborators in my first project and helped me a lot in understanding how this work has to be accomplished. Sven Bachthaler, Rudy R. Hashim and Ian Thornton were of particular importance to me in the video visualization project. Thank you for the support.

Many thanks to both, Thomas Klein my all time office mate for the many constructive discussions and all the funny situations in our office; and Magnus Strengert for answering an awful amount of technical questions about visualization and graphics hardware, and for the collaboration in the spectral volume rendering project, which was the most practical theoretic work ever, but is unfortunately not included in this thesis. Moreover, I thank the students I supervised and with whom I worked with over the last years in alphabetical order, Thomas Derr, Jochen Eggert, Andreas Lauser, Jörg Oberfell, Tim Reiner, Fabian Schick, Thorsten Schmidt, Mikael Vaaraniemi and Edmund Wolf.

Acknowledgment to Ulrike Ritzmann, for her help with the formalities; and to the persons I had the pleasure to work with at the Universität Stuttgart; in particular (in alphabetical order, without the ones mentioned above), Wolfgang Bayerlein, Katrin Bidmon, Harald Bosch, Marianne Castro, Carsten Darchsbacher, Joachim Diepstraten, Mike Eissele, Thomas Engelhardt, Martin Falk, Steffen Frey, Mark Giereth, Frank Grave, Sebastian Grottel, Gunter Heidemann, Julian Heinrich, Marcel Hlawatch, Benjamin Höferlin, Andreas Hub, Steffen Koch, Sebastian Klenk, Steffen Koch, Hermann Kreppein, Andreas Langjahr, Dietmar Lippold, Julia Möhrmann, Christoph Müller, Thomas Müller, Guido Reina, Matthias Ressel, Martin Rotard, Filip Sadlo, Harald Sanftmann, Thomas Schlegel, Martin Schmid, Bernhard Schmitz, Waltraud Schweikhardt, Christiane Taras, Markus Üffinger, Joachim Vollrath and Michael Woerner.

A big shout-out goes to my best friend Markus *"W1"* Weigel for never letting me down, even if sometimes it was hard to keep in contact. Thanks for all the good times we had over the last years, especially for all the fun battles in those MMORPGs we played. Many thanks to my friend Oliver *"NBK"* Zweigle for forming a tight DJ team over more than five years, tearing clubs down in the south and being a part of the legendary FF Crew. I am more than thankful to my friends Roman de Giuli aka. *"Perryroman"* and Jerome Kuhn aka. *"Ialone"* for being part of the band *"Reimheitsgebot"* and rocking concerts together all over Germany, but most of all for being true friends over nearly two decades now – holler!

It goes without saying that I am most thankful to my family for their constant support and belief in me. To my sister for exploring the world together with nothing more than a backpack and inspiring me on tour. But most of all I am grateful to my mum and my dad, because for all my trials and tribulations you had the invaluable piece of advice and gave me the strength to struggle on when it was necessary. For always being the backup I could count on in hard times, I am more than grateful. *”Everything’s gonna be alright.”*

Ralf Peter Botchen

# Chapter 2 Graphics and Visualization Fundamentals

This chapter is supposed to provide visualization fundamentals and an overview of previous work, which is significant for the comprehension of the employed techniques presented in this thesis. It gives a general survey on visualization problems, the kind of data grids that were processed during this work, basic visualization techniques and visualization related technologies that have been adopted and extended to implement new scientific multi-field visualization methods.

## 2.1 Visualization

One of the most distinctive and accurate cognitive organs is the human visual system. It has developed and improved its functionality over many millenniums to perceive and process an immense bandwidth of information. The visualization of abstract or concrete information is not a practical invention of the computer era, a nice example of early visualization are ancient Paleolithic art and cave paintings, created by early mankind all over the world, to illustrate their life, rituals and surroundings. In Figure 2.1, the left image shows one of the oldest, human made, realistic images of large animals. These paintings were found in the Lascaux cave in France and are estimated to be 16,000 years old. The paintings on the right side of Figure 2.1 show a group of hunters tracing down a herd of elephants. This piece is dated back 8,000 years and has been discovered in the Cederberg cave in South Africa. Even in this early stage of visualization, illustrations have been used to record multiple important information – like the execution of a procession or the way of hunting a herd animals and the number of hunters needed to trace them down – and to communicate this knowledge to the posterity.

The perpetual evolution to today's computer age led to several new forms of creating and gathering data. With each novel technology, the amount of producible data is steadily increasing and in the majority of cases, this data is very abstract and difficult to understand for the human in its raw form. This requires more

Figure 2.1: Examples of early visualization, showing the ritual of hunting animals. Left: Paleolithic art in the Lascaux cave in France. Right: Cave art in the Cederberg cave in South Africa. Images courtesy of Wikimedia®.

sophisticated methods to extract and illustrate only important information that is contained in the data. Like statistics, visualization aims at analysis and correct interpretation of this information, both quantitative and qualitative. Therefore, the focus of visualization is to gain more insight into any kind of data and to bring the contained information as a picture before the mind. It is simply a way of making things clearer and more understandable, because if you can see it then it becomes more credible. The insights provided by visualization can support the work in other scientific fields and might as well improve the daily public live.

## 2.2 The Visualization Pipeline

The visualization of scientific data plays a significant role in research projects just as it is important for advances in industry. Data sets resulting from real world measurements or numerical simulations are often so large and complex that it is not possible to see and understand the important parts without applying a proper visualization technique. Thus, the goal of scientific visualization is to make visible the invisible, which means to extract the most interesting information from a dataset and to transform it into a visual representation that can be understood and analyzed. To obtain this goal, the data has to go through several stages of processing that can be described by a multi-stage pipeline, as shown in Figure 2.2. This visualization pipeline describes the way from *data acquisition* over *filtering* and *mapping* to *rendering* and is one of the most valuable concepts in visualization [52].

*Data acquisition* is the first stage of the pipeline and includes all steps that are required to obtain the *raw data*. In this work, the processed data ranges form numerical simulations (DNS, LES, DES) of flow phenomena over (PIV) measurements of fluid flow and aerosols, discussed in Chapter 3, to sensor data from

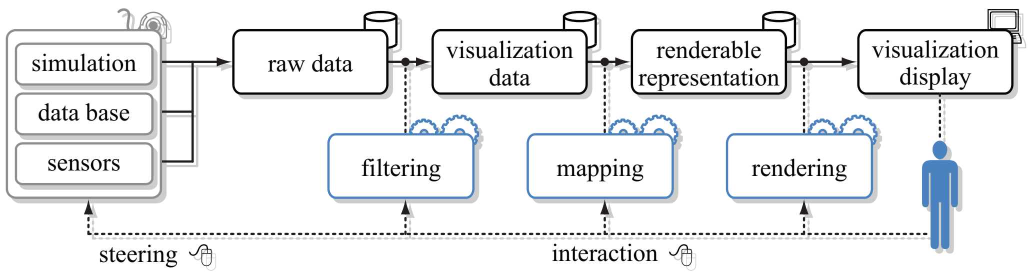


Figure 2.2: The visualization pipeline.

medicine (CT, CTA, MRI, fMRI), detailed in Chapter 4, as well as video data acquired by (CCTV) cameras used in Chapter 5. Apart from the mentioned data sources, visualization techniques are also used to illustrate collected information stored in data bases instead of simulation or measurements, e.g., the spreading of diseases, the distribution of health care, economical fluctuations or the statistical analysis of a survey. This work will only examine this kind of data marginally for the utilization of visualization techniques and rather proposes methods to process scientific data as mentioned above. Depending on the data source and the acquisition technique, the *raw data* might require some kind of pre-processing. Therefore, in the *filtering* stage the potentially very large *raw data* is prepared for visualization by data reduction or data enhancement. Usually user centered sub sets of data are selected and extracted as *visualization data*. Operations that rank among this stage are smoothing filters, interpolation, correcting erroneous measurements and segmenting, registering or labeling the data. The *mapping* stage transforms the *visualization data* into a *renderable representation*. The selected focus data are mapped to a set of geometric primitives (e.g., points, lines, triangles) and additional attributes (e.g., color, texture, size) that can be used in the *rendering* stage to produce displayable images. In modern visualization systems, this *rendering* stage is processed by graphics hardware and the geometric primitives are passed through another pipeline – the so-called *rendering pipeline* – where they undergo several operations like viewing transformation, lighting, clipping or projection, before they are rasterized. The functionality of the *rendering pipeline* will be introduced in Section 2.6. The final step is the *visualization display*, where the rasterized output of the rendering process is usually directly displayed on an output device as single image or as animated sequence of a pre-computed video.

An essential aspect of the visualization process is the possibility for the user to interact with all stages of the pipeline and to get a direct feedback of the system (cf. Figure 2.2). In many cases and especially for data that is unknown to the user, only an explorative approach of data analysis leads to a better insight. This requires visualization methods that support an interactive tuning of all parameters.

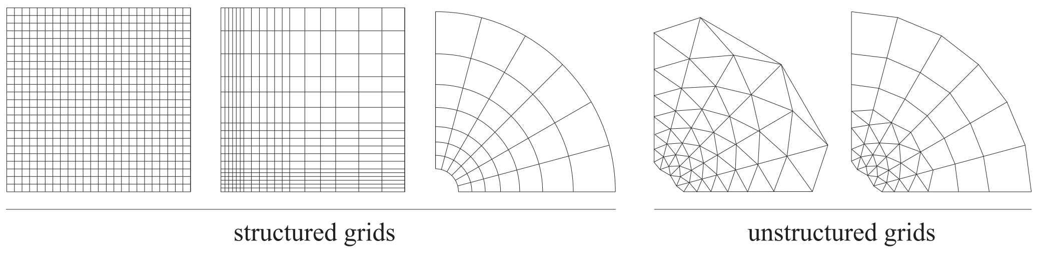


Figure 2.3: Common grid types in scientific visualization. From left to right: Cartesian grid; rectilinear grid; curvilinear grid; unstructured simplex grid; unstructured zoo grid. Any combination of these grid types is possible.

## 2.3 Grid Types and Data

Depending on the method of measuring or simulation, data for scientific visualization have different characteristic attributes that allow a classification. The data can occur in different form or structure, the dimensionality of the data domain can vary (0D, 1D, 2D, 3D, ...) and the type of the data might be of scalar or vector form. If the data type is of higher dimensionality, it is called tensor or multivariate data. This kind of data differs from the previously mentioned in a way that it can contain various attributes that do not necessarily have to correlate, e.g., pressure, temperature and velocity.

Independent of the topology, the data is usually stored as a set of tuples containing a spatial position and one or more data values of arbitrary type, also known as data points. A data structure is called grid, if two neighboring data points can be connected by an imaginary line and thus form the grid topology. Gridless data given as a set of points without topology is also called *scattered data*. A multitude of data structures have been proposed, to satisfy the different needs for specific applications. Figure 2.3 shows a rough classification of grid types used in scientific visualization; they can be subcategorized into *structured* and *unstructured* grids. The relevant grid types as well as the fundamentals of point clouds, used in this work should be described in the following.

### 2.3.1 Structured Grids

For structured grids, the connectivity of grid points is implicitly given and need not to be stored with each data value. As shown in Figure 2.3, there exist different types of structured grids that can be further classified as *Cartesian*, *uniform*, *rectilinear* and *curvilinear*. The simplest grid is the Cartesian grid, of which all cells have the same form and the cell size is of unit length in each dimension. This grid type is preferred in many applications since it can be stored in a plain array and thus, significantly eases data access and simplifies the complexity of algorithms.

Uniform grids are very similar to Cartesian grids, the cells are in general but not necessarily oriented along the orthogonal axes of coordinates. But in contrast to a Cartesian grid, the cells have been scaled non-uniformly in at least one dimension and form cuboids. With subject to the size $\Delta x \times \Delta y \times \Delta z$ of a cubic cell, the position of the vertices $\mathbf{x}^{unif}_{i,j,k}$ in an uniform grid of dimension $n_x \times n_y \times n_z$ is defined as

$$\mathbf{x}^{unif}_{i,j,k} = \mathbf{t} + \mathbf{O} \begin{pmatrix} i\Delta x \\ j\Delta y \\ k\Delta z \end{pmatrix}, \qquad i = 0, \ldots, n_x;\ j = 0, \ldots, n_y;\ k = 0, \ldots, n_z \quad ,$$

where $\mathbf{O}$ is a rotation matrix and $\mathbf{t}$ a translation vector that is applied, if the grid point with indices $(0, 0, 0)$ in world coordinates does not match with the origin.

In contrast to the constant cell size of Cartesian and uniform grids, the generalized rectilinear grids may have grid point distances $\Delta x$, $\Delta y$ and $\Delta z$ of arbitrary size which depends on the indices $i$, $j$ and $k$ respectively. Therefore, the vertex positions $\mathbf{x}^{recti}_{i,j,k}$ can be computed by

$$\mathbf{x}^{recti}_{i,j,k} = \mathbf{t} + \mathbf{O} \begin{pmatrix} x(i) \\ y(j) \\ z(k) \end{pmatrix}, \qquad i = 0, \ldots, n_x;\ j = 0, \ldots, n_y;\ k = 0, \ldots, n_z \quad ,$$

with $x(i)$, $y(j)$ and $z(k)$ as coordinate functions.

Curvilinear grids have the same connectivity as rectilinear or uniform grids, but they do not possess an implicit definition for the vertex positions. Instead, their construction follows a curved line and the position for each vertex $\mathbf{v}^{c}_{i,j,k}$ has to be stored as additional vector. These grids are well suited to model simulation domains that are bent around a curved obstacle, i.e., a cylinder. Although the cells of curvilinear grids are non-intersecting and conforming, they bear some difficulties for visualization and are often resampled for this purpose.

Two-dimensional and three-dimensional structured grids are widely-used in car manufacturing and aerospace industry to run simulations or record sensor data from measurements. In image and video processing, a two-dimensional Cartesian grid is the most common way to store the image data that comes from video cameras or rendering systems.

### 2.3.2 Unstructured Grids

Unstructured grids are the most general grid type, as they have no predefined connectivity and do not impose any restrictions on the shape of the grid cells. In practical applications, however, these grids usually only consist of tetrahedra, hexahedra, prisms and pyramids – since the interpolation functions for these cells are well known and computationally feasible – but any other more complex form of cell type can be used.

Since unstructured grids have irregular topology, the connectivity information has to be stored explicitly. This requires more space and leads to larger data structures, but also results in more flexibility. Unstructured grids are spatially adaptive; this means that they can discretize domains of any shape and topology, by locally adapting the resolution – i.e., the size – or the shape of the grid cells.

Several variants of unstructured grids can be found especially in the context of finite-element-analysis. Among the most important types are the simplex grids. They consist only of the simplest $n$-dimensional primitives. In two dimensions, this is the triangle, in three dimensions the tetrahedron. In important feature of simplex grids is the possibility of linear interpolation within the cells, which makes them a very interesting basis for many visualization techniques.

Along the simplex grids, there are unstructured grids in use that consist of different basic primitives – as illustrated on the right side of Figure 2.3 – also known as *zoo-grids*. For the purpose of simulation the different cell shapes bear several advantages. However, for visualization the cells are usually decomposed into the corresponding simplices.

### 2.3.3 Point Clouds

A point cloud is a set $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subsetneq \mathbb{R}^D$ of data points (possibly acquired by a 3D scanning device), where $N$ is the number of points and $D$ the dimension of the domain. Point based data processing is receiving a growing amount of attention in computer graphics, as the point is a rendering primitive for direct visualization, without the need to generate triangle meshes for rendering.

However, besides this advantage, the biggest challenge for point based data visualization is to find an appropriate interpolation scheme, to obtain data values between the given points in a computationally feasible amount of time. Global interpolation methods are too expensive, as the influence of every single data point has to be considered. On the other hand, it is difficult to achieve a smooth global approximation with a local formulation. Beyond this, an efficient search for local neighbors of a given point implies that the point cloud is stored in a hierarchical data structure. Recently, the use of radial basis functions to reconstruct large data has been addressed by researchers [26], and will be used for multi-field visualization in Chapter 6. These functions are beneficial, since their interpolation scheme is known to converge exponentially and the fast evaluation enables interactive methods for point based volume visualization [64], as well as for meshless surface approximation [149; 150].

### 2.3.4 Grid Interpolation

Independent of the utilized grid, most visualization techniques require data values not only on the given data points, but also at arbitrary locations within the data domain. Estimating these intermediate data values and coordinates is the task of

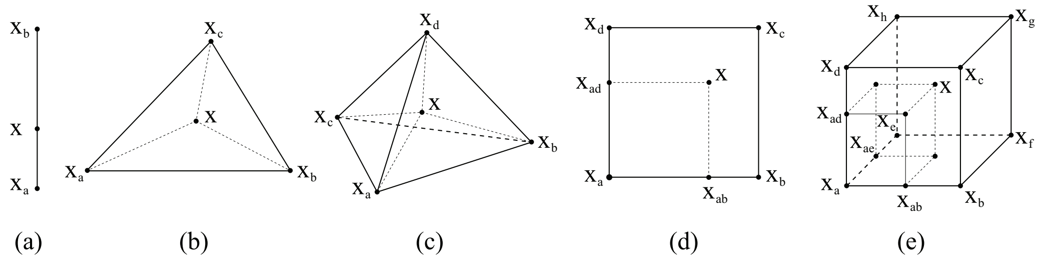


Figure 2.4: Interpolation schemes: (a) linear on a line segment; (b) linear in a triangle; (c) linear in a tetrahedron; (d) bilinear in a square; and (e) trilinear in a cube.

interpolation methods. This section introduces the most common interpolation schemes used in computer graphics and utilized in this work.

**Nearest-Neighbor Interpolation** The simplest interpolation method is *nearest-neighbor interpolation*, which uses the data point closest to the actual position to determine the new data value. As only one data point is used, the interpolated data value matches the value of the neighboring point. This causes nearest-neighbor interpolation to result in discontinuous transitions across cell boundaries and more accurate interpolation schemes have to be considered, if more precise and continuous results are desired.

**Linear Interpolation** Linear interpolation is a very useful continuous interpolation scheme, because it can be applied to any simplex grid cell. Figure 2.4 (a-c) illustrates linear interpolation for the most common simplices. In the one-dimensional case, if two scalar values $s(\mathbf{x_a})$ and $s(\mathbf{x_s})$ are given on two points $\mathbf{x_a}$ and $\mathbf{x_b}$ of a line segment. For any point $\mathbf{x}$ that lies on the line between $\mathbf{x_a}$ and $\mathbf{x_b}$, using the weight $\alpha \leq 1$, the linear interpolated of value $s(\mathbf{x})$ can be written as

$$s(\mathbf{x}) = (1-\alpha)\ s(\mathbf{x_a}) + \alpha\ s(\mathbf{x_b})\ , \quad \text{with} \quad \alpha = \frac{|\mathbf{x}-\mathbf{x_a}|}{|\mathbf{x_b}-\mathbf{x_a}|} \quad . \tag{2.1}$$

For the two-dimensional case, as depicted in Figure 2.4 (b), the value $s(\mathbf{x})$ at point $\mathbf{x}$ has to be interpolated within a triangle $\triangle\ \mathbf{x_a x_b x_c}$. This scheme which is also known as *barycentric interpolation*, is slightly different from linear interpolation between two points, but also works with a linear weighting of the values at the three given points to obtain the desired value at position $\mathbf{x}$. The interpolation function is defined by

$$s(\mathbf{x}) = \alpha\ s(\mathbf{x_a}) + \beta\ s(\mathbf{x_b}) + \gamma\ s(\mathbf{x_c}) \quad ,$$

where the weights $\alpha$, $\beta$ and $\gamma$, with $\alpha+\beta+\gamma = 1$, are the barycentric coordinates of $\mathbf{x}$ in $\triangle\ \mathbf{x_a x_b x_c}$. The weights are the surface areas of the sub-triangles

$\triangle\ \mathbf{x_a x_b x}$, $\triangle\ \mathbf{x_a x x_c}$ and $\triangle\ \mathbf{x x_b x_c}$, each divided by the surface area of $\triangle\ \mathbf{x_a x_b x_c}$ for normalization.

In a similar way, this can be extended to linear interpolation for a three-dimensional simplex. A scalar value $s(\mathbf{x})$ for any point $\mathbf{x}$ inside a $\boxtimes\ \mathbf{x_a x_b x_c x_d}$ tetrahedron may be computed as

$$s(\mathbf{x}) = \alpha\ s(\mathbf{x_a}) + \beta\ s(\mathbf{x_b}) + \gamma\ s(\mathbf{x_c}) + \delta\ s(\mathbf{x_d}) \quad ,$$

where the sum of $\alpha$, $\beta$, $\gamma$ and $\delta$ equals one. Analog to the two-dimensional case, in three dimensions, the weights are computed by the sub-volumes $\boxtimes\ \mathbf{x_a x_b x_c x}$, $\boxtimes\ \mathbf{x_a x_b x x_d}$, $\boxtimes\ \mathbf{x_a x x_c x_d}$ and $\boxtimes\ \mathbf{x x_b x_c x_d}$, divided by volume of the whole tetrahedron.

**Bilinear and Trilinear Interpolation** Linear interpolation can be easily extended for regular grids by repeatedly applying linear interpolation in each direction. In the two-dimensional case of a square, the product of interpolations is called *bilinear interpolation* and can be formulated as

$$s(\mathbf{x}) = (1-\alpha)(1-\beta)\ s(\mathbf{x_a}) + \alpha(1-\beta)\ s(\mathbf{x_b}) + \alpha\beta\ s(\mathbf{x_c}) + (1-\alpha)\beta s(\mathbf{x_d}) \quad ,$$

with

$$\alpha = \frac{|\mathbf{x_{ab}} - \mathbf{x_a}|}{|\mathbf{x_b} - \mathbf{x_a}|} \quad \text{and} \quad \beta = \frac{|\mathbf{x_{ad}} - \mathbf{x_a}|}{|\mathbf{x_d} - \mathbf{x_a}|} \quad .$$

Analogously, *trilinear interpolation* in a cube can be formulated as

$$\begin{array}{rlcll} s(\mathbf{x}) = & (1-\alpha)\ (1-\beta)\ (1-\gamma)\ s(\mathbf{x_a}) & + & \alpha\ (1-\beta)\ (1-\gamma)\ s(\mathbf{x_b}) & + \\ & \alpha\ \beta\ (1-\gamma)\ s(\mathbf{x_c}) & + & (1-\alpha)\ \beta\ (1-\gamma)\ s(\mathbf{x_d}) & + \\ & (1-\alpha)\ (1-\beta)\ \gamma\ s(\mathbf{x_e}) & + & \alpha\ (1-\beta)\ \gamma\ s(\mathbf{x_f}) & + \\ & \alpha\ \beta\ \gamma\ s(\mathbf{x_g}) & + & (1-\alpha)\ \beta\ \gamma\ s(\mathbf{x_h}) & , \end{array}$$

with

$$\alpha = \frac{|\mathbf{x_{ab}} - \mathbf{x_a}|}{|\mathbf{x_b} - \mathbf{x_a}|}, \quad \beta = \frac{|\mathbf{x_{ad}} - \mathbf{x_a}|}{|\mathbf{x_d} - \mathbf{x_a}|} \quad \text{and} \quad \gamma = \frac{|\mathbf{x_{ae}} - \mathbf{x_a}|}{|\mathbf{x_e} - \mathbf{x_a}|} \quad .$$

It should be noted that any bilinear and trilinear interpolation can be computed by successively applying linear interpolations. A second important characteristic of these interpolation schemes is the restriction to the vertices of a cell, which means that only the function values of the vertices of a specific cell are required, in order to interpolate the value of any point within this cell.

**Higher-order Interpolation** Especially in flow simulation or surface modeling, the reconstructed data is often a function of higher-order. In this case, piecewise linear interpolation might not be adequate and higher-order interpolation schemes, such as Lagrange interpolation or Hermite interpolation are required. For $n+1$ given data points, these methods can reconstruct a unique interpolation polynomial of degree $n$, bearing the advantage that these functions are continuous and differentiable. However, although there exist a multitude of sophisticated higher-order interpolation schemes, they are not utilized for the work in this thesis and thus are not discussed in more detail here. One reason for this is the fact that computing higher-order interpolation requires to solve a system of linear equations to determine the coefficients. This is computationally very expensive and slow and consequently not applicable, as real-time interactive visualizations require interpolation schemes that can be evaluated efficiently on the graphics hardware.

## 2.4 Introduction to Flow Visualization

A major part of this thesis concentrates on the visualization of flows or uses the information of motion to embed it in illustrations. In either case, the flow is described by a 2D or 3D field, where each grid point is assigned a velocity vector, indicating the direction of the flow and its speed. For a position $\mathbf{x}$ in 3D and at time $t$, this vector is defined by

$$\mathbf{v}(\mathbf{x},t) = \begin{pmatrix} u_1(\mathbf{x},t) \\ u_2(\mathbf{x},t) \\ u_3(\mathbf{x},t) \end{pmatrix} \quad . \tag{2.2}$$

### 2.4.1 Particle Based Techniques

Particle based methods are widely used in scientific flow visualization, as they can show the local motion of the flow for individual particles along their integral curves, as well as representing the global behavior of the flow with a set of traced particles. Particle traces form characteristic lines in a vector field; there exist *streamlines*, *pathlines*, *streaklines* and *timelines*, whereby only the first three are used in this work.

From a Lagrangian point of view a *streamline* corresponds to a line through the velocity field which is tangent to the velocity at every point, with the exception that $\mathbf{v}(\mathbf{x}) = 0$. It is a solution to the initial value problem of an ordinary differential equation

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{v}(\mathbf{x}(t)) \qquad , \qquad \mathbf{x}(0) = \mathbf{x}_0 \quad , \tag{2.3}$$

where $t$ is the parameter of the resulting curve $\mathbf{x}(t)$. A streamline can be considered as a snapshot of the velocity field at a fixed time step $t$.

A *pathline* is obtained by setting out a particle and tracing its path in unsteady flow. The trajectory of a massless particle injected into any time dependent velocity field $\mathbf{v}(\mathbf{x}, t)$ at position $\mathbf{x}_0$, is given by the Lagrangian formulation of the underlying equation of motion

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{v}(\mathbf{x}(t), t) \qquad , \qquad \mathbf{x}(0) = \mathbf{x}_0 \quad . \tag{2.4}$$

In experimental visualization, this can be achieved by long-term film exposure.

A *streakline* is the connection of all dye particles released from one seed location at different time steps. A streakline through a certain point $\mathbf{x}_0$ at time step $t_1$ is not uniquely defined, as another choice of $t$, such as $t_2$ might lead to a different streakline through $\mathbf{x}_0$. The tangent curve $\mathbf{x}(t)$ along $\mathbf{v}$ can be computed by integrating $\mathbf{x}(t)$ numerically. A common approach is to use Euler integration

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta t \mathbf{v}(\mathbf{x}_i) \quad , \tag{2.5}$$

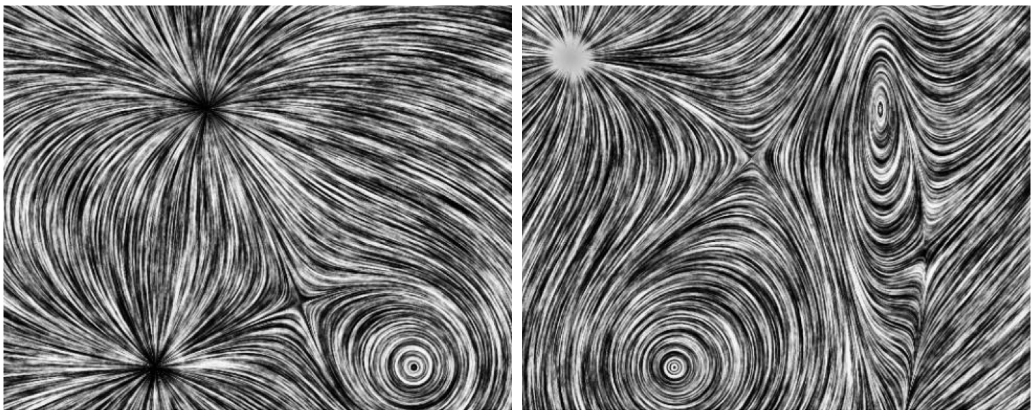
where $\mathbf{x}_i$ are positions along the particle trace, $\mathbf{v}$ is the flow field, and $\Delta t$ the integration step size. The tracing procedure can be described as follows. From a given starting point $\mathbf{x}_0$, which is the chosen particle seed point, a forward integration to obtain the next position of the particle is applied according to Equation (2.5).

### 2.4.2 Line Integral Convolution

Vector fields can be visualized in various ways. The particle based techniques presented in the previous section are restricted to a spatially coarse distribution of tracelines. In contrast, texture based methods aim to give a dense representation of the flow domain. Line integral convolution (LIC), as introduced by Cabral and Leedom [18] in 1993, is an effective method of visualizing flow fields with small scale structures. The algorithm filters an input texture with a one-dimensional convolution kernel along (curved) streamlines of a given vector field and generates a filtered texture as output, as shown in the images of Figure 2.5. In most cases, a texture with white noise serves as input, but pink noise or any other texture pattern can be used. The intensity $I(\mathbf{x}_0)$ at an arbitrary position $\mathbf{x}_0$ of the output image is given by

$$I(\mathbf{x}_0) = \int_{t_0 - t_l}^{t_0 + t_l} \kappa(t - t_0) T(\mathbf{x}(t)) dt \quad , \tag{2.6}$$

where $\kappa$ is the convolution kernel, $T$ the input texture and $\mathbf{x}(t)$ describes the parameterized streamline through $\mathbf{x}_0$ $(\mathbf{x}_0 = \mathbf{x}(t_0))$. Various kernel functions $\kappa$ can be used in the filter operation, whereby usually a symmetric kernel such as the Gaussian function is chosen. For the final intensity $I(\mathbf{x}_0)$ at position $\mathbf{x}_0$, the texture values along the segment $\mathbf{x}(t)$ of the streamline with $(t_0 - t_l \leq t \leq t_0 + t_l)$,



(a) (b)

Figure 2.5: Illustration of two flow datasets with the LIC method. Image (a) contains a source, a sink and a clockwise spinning vortex. Image (b) shows one source a clockwise and an anti-clockwise spinning vortex. The global illustration of the underlying flow fields even emphasizes the forming saddle points.

are weighted by the corresponding kernel values $\kappa(t - t_0)$. The filter operation can be approximated via a discrete convolution

$$I(\mathbf{x}_0) \cong \sum_{i=-t_l}^{t_l} \tilde{\kappa}(i\Delta t) T(\mathbf{x}(i\Delta t)) \quad , \tag{2.7}$$

with a discrete convolution kernel $\tilde{\kappa}$ with length $2t_l + 1$ and a step size $\Delta t$ along the streamlines. As a result of the convolution operation, the texture values average together to highly correlated, slowly varying intensities along individual streamlines of the field, but independent in directions perpendicular to the flow.

The LIC technique has been extended in various ways, e.g., oriented line integral convolution (OLIC) [160] which illustrates flow fields by convolving a sparse texture with an anisotropic convolution kernel; improvements and extensions to the spot noise technique have been presented [31]; as well as multi-frequency noise techniques [74]. Enhancements of LIC have been discussed for advanced color coding [136], time dependent animation [42; 67], or the application to unsteady flow [88; 137].

## 2.5 Introduction to Volume Visualization

Another major part of this thesis deals with multi-field datasets, containing various three-dimensional scalar data fields, usually rendered with a kind of volume visualization. This section presents the basic ideas of these volume rendering techniques. In this context, the term *volume visualization* specifies representation, manipulation and illustration of volumetric scalar data, whereby the process includes all stages of the visualization pipeline, i.e., filtering, mapping and rendering.

From a mathematical point of view, all volume data can be described as a function $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ that maps a three-dimensional data point to a scalar value. In practice, the data is often given on a three-dimensional Cartesian grid – called volume – where the uniform grid cells are referred to as *voxels*. Depending on the desired representation, either a surface representation or a volumetric illustration, we speak of *indirect volume visualization* or *direct volume visualization*. While the former extracts an isosurface to obtain a geometric representation of 2D primitives, the later evaluates the volume rendering integral that models physical effects of non-opaque material and enables a semi-transparent visualization of the whole domain. Both techniques are discussed in the following.

### 2.5.1 Indirect Volume Visualization

Indirect volume rendering techniques map the original volume data to an intermediate representation – such as geometric primitives – because this representation can be processed and visualized with less effort. However, in most cases this mapping includes loss of data and accuracy due to interpolation and approximation which is generally accepted, as parts of the whole data are not needed for visualization and can be neglected. For example, in medical illustrations, the reconstruction of an opaque bone structure with a volumetric rendering technique might be inefficient, when only the opaque hull of this structure needs to be visualized. In a volumetric dataset acquired by a medical imaging technique like CT or MRT, the material representing bone is given by a similar scalar value, forming a connected set or contour in the data. This contour can be reconstructed by computing an *isosurface* for a specified *isovalue*, i.e., the scalar that represents the bone.

One of the most established algorithms for extracting isosurfaces from volumetric data given on cuboid grid cells is the *Marching-Cubes* [90] algorithm proposed by Lorensen and Cline. The Marching-Cubes algorithm processes every grid cell and reconstructs for a given isovalue the corresponding isosurface in this cell, consisting of triangles and a normal on the vertices of each triangle. During this process, the scalar value of every vertex of a cell is tested against the isovalue and the vertex is classified as greater as or smaller than the value. From the resulting classification a index is generated, which represents one of the 256 possible triangle configurations that exist for an isosurface intersecting a cube. Due to symmetry, these cases can be reduced to 15 basic configurations as illustrated in Figure 2.6 and stored in a pre-computed indexed lookup table. Ambiguous cases can be eliminated by using the asymptotic decider as demonstrated in [108]. The vertices of the intersecting triangles are obtained by linear interpolation along the edges of the cube (cf. Equation (2.1)). In case normals

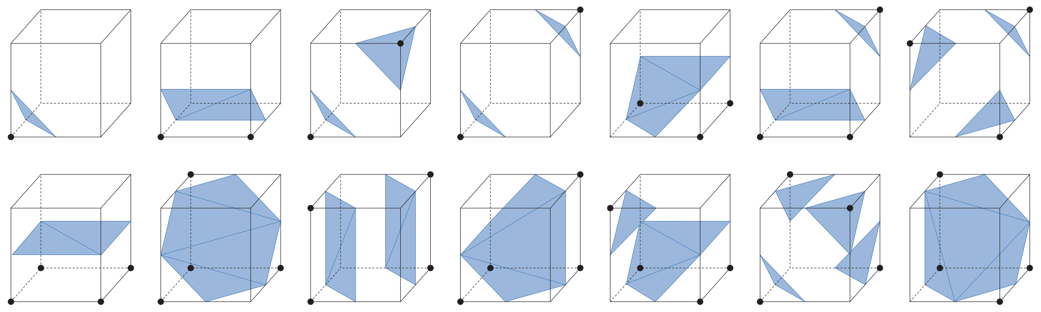


Figure 2.6: Marching-Cubes configurations. Note that the empty case is not illustrated.

are needed for advanced surface shading, the normal for each triangle vertex can be computed by linearly interpolating the vertex normals of the cubes' vertices. The latter are obtained, e.g., with the central differences method considering the vertex neighbors. The generated primitives can be efficiently processed "on-the-fly" by conventional graphics hardware up to a large number of primitives, which makes this method a preferable choice for many applications. A variant of the MC implementation – the *Marching-Tetrahedra* – is used in Section 3.5 to extract volume contours of scalar features in flow fields.

### 2.5.2 Direct Volume Visualization

While indirect volume visualization techniques work on an intermediate, lossy representation of the original dataset, direct volume rendering techniques use the original data for an instantaneous visualization. A general analytic description of the direct volume visualization process was presented in [82; 101]. The authors showed that all known volume rendering approaches can be generalized as specification of an underlying physical transport theory model for propagation of light in materials. For modeling the physical effects of light propagation through semi-transparent material, three characteristics need to be considered: absorption, emission and scattering. Hence, the equation of radiative transfer completely describes the radiation field in a participating medium, with respect to these three characteristics. The intensity radiated from a given point $\mathbf{x}$ in direction $\mathbf{v}$ is determined by the *radiance* $I(\mathbf{x}, \mathbf{v}, \nu)$, which is dependent of the frequency $\nu$.

Scattering of light is a complex process which changes both frequency $\nu$ and direction $\mathbf{v}$ of the radiant energy. Iterating along a ray and accumulating radiance $I$ is only valid if scattering can be completely neglected. Thus, volume rendering approaches in general use an *emission-absorption model* that ignores scattering effects.

### The Volume Rendering Integral

Lets consider a ray of light traveling along a direction $\mathbf{v}$ parameterized by a variable $s$. The ray enters the volume at position $s_0$. By suppressing the argument $\mathbf{v}$, the analytic solution of the emission-absorption equation of transfer over a line segment reads in integral form

$$I_1 = I_0 T(s_0, s_1) + \int_{s_0}^{s_1} \eta(s) T(s, s_1) ds \quad , \tag{2.8}$$

with *transparency* $T$ between two points $s_0$, $s_1$ and *emission* $\eta$, whereby

$$T(s_0, s_1) = e^{-\tau(s_0,s_1)} = e^{-\int_{s_0}^{s_1} \kappa(t)dt}, \qquad T \in [0,1] \quad .$$

Here $\tau(s_0, s_1) = \int_{s_0}^{s_1} \kappa(t)dt$ is the *optical depth*, with *absorption* $\kappa$, which defines how far a ray enters a material until it will be completely absorbed. As usual $I_0$ is given by the boundary conditions. The numerical solution of Equation (2.8) requires a discretization along the ray. The integration range is usually divided into $n$ intervals $s_k$. It is by no means necessary for the $s_k$ to be positioned equidistantly, though this is the most commonly used approach. The intensity $I$ at a discrete position $s_k$ can then be computed iteratively according to

$$I(s_k) = I(s_{k-1}) T(s_{k-1}, s_k) + \int_{s_{k-1}}^{s_k} \eta(s) T(s, s_k) ds \quad , \tag{2.9}$$

with the constants

$$\theta_k = T(s_{k-1}, s_k) \quad , \quad b_k = \int_{s_{k-1}}^{s_k} \eta(s) T(s, s_k) ds \quad \text{and} \quad b_0 = I(s_0) \quad ,$$

whereby the *absorption* term $\theta_k$ is called the *transparency* of the medium and the *emission* term $b_k$ describes the increase of radiant energy emitted within the range $[s_{k-1}, s_k]$ along the ray. If one knows something about the absorption and emission coefficients, e.g., that they behave like a polynomial of some degree, one can exploit this, which will accordingly result in a higher-order quadrature rule. However, most often in practice absorption and emission coefficients will be given as an array of data values at discrete points, and fairly little will be known about their functional form.

With these presumption, it is now possible to use an Eulerian sum for the evaluation of the integral in Equation (2.9), and assuming the source term and the attenuation factor for a certain segment $s_k$ as constants $b_k$ and $\theta_k$, then the integration is reduced to a finite sum over the accumulated opacity. Computing

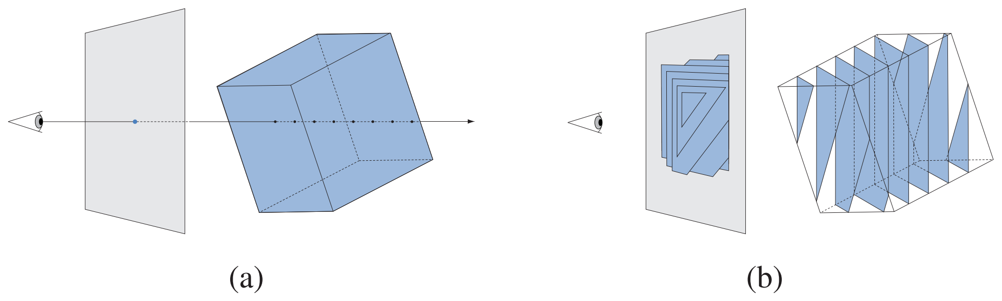


Figure 2.7: For Raycasting (a), one viewing ray is traced per pixel. The slice-based approach (b) accumulates all view aligned slices per pixel.

the exponential attenuation along the ray by using the first two terms of its Taylor expansion series, the final intensity for a ray segment $s_k$ is given by

$$I(s_k) = \sum_{i=0}^{n} b_i \prod_{j=0}^{i-1} \theta_i \quad . \tag{2.10}$$

This iterative approach is the fundamental equation for almost all methods of direct volume rendering, whereby the emitter $b_i$ is usually given by a color multiplied with its opacity $c_i\alpha_i$ and the attenuation term $\theta_i$ is given by $(1 - \alpha_i)$.

For these approaches the participating medium of a three-dimensional object is represented by a scalar value assigned to each discrete voxel. The physical properties of the scalars are given by a color and opacity value which are stored in a mapping table, the so called *transfer function*. To obtain the final intensity, the mapped voxel values are composited along the viewing direction for every pixel of the rendered image, evaluating Equation (2.10). Two of the most common approaches that have been used throughout this thesis are outlined here.

## Volume Raycasting

Raycasting is based on the idea of computing the final color of every pixel in the image by shooting rays – that originate in the viewers' eye – through the scene and solving Equation (2.10) (see Figure 2.7 (a)). Compositing the color and opacity for each pixel is done by evaluating discrete sampling points along the ray and applying the recursive *over* operator [115]. For raycasting, the compositing is commonly accomplished in *front-to-back* order, which results in

$$\begin{aligned} c_{out} &= c_{in} + (1 - \alpha_{in})\alpha_i c_i \quad , \\ \alpha_{out} &= \alpha_{in} + (1 - \alpha_{in})\alpha_i \quad , \end{aligned} \tag{2.11}$$

whereby $c_i$ and $\alpha_i$ are the pre-multiplied color contribution and opacity of the $i$-th sample, and $n$ is the total number of samples along the ray.

**Slice-based Volume Rendering**

The advent of hardware-accelerated texture mapping lead to slice-based volume rendering algorithms [17], which are in general slightly faster then raycasting. The slice-based approach renders a set of textured slices that are aligned parallel to the viewing plane (see Figure 2.7 (b)). The per pixel compositing of the volume samples from each slice is commonly performed with the *back-to-front* version of the *over* operator

$$c_{out} \quad = \quad (1 - \alpha_i) c_{in} + \alpha_i c_i \quad , \qquad i = 1, \cdots, n \quad , \tag{2.12}$$

which neglects the additional alpha computation, and thus, improves performance.

## 2.6 The Rendering Pipeline

The vantage point for hardware accelerated visualization are graphical primitives, such as points of a volume cloud, line segments of a particle trace or triangles of an isosurface, generated by the visualization algorithms described in the sections above. These primitives – which are commonly given as list of vertices – serve as input for the graphics hardware and are processed by the so-called *rendering pipeline* (see Figure 2.8) which generates a final two-dimensional rasterized image as output.

The vertices of the geometric primitives to be rendered are usually specified in a *local coordinate system*, which is often chosen in a way such that the center of the whole object coincides with the origin. For rendering, these vertices have to be combined into a single *world coordinate system*, which is independent of any viewer. In this world space, lighting computations can be performed and texture coordinates are specified. Then the vertices are transformed to *eye coordinate system*, which is defined by a viewing direction, an up vector and a right vector (implying a right-handed coordinate system). In this coordinate system culling is performed, since the visibility of the polygons depends on the line-of-sight vector and the normal and center of each polygon. The vertices in eye coordinates are still three-dimensional, they are projected to the viewing plane to obtain a two-dimensional representation of the scene in *clip coordinates* or *screen coordinates*. In this space, clipping is performed and hidden surfaces are removed in scenes with several opaque objects. The eye-space to screen-space transformation is non-linear and perspective division by the fourth component of the homogenous coordinates is applied to obtain *normalized device coordinates*. Finally, the last stage of the rendering pipeline consists of rasterizing the projected polygons. During this last step, shading, texturing and hidden surface removal are performed on a per per-fragment basis. In this context, a fragment can be regarded as a pixel carrying additional information like depth values and texture coordinates.

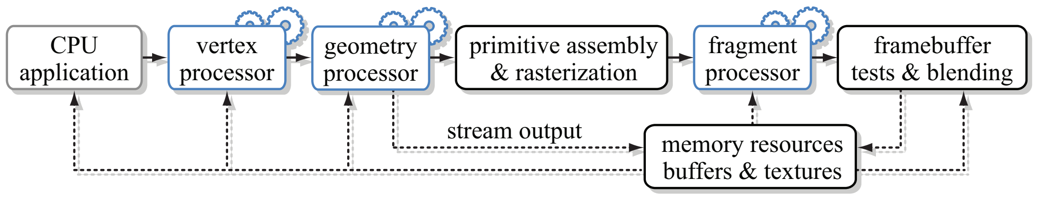

CPU application
vertex processor
geometry processor
primitive assembly & rasterization
fragment processor
framebuffer tests & blending
stream output
memory resources buffers & textures

Figure 2.8: The programmable rendering pipeline. Blue boxes indicate freely programmable units.

## 2.7 Graphics Processing Units

Originally the rendering pipeline was implemented on graphics hardware as a fixed-function pipeline and could not be controlled other than by changing the input data and manipulating various state variables to influence transformation, lighting and blending of polygons. With the introduction of new graphics processors, this rendering pipeline evolved to a freely programmable pipeline in recent years, allowing the programmer to write shader programs that define operations performed in the geometry processing stage or the fragment processing stage, as shown in Figure 2.8.

A *vertex shader* can be implemented to program the vertex processing unit with mathematical functions processing the vertex data, e.g., geometrical transformations, per-vertex lighting and computation of texture coordinates. This unit can only process existing geometry provided by the application or passed by the stream output from a later stage. In contrast, the lately introduced *geometry shader* can generate a limited amount of new geometry from existing geometry. In a general sense, the creation of new geometry is not a part of the actual rendering process and thus, this stage takes a special role but also counts to vertex or geometry processing. Further, the geometry shader has to be executed after the vertex shader and processes incoming vertices with attached adjacency information. The new vertices are passed as stream output to buffers in graphics memory and can be read back into the pipeline, to be processed in a subsequent rendering pass. The *pixel shader* is used to manipulate the rasterized fragments by means of color, or optionally more attributes such as depth and opacity. The pixel shader is executed for each rendered pixel, without any knowledge of the scene's geometry or neighboring pixels.

It is important to note that modern GPUs are based on the SIMD related stream processing paradigm that allows the application to exploit a highly parallel processing of the streamed data. In terms of stream processing, all shader programs are computation kernel functions that perform the same operation in parallel on all processed elements independently.

# Chapter 3 Visualization of Multi-field Flow Data

The visualization of vector fields has evolved to an important and interesting area of research, bearing a multitude of different visualization techniques [85] that have the ability of representing flow features in a local [68] and global [96] way, for steady [155] and unsteady [147] vector fields. Most of theses methods are based on the visualization of a single feature that can be extracted from a flow field, such as vortices, wakes [54] or shear layers [103]. All proposed vortex detection approaches employ flow attributes that describe a rotation or a swirling motion around a center of a region, leading to a line-based description [2; 114; 145], or a region-based description [3; 65; 87] of the vortex. Several other state-of-the-art feature extraction and visualization approaches can be found in [116]. For the purpose of analysis, not only features are of interest, but it is important to know about the reliability of the data and in particular of uncertain or erroneous regions. From the visualization point of view, the reliability or uncertainty as an attribute of the data can be understood as scalar feature too. A major part of research in this area has focused on representing uncertainty in simulation or analytical data, by extracting and manipulating geometry, e.g., isosurfaces with uncertainty [15; 70; 121], applying the uncertainty of surface boundaries to volume rendering [77], or manipulating the thickness of an isosurface [70]. Various types of glyphs for uncertainty in flow visualization have been used in [89; 170]. A classification of distinct possibilities of uncertainties – that can occur in a wide field of applications – and an extensible overview on uncertainty visualization techniques can be found in [48; 112]. The proper understanding of interaction between various processes and features in those fields is a pre-requisite for solving problems in engineering sciences. Many phenomena cannot be described by concentrating on them in isolation and thus, multi-field visualization concepts that include various kinds of field problems, as well as the reliability of the underlying process or data are needed. So far, only a few systems for efficient combination and visualization of high-dimensional features have been developed. Amongst others, these systems support a multi-field graph to visualize correlations in various scalar data [128], the selection of various features from a combined feature domain with a brush metaphor [32], or the post-classification of streamlines [126] that pass a region of multiple features by a Boolean function.

With the background of latest results proposed in literature, the work developed in this chapter describes enhanced multi-field flow visualization techniques, with a focus on uncertainty visualization in Sections 3.4.2 and 3.4.3, and combined feature visualization in Section 3.5. The uncertainty based visualization techniques extend 2D texture advection by an additional processing step. The first three methods are derived from a generic filtering process that is incorporated into the traditional texture advection pipeline. The first approach applies a cross advection perpendicular to the flow direction. The second method employs isotropic diffusion that can be implemented by Gaussian filtering. The third method adopts a multi frequency noise approach to apply pre-filtered input patterns to the advection process. These techniques are then enhanced by two color based uncertainty visualization methods, one that directly maps an uncertainty extent related color to the advected streaklines; and a second one that first detects uncertainty edges around streaklines, before mapping color to the edges only.

In the subsequent sections of this chapter, a flow feature visualization system is presented that uses first-order fuzzy logic (FOFL) for a combined visualization an analysis of flow features. The system allows the user to define and combine multiple feature criteria as logic point predicates and display the resulting characteristic set as isosurface with geometric primitives. For this purpose, the workflow of the system is subdivided into three layers: (i) feature definition and extraction; (ii) logical combination of feature sets; and (iii) visualization of the structures created by the feature sets in combination with the surrounding flow behavior. Combined feature visualization is an effective tool for handling large multi-field flow data, as the logical combination of features can be pre-computed, and only the resulting set needs to be processed during visualization.

The work described in Sections 3.4.2 and 3.4.3 is based on two publications [12; 13], and was carried out in collaboration with Daniel Weiskopf from the Universität Stuttgart, Germany. Andreas Lauser from the Universität Stuttgart, Germany was an active collaborator during the development of the combined visualization technique published in [11] and presented in Section 3.5. He must be credited for the implementation of the framework and the operators. Before detailing the methods introduced in these sections, an overview of computational fluid dynamics, different simulation and measuring techniques, as well as various flow feature classification methods is given.

## 3.1 Basic Ideas on Fluid Dynamics

Computational fluid dynamics has two major sub-disciplines, including hydrodynamics – the theory of fluids in motion; and aerodynamics – the theory of gases in motion. In terms of simplification, both will be summarized and called fluid below. The state of a fluid can be described by means of mathematical functions that are specified by five state variables, such as the velocity $\mathbf{v}(\mathbf{x}, t)$ and the thermodynamic variables pressure $p(\mathbf{x}, t)$ and density $\rho(\mathbf{x}, t)$, for a time $t$ at any location $\mathbf{x}$ in the fluid. In order to determine these five state variables, an appropriate differential equation that fulfills the conservation laws is required. From the principle of the conservation of mass, the *continuity* equation can be derived as

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0 \quad , \tag{3.1}$$

where the $\nabla$ is used as differential operator called “nabla”. In verbal terms this equation means that the amount of fluid flowing into a volume in space must be equal to the amount of fluid flowing out. Cauchy’s first law of motion can be deduced from the principle of conversation of *momentum* written as

$$\frac{\partial \mathbf{v}}{\partial t} = -(\mathbf{v} \cdot \nabla)\mathbf{v} - \frac{\nabla p}{\rho} + \mathbf{F} \quad ,$$

where $\mathbf{F}$ describes an external force, such as the gravitational acceleration $\mathbf{g}$. As this equation is only valid for perfect fluids without inner friction, such as non-viscous fluids, a term $\mu$ containing the viscosity coefficient needs to be introduced for viscous fluids. These considerations lead to the so-called Navier-Stokes equation for compressible fluids. This equation accounts for the motion of the fluid through space, along with any internal or external forces that act on the fluid and can be formulated as

$$\frac{\partial \mathbf{v}}{\partial t} = -(\mathbf{v} \cdot \nabla)\mathbf{v} - \frac{\nabla p}{\rho} + \mathbf{F} + \mu \nabla^2 \mathbf{v} + \frac{\mu}{3}\left(\nabla(\nabla \cdot \mathbf{v})\right) \quad , \tag{3.2}$$

where $\frac{\partial \mathbf{v}}{\partial t}$ is the derivative of velocity with respect to time and equals $-(\mathbf{v} \cdot \nabla)\mathbf{v}$ the convection term, $-\frac{\nabla p}{\rho}$ the pressure term, the external force $\mathbf{F}$, the viscosity term $\mu \nabla^2 \mathbf{v}$ and the heat transfer term $\frac{\mu}{3}\left(\nabla(\nabla \cdot \mathbf{v})\right)$. For incompressible fluids, such as they are used throughout this thesis, the later term can be neglected, as $\nabla \cdot \mathbf{v} = 0$. With specific initial conditions about the properties of the considered fluid – such as viscosity and heat transfer – and an equation of state that relates the pressure and the density of the fluid, it is possible to completely describe the flow behavior by solving the set of partial differential equations. Unfortunately, these equations have analytic solutions only for very simple flows and thus, for most problems numerical techniques as listed in Section 3.2 are applied. Usually this is done by dividing the simulation domain into a large number of cells and then, solving an algebraic version of the equations for each cell, considering the values in the neighboring cells.

## 3.2 Simulation and Measuring Techniques in a Nutshell

Flow data can be acquired by various ways of simulation or real-world measuring techniques. This section gives an overview of techniques that have been applied to obtain the data used to validate the multi-field flow visualization techniques presented in this thesis. All simulation techniques aim to solve the Navier-Stokes equations given in Equation (3.2) with different approaches for the turbulence model. Measuring systems try to reconstruct the behavior of the flow by capturing the movement of particles injected into the flow in relatively small time intervals.

**Direct Numerical Simulation**

DNS directly solves the Navier-Stokes momentum equations numerically without any turbulence model [130]. The used algorithms vary in the method that solves the equations, e.g., explicit or implicit techniques can be applied. For explicit methods, the time step has to be proportional to the spatial grid in order to keep the computation stable. Implicit methods allow for bigger time steps to improve computation performance. On the one hand, these simulations are capable of representing all temporal and spatial scales of turbulence; on the other hand, they require very large computational resources. Especially for turbulent flow with many small structures, DNS needs a high-resolution grid to conserve the kinetic energy, which leads to exceeding computational costs. Therefore, DNS is mainly applied to small grids or flows with relatively low Reynolds numbers.

**Reynolds-Averaged Navier-Stokes**

RANS methods solve the Reynolds equations, which are time-averaged Navier-Stokes equations [33]. These equations can be acquired by separating flow velocity into mean and fluctuating parts, introducing new terms known as Reynolds stresses. The momentum equations are coupled with appropriate conservation equations for the mass and the energy. Sometimes the energy equation is not needed. This occurs when the flow is incompressible, in thermal and chemical equilibrium (which is the case of low speed aerodynamic flows). In contrast to DNS, this approach can also be applied to flows in complex geometry and at high Reynolds numbers at very low computational costs and is therefore commonly used in CFD. However, RANS methods require the use of turbulence models that characterize all unsteady turbulent motion. Unfortunately, no turbulence model has been developed to this day that is capable of representing the turbulent motion accurately for a wide variety of flows. This limitation of RANS has led to increasing interest in other simulation techniques.

### Large-Eddy Simulation

LES is based on Kolmogorov's theory of self similarity [78], and was first proposed by Smagorinsky [138]. LES is capable of resolving the major part of the kinetic energy of turbulent motion, where only the small structures have to be modeled. The popularity of LES is based on the fact that the kinetic energy content of turbulent flow decreases with increasing wave number, whereby the major portion of the Reynolds stress can be solved. LES techniques solve space-filtered and/or time-filtered Navier-Stokes equations. Therefore, it is possible to directly compute the large scales (i.e. the large eddies) with less computational costs. The effects of the smaller flow structures have to be modeled by a sub-grid scale (SGS) model, since low-pass filtering introduces unknown quantities. The main advantage of LES is that the computational costs are substantially smaller than for DNS (but higher than for RANS), since not all scales have to be solved.

### Detached-Eddy Simulation

DES is a hybrid approach for the prediction of turbulent motions in flow fields with high Reynolds numbers [141]. It uses a single turbulence model that has the capacity of an SGS model in regions with a grid resolution fine enough for LES computation, and RANS in all other areas. More precisely, RANS is applied in zones where the turbulent length scale is less than the maximum grid dimension or near solid boundaries. Since the computational demands of pure LES increase significantly in the vicinity of walls, zonal approaches like DES are often employed to save computation costs. A weak point of those hybrid approaches is that RANS and LES have different requirements with respect to the underlying grid, and therefore, the success of the computation highly depends on the quality of the selected sub-grids for the different methods.

### Particle Image Velocimetry

Real-world fluid flow can be measured by particle image velocimetry (PIV) [59; 117]. The basic principle of PIV is to inject particles into the flow and to measure the movement of these particles between two light pulses. Usually, a planar laser light sheet technique is used. In very short intervals the target area is illuminated twice by a double-pulsed laser and recorded onto the CCD array of a digital camera (see Figure 3.1). Since the CCD chip must be able to capture each light pulse in separate image frames, the resolution in time is bound to the image frequency of the camera. Afterwards, appropriate algorithms evaluate consecutive images and determine the displacement of particles in the flow. The most common way of measuring displacement is to divide the image plane into small interrogation areas (IA) and cross correlate the images from the two time exposures. With this method, even unsteady or non-periodic flow fields can be measured.

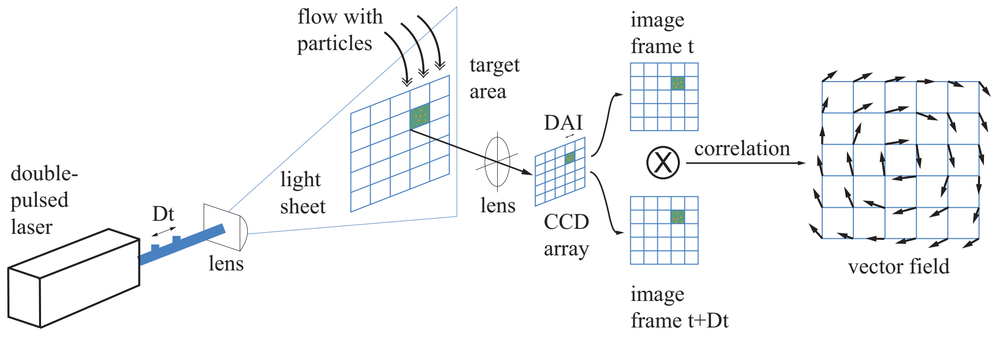


Figure 3.1: Configuration of a particle image velocimetry system.

One possible cross correlation algorithm works as follows: Each image is divided into $l$ small IAs with edge length $K$ (in pixels); the IAs are then shifted and compared with other parts of the image. For each translation $\Delta\mathbf{x}$ of one IA $A$ with $K^2$ pixels to another domain $B$ of the same size, the cross correlation

$$C(\Delta\mathbf{x}) = \sum_{i=1}^{K}\sum_{j=1}^{K} B_{ij} \cdot A_{ij}(\Delta\mathbf{x}) \tag{3.3}$$

can be computed. If the coefficient $C$ is a maximum and/or minimum for a translation $\Delta\mathbf{x}$ (depending on the matrix function), then the largest match between $A$ and $B$ and thus the shift of the particles has been found for IA $A$. Solving Equation (3.3) for all IAs $\Delta\mathbf{x}_l$ of the image results in a collection of displacement vectors that can be converted to velocity vectors by using the time difference between both evaluated images:

$$\mathbf{v}_l(\mathbf{x}_l) = \frac{\mathrm{d}\mathbf{x}_l}{\mathrm{d}t} \approx \frac{\Delta\mathbf{x}_l}{t_2 - t_1} \quad .$$

A problem of all real-world measuring systems is that during data acquisition errors can be introduced into the datasets from several sources [117]:

- Random error due to noise in the recorded images.
- Bias error arising from the process of computing the signal peak location to sub-pixel accuracy.
- Calibration inaccuracy of the CCD camera.
- Acceleration error caused by approximating the local Eulerian velocity from the Lagrangian motion of tracer particles.
- Gradient error resulting from rotation and deformation of the flow within an interrogation area leading to a loss of correlation.

- Dimension of the cross-correlated interrogation areas.
- Tracking error resulting from the inability of a particle to follow the flow without slip.

Some of these errors can be minimized by carefully choosing the experimental conditions, but others cannot be eliminated and thus, the final data inherits uncertainties by the nature of the measuring system. Since engineers are fully aware of the existence of those uncertainties, it can be of great advantage to include them to an interactive visual feedback during analysis.

## 3.3 Feature Classification

The introduced simulation and measuring techniques of section 3.2 can provide a plethora of multi-field data, including fields such as velocity, density, pressure or temperature of the fluid. Feature detection provides a powerful means of automatically identifying regions of interest by detecting analytically based features in computational fluid dynamics data (e.g., [58; 65; 91; 131]). In general, a feature is a prominent attribute or aspect that characterizes something of interest. As proposed by Peikert [114], a flow feature can be classified by its locality or dimensionality and thus, characterizes a set of points in a multivariate field, defined by a given feature criterion. For example, feature extraction methods can be based on criteria like vortex core lines [145], vortex core regions [2], ridge lines, valley lines, or separation lines [72], but also other properties like pressure, temperature and uncertain or erroneous data values [112] can be of interest for the analyst. This section gives an overview of classified feature criteria commonly used for the purpose of analyzing gaseous or fluid flow phenomena.

**Jacobian**

Most flow feature detection methods deal with the analysis of the flow behavior around the nearest neighborhood of a given point. For this analysis, the velocity gradient tensor (also known as *Jacobian* $J$) plays an important role, as it consists of the partial derivatives that give information about the extent of change at a certain location in a vector field. If $\mathbf{v}$ represents a velocity field of a moving fluid, then $J$ represents the velocity gradient tensor

$$J = \begin{pmatrix} \frac{\partial \mathbf{v}_x}{\partial x} & \frac{\partial \mathbf{v}_x}{\partial y} & \frac{\partial \mathbf{v}_x}{\partial z} \\ \frac{\partial \mathbf{v}_y}{\partial x} & \frac{\partial \mathbf{v}_y}{\partial y} & \frac{\partial \mathbf{v}_y}{\partial z} \\ \frac{\partial \mathbf{v}_z}{\partial x} & \frac{\partial \mathbf{v}_z}{\partial y} & \frac{\partial \mathbf{v}_z}{\partial z} \end{pmatrix} . \tag{3.4}$$

The velocity gradient tensor is made of a symmetric and a antisymmetric part, the deformation tensor $S_{ij}$ and the rigid body rotation tensor $\Omega_{ij}$ (see Equation (3.7) which is also known as vorticity tensor. The determinant of the Jacobian matrix represents the transformation of one volume unit from one coordinate space to another. Its importance is due to the fact that it represents the best linear approximation to a differentiable function near a given point and it is used to extract and classify a number of features from flow fields.

**Vorticity**

Vorticity is related to the local angular rate of rotation and can be understood as the curl of the fluid velocity at a point. Mathematically, the vorticity is a pseudo vector defined by the direction of the axis of rotation and the magnitude of the rotation itself. Given the Jacobian, the vorticity can be easily extracted from the rotational part $\Omega$ of the matrix [54], and can be written as

$$\omega = \nabla \times \mathbf{v} \quad . \tag{3.5}$$

Vorticity can also be considered as the circulation per unit area at a point in a fluid flow field. A vector field which has zero curl everywhere is called irrotational, which is always true, if the vector $\mathbf{v}$ represents the gradient of a scalar field.

**Helicity**

By computing the helicity of a velocity field, we can examine the potential for helical flow, or flow that appears to move in a corkscrew pattern. Helicity is computed using Equation (3.6), and physically represents the curl in the direction of the velocity field as

$$h = (\nabla \times \mathbf{v}) \cdot \mathbf{v} \quad . \tag{3.6}$$

If the fluid moves in a dominant stream wise direction, then helicity looks similar to vorticity. However, if the flow is not dominated by a single direction, then the helicity will provide interesting and different results than those obtained by computing and analyzing just the vorticity of the field.

**$\lambda_2$ Vortices**

Vortices are among the most relevant features in a flow for many applications. The $\lambda_2$ method was developed by Jeong and Hussain [65], to identify vortical regions, as they realized that intuitive measures such as vorticity or pressure-minima are not always adequate indicators. This unreliability is caused by two physical effects. First, viscous effects can eliminate pressure minima in vortical regions and second, an unsteady straining can result in pressure minima in non-vortical regions. The first effect causes false negatives and the second yields false positives, both misleading an analysis. To discard these effects of unsteady irrotational straining and viscous effects caused in conjunction with false pressure evaluation, the $\lambda_2$-method depends on the velocity gradient tensor of Equation (3.4) and simply neglects these terms in the derivation of the gradient taken from the Navier-Stokes equations. In the end, the $\lambda_2$ vortex detection problem is computed by first decomposing the Jacobian matrix of the velocity field – i.e., the velocity gradient tensor $\mathbf{v}_{i,j}$ – into the symmetric part $S$ and the antisymmetric part $\Omega$

$$\begin{aligned} \mathbf{v}_{i,j} &= \frac{1}{2}(\mathbf{v}_{i,j} + \mathbf{v}_{j,i}) + \frac{1}{2}(\mathbf{v}_{i,j} - \mathbf{v}_{j,i}) \\ &= S_{ij} + \Omega_{ij} \ , \end{aligned} \tag{3.7}$$

where $S = \frac{J+J^T}{2}$ , $\Omega = \frac{J-J^T}{2}$ and $J = \nabla\mathbf{v}$ . From a physical point of view, $S$ is the strain-rate tensor and $\Omega$ the vorticity tensor. This reduces the problem in the next step to an eigenvalue analysis of the matrix $S^2 + \Omega^2$. Since this matrix is real and symmetric, it results in three eigenvalues — the roots of the characteristic polynomial — denoted by $\lambda_1 \geq \lambda_2 \geq \lambda_3$. A $\lambda_2$ vortex is then defined as a connected region where two of the eigenvalues are negative, or equivalently, where $\lambda_2 < 0$. Since this classification results in a whole region the $\lambda_2$ criterion is rather fuzzy than binary.

### Shear Layers

Boundary layers or shear layers are important, since they embody two kinds of features that can lead to meaningful conclusions. On the one hand, they tell that the fluid is under shear stress and on the other hand, they also indicate the generation of vorticity. This suggests a marker that is a function of both $\Omega$ and shear. As shown in [54], the stress tensor contains both the bulk and the shear stress and is independent from the coordinate system. To extract a single scalar that is coordinate system invariant and has the bulk terms removed, it is necessary to diagonalize this tensor. The result is always three real eigenvalues $(\lambda_{s1}, \lambda_{s2}, \lambda_{s3})$ that produce a vector which signifies the principle axis of deformation [54]. The norm of the second principal invariant of the stress deviator can be used as a measure of the shear. This measure is denoted as

$$\lambda_s = \sqrt{\frac{(\lambda_{s1} - \lambda_{s2})^2 + (\lambda_{s1} - \lambda_{s3})^2 + (\lambda_{s2} - \lambda_{s3})^2}{6}} \quad . \tag{3.8}$$

A scalar shear field $s_H$ can then be constructed from a function of $|\Omega|$ and $\lambda_s$. Using only the two strongest eigenvalues empirically gives better results in two dimensions. Therefore Equation (3.8) reduces to $\lambda_s = \lambda_{s1} - \lambda_{s2}$.

### Shock Detection

Shocks represent a large class of features which more broadly are represented by connected regions of sharp discontinuities. The potential shock regions can be identified and further classified by methods first defined in [97]. The quantities are calculated by

$$E_1 = min(\tfrac{\mathbf{v}}{|\mathbf{v}|} \cdot \nabla\xi, 0) \ , \ E_2 = max(\tfrac{\mathbf{v}}{|\mathbf{v}|} \cdot \nabla\xi, 0) \ , \ E_3 = \left|\nabla\xi - \tfrac{\mathbf{v}}{|\mathbf{v}|}\mathbf{v} \cdot \nabla\xi\right| \ , \tag{3.9}$$

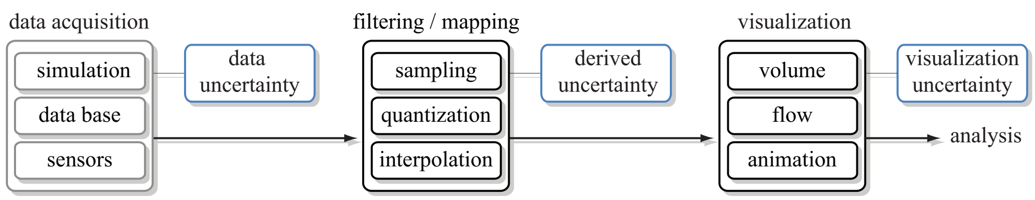


Figure 3.2: Possible sources for uncertainties from data acquisition to visualization.

where $\mathbf{v}$ denotes the velocity field and $\xi$ can represent either pressure $p$, density $\rho$, or Mach number $M$. If $\xi = p$ and $E_2 > 0$, then the point is part of a compression shock, otherwise if $E_1 < 0$, the point is part of an expansion shock. If $\xi = \rho$ and $E_2 > 0$, then the point is part of an expansion shock, otherwise if $E_1 < 0$, the point is part of a compression shock. If $\xi = M$ and $E_2 > 0$, then the point is part of an expansion shock, otherwise if $E_1 < 0$, the point is part of a compression shock. The value $E_3$ represents shear shock orthogonal to the flow direction.

### Scalar Attributes and Derived Quantities

Although scientist and engineers are primarily interested in the overall flow behavior, or local occurring vortical regions or shear layers, depending on the requirements of the application domain other attributes can be of importance. The analysis of various fields simultaneously can include various attributes such as pressure $p$ or density $\rho$ and derived quantities like velocity magnitude $\|\mathbf{v}\|$, vorticity magnitude $\|\omega\|$ or the divergence $\text{div}(\mathbf{v}) = \nabla \cdot \mathbf{v}$ of a vector field.

### Uncertainty

For an efficient understanding of the visualized data, it is important to indicate how accurate and reliable the provided data and thus the visualization is. As shown in the three stage pipeline of Figure 3.2 (adopted from [89; 112]), several different uncertainties or errors can be introduced to the data on the way from data acquisition to the stage of visualization. In the visualization step, algorithmic uncertainties can occur, e.g., by approximating factors for global illumination or by interpolating data values on slices in volume rendering. Transformation uncertainties are introduced by converting from one unit to another or by scaling, resampling, or quantization. In this thesis however, the focus lies on uncertainties that appear in the first stage — during data acquisition, as this stage is most interesting for uncertainty visualization, because the errors from data acquisition typically cannot be influenced or reduced by subsequent processes.

In this context, raw data from real-world measurements such as the PIV method as detailed in Section 3.2, is one possible basis to compute an error value, as they intrinsically inhere measuring errors. The utilized error measure is typically based on the root mean square. For the PIV method this yields $N$ measurements

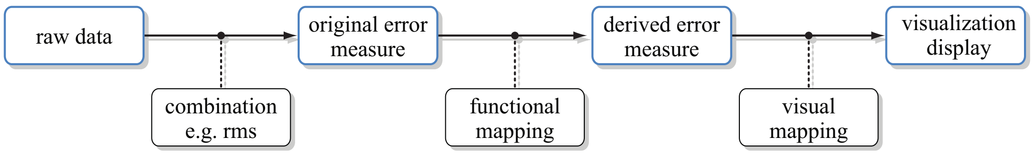


Figure 3.3: Pipeline for visualizing uncertainties. (Adopted from Pang *et al.* [112]).

for the vector field at each spatial location and time step. The raw data measurements are denoted $\mathbf{v}_i$ with $i = 1 \ldots N$. The average vector $\mathbf{v}$ serves as basis for traditional vector field visualization. Then, the root mean square

$$r_{\text{rms}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} ||\mathbf{v}_i - \mathbf{v}||^2} \quad ,$$

can be used as the measure of uncertainty.

However, any other error computation could be used as uncertainty measure, e.g., for various simulation methods it is of interest to which extent the results of each simulation differ. In this case, a possible measure for uncertainty is the magnitude of the difference vector of both methods, computed for each grid cell of the field. This yields

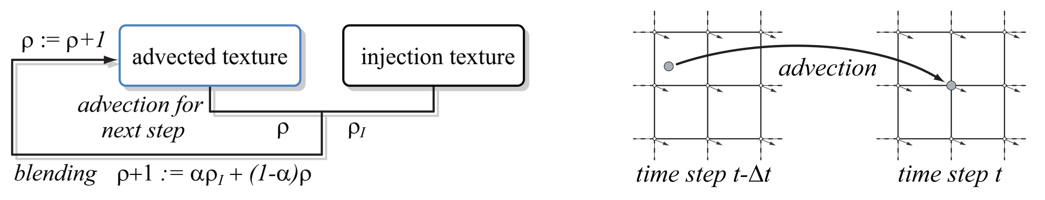
$$d = ||\mathbf{v_1} - \mathbf{v_2}|| \quad .$$

It should be noted that for visualization the original error value needs to be mapped to a derived error measure, e.g., by a linear or a non-linear function, in order to obtain values in a useful range. This possibly space-variant and time-dependent uncertainty measure is denoted by $u(\mathbf{x}, t)$.

## 3.4 Strategies for Uncertainty Visualization

The goal of this section is to enhance texture advection for the visualization of multi-field flow data. In this particular case, the basic technique is enriched by a visualization of flow uncertainty, without losing the benefits of texture advection, i.e., its flexibility and efficiency. From an abstract point of view, uncertainty visualization can be structured into a three-stage process (see Figure 3.3). The visualization essentially requires the ability to represent one additional single-valued attribute: a measure for uncertainty or error.

Various means of a visual mapping of multi-variate data could be employed to represent the error attribute. A simple and well-known method is to map the derived error value to color and to overlay this color on top of the underlying flow visualization. However, this kind of visualization method only shows uncertainty at a respective point, i.e., it results in a localized representation. In contrast, uncertainty in a flow leads to an uncertainty of particle transport, which should also



(a) (b)

Figure 3.4: (a) pipeline for texture based flow visualization, (b) shows the advection step.

be represented by means of “uncertain” particle traces. It is reasonable to mark a particle that has been advected throughout an error-affected area and to emphasize this in the further process of the flow. Furthermore, an uncertainty-affected region can influence its neighborhood and this should be taken into account in the visualization step. Therefore, the goal is to incorporate the uncertainty representation within the concept of texture-based particle transport. Another advantage of this approach for the first three of the proposed techniques is that the error attribute is encoded in the same perceptual channel as the original flow—in the form of a texture—and, thus, other perceptual channels (e.g., color) could be used to encode additional attributes. As the following approaches of uncertainty flow visualization are based on texture advection, the basic principles of texture based flow visualization are reviewed first in the subsequent section.

### 3.4.1 Texture-based Flow Visualization

Texture advection is a well-established and versatile method for visualizing steady and unsteady flow and has been strongly advanced lately, not only because of the rapidly increasing performance and functionality of GPUs [132; 167]. It is directly related to Line Integral Convolution (LIC) [18] introduced in Section 2.4.2, and most examples of texture advection can be considered as LIC with an exponential filter kernel. The availability of dense noise-based and sparse dye-based representations, as well as the possibility for the user to interact and manipulate all important parameters on-the-fly makes it a powerful tool for interactive visualization. Early versions of texture advection [102] were extended to 2D Lagrangian-Eulerian Advection (LEA) [66], 2D Image Based Flow Visualization (IBFV) [155], or GPU-based texture advection [166] of dense particles and for a sparse representation as done with the metaphor of dye advection [162; 163]. Semi-Lagrangian transport [66; 142], which is the basis for the later implementation of texture advection for uncertainty visualization, is briefly described in this section. For a deeper insight on state-of-the-art texture-based flow visualization the reader is referred to [85].

Particles or injected dye are represented on a regularly sampled grid or texture. This property field is denoted by $\varrho(\mathbf{x})$. The points $\mathbf{x}$ are from the domain of the

$n$D vector field, $\mathbb{R}^n$. For the Eulerian approach, particles lose their individuality and their position is implicitly given by the location of the corresponding texel in the property field. Particles are transported along streamlines for steady, or along pathlines for unsteady vector fields $\mathbf{v}(\mathbf{x}, t)$, where $t$ denotes time. The Lagrangian formulation of motion given in Equation (2.4), can be integrated to compute the pathline of an advected massless particle,

$$\mathbf{x}(t_1) = \mathbf{x}(t_0) + \int_{t_0}^{t_1} \mathbf{v}(\mathbf{x}(t), t)\, dt \quad . \tag{3.10}$$

The evolution of the property field $\varrho(\mathbf{x}, t)$ is governed by

$$\frac{\partial \varrho(\mathbf{x}, t)}{\partial t} + \mathbf{v}(\mathbf{x}, t) \cdot \nabla \varrho(\mathbf{x}, t) = 0 \quad .$$

This partial differential equation can be solved by semi-Lagrangian transport [66; 142], which leads to a stable evolution even for large step sizes. A backward texture lookup is often employed:

$$\varrho(\mathbf{x}(t_0), t_0) = \varrho(\mathbf{x}(t_0 - \Delta t), t_0 - \Delta t) \quad . \tag{3.11}$$

Starting from the current time step $t_0$, an integration backwards in time according to Equation (3.10) provides the position at the previous time step, $\mathbf{x}(t_0 - \Delta t)$. Texture-based methods often produce only short streamlines or streaklines and, therefore, first-order Euler integration typically provides sufficient accuracy:

$$\mathbf{x}(t_0 - \Delta t) = \mathbf{x}(t_0) - \Delta t \mathbf{v}(\mathbf{x}(t_0), t_0) \quad .$$

The property field is evaluated at this previous position to access the particle that is transported to the current position. Tensor-product linear interpolation (bilinear in 2D or trilinear in 3D) is applied to reconstruct the property field at locations different from grid points.

One advantage of texture advection is that it can be used to visualize larger structures in the flow, like streamlines in steady flow or streaklines in time-varying flow. However, sequentially applying the advection step to an input texture leads to a heavy deformation of the patterns in this texture according to the underlying flow field, but it does not reveal the desired clear structures of the flow, such as streamlines. To achieve this, these structures can be generated by injecting smooth dye patterns or noise-based particles into the property field $\varrho$ after each advection step, following the IBFV approach [155]. According to Figure 3.4, in the accumulation step newly injected particles from the input texture $\varrho_I$ are combined with the advected particles in the property field $\varrho$. The accumulation is done by means of a blending function, e.g., alpha blending as defined in Equation (2.12) can be applied as

$$I(\varrho(\mathbf{x}, t+1)) = \alpha I(\varrho_I(\mathbf{x}, t)) + (1-\alpha) I(\varrho(\mathbf{x}, t)) \quad ,$$

which represents the discretized version of an exponential filter kernel [38] in the context of LIC [18]. Texture advection is highly applicable for uncertainty visualization because it offers important benefits. First, the injection scheme is flexible in allowing the user to gradually change the density of new particles from a few randomly injected particles, up to a densely filled property field represented by white noise. Second, texture advection lends itself to a direct and efficient mapping to GPUs, which leads to an interactive visualization of large datasets.

### 3.4.2 Texture-Based Uncertainty Visualization

Texture advection as described in the previous section is capable of showing the vector field of the flow (i.e. a single vector at each data point), but needs to be extended to visualize additional information. The goal of this section is to enhance texture advection for the visualization of multi-field flow data. In this particular case, the basic technique is enriched by a visualization of one more scalar attribute, such as uncertainty or error, without losing the benefits of texture advection, i.e., its flexibility and efficiency.

The original uncertainty that is described by a scalar value may be mapped to a derived measure, e.g., by a linear or a non-linear function, in order to emphasize uncertainty ranges and obtain useful values. The result of this mapping is denoted by the uncertainty measure $u(\mathbf{x}, t)$. Uncertainty visualization addresses the display of $u(\mathbf{x}, t)$. The crucial observation is that uncertainty of the vector data results in an uncertainty of pathlines. Therefore, the particle transport should take into account $u(\mathbf{x}, t)$ by modifying the advection process. In fact, the pipeline from Figure 3.4 is just extended by one additional image-filtering stage that works on the advected texture, but is completely decoupled from the texture advection computation. The first steps of the visualization cycle ("load flow data", "texture advection", and "particle and dye injection") are identical to traditional texture advection. The additional error filtering stage can be embedded into this traditional pipeline without any major drawback, as shown in Figure 3.5. Error filtering aims at manipulating the spatial frequency perpendicular to particle traces to show uncertainty $u(\mathbf{x}, t)$, i.e., not only the spatial frequency along the flow changes as in LIC [18], or in basic texture advection [85]. But this uncertainty filter affects the advected property field $\varrho$ according to the following 2D convolution:

$$\varrho_{\text{filtered}}(\mathbf{x}) = \int\limits_{V(\mathbf{x},\mathbf{v})} f(u, \tilde{\mathbf{x}}, \mathbf{v}) \varrho(\mathbf{x} + \tilde{\mathbf{x}}) \, d^n \tilde{x} \quad . \tag{3.12}$$

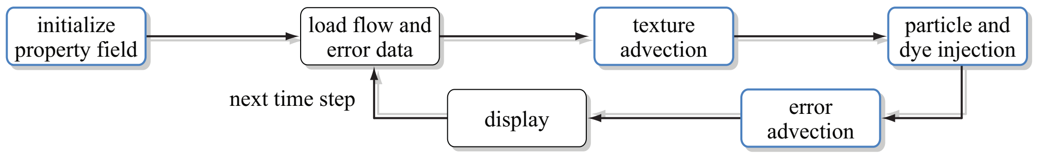


Figure 3.5: Flowchart of the texture advection pipeline extended extended by an error advection step.

In contrast to many other convolution filters, the filter $f$ may be space-variant and flow-dependent, and so is the compact $n$D integration domain $V(\mathbf{x}, \mathbf{v})$, at which the dimension is restricted to $n = 2$ for visualization. The integral is evaluated at a fixed time $t$. The goal of the convolution filter is to modify the spatial frequency perpendicular to particle traces: it essentially smears out particle traces. While LIC or traditional texture advection only modify the spatial frequencies along the flow, uncertainty-aware texture advection also affects the spatial frequencies perpendicular. The specific character of the uncertainty visualization is controlled by the choice of filter kernel and integration domain. This class of filtering can be understood as post-processing. Two different approaches for post-processing uncertainty filters, i.e., *cross advection* and *error diffusion* and a third technique called *multi-frequency noise* which implements pre-processed filtering on the injection texture are described below.

***Cross Advection*** The first specific example for error filtering is the *cross advection* approach. Here, the filter domain is reduced to a line perpendicular to the current flow direction. The fundamental idea is that particles are additionally transported perpendicular to the flow direction, smearing out the streakline of the particle trace. This essentially leads to a 1D convolution similar to that of traditional texture advection. The only difference is that a symmetric filter (in both directions) is applied. The discretized version of the filter reads

$$\varrho_{\text{filtered}} = \sum_{i \in \{-1,0,1\}} \kappa_i \varrho_i \quad , \tag{3.13}$$

where $\kappa_i$ is the discrete filter kernel and $\varrho_i$ are samples of the property field along the perpendicular line,

$$\varrho_i = \varrho(\mathbf{x} + i\mu\Delta t M_{\text{rot}} \mathbf{v}(\mathbf{x}, t)) \quad . \tag{3.14}$$

Here, $\mu$ denotes the error-depending relative step size, $\Delta t$ is the step size of traditional texture advection, and $M_{\text{rot}}$ is a 2D matrix for a rotation by 90 degrees. The integration width and, thus, the length scale for smearing is controlled by the

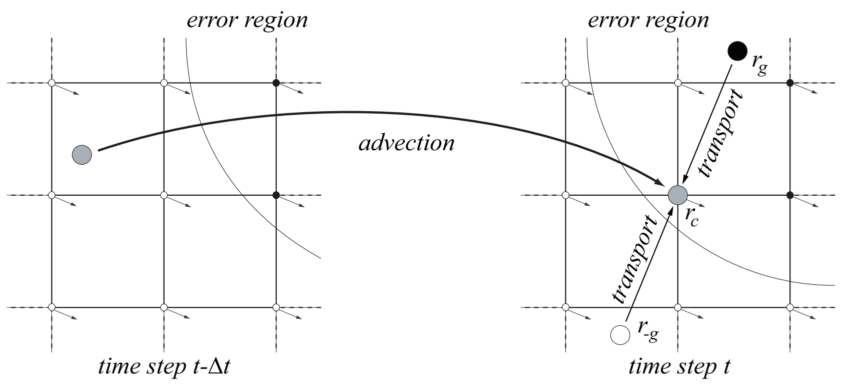


Figure 3.6: Semi-Lagrangian transport of a particle along flow direction into an error region from time step $t - \Delta t$ to time step $t$. The cross advection is applied in time step $t$.

error value via $\mu$. Equation (3.13) implements the analog of line integral convolution based on texture advection, with the following differences: First, a convolution perpendicular to the flow direction is performed; second, a symmetric convolution filter in both directions is applied. This filtering maintains a constant overall brightness if a normalized kernel is used (i.e., $\sum \kappa_i = 1$). A typical choice for this 1D filter kernel is $\kappa_i = 0.25, 0.5, 0.25$.

Figure 3.6 illustrates the two steps performed for the complete cross advection approach. In the first step, a particle (grey dot) is transported by traditional semi-Lagrangian advection along the flow direction into an uncertainty or error affected region. Black grid points lie inside the error region, white grid points outside. In the next step shown on the right side of Figure 3.6, the intensity values are computed for the current point (grey dot) and the two cross directional points (white and black dots), by combining the values according to Equation (3.13).

The filtering process can be slightly modified by replacing the weighted sum in Equation (3.13) by a maximum function according to

$$\varrho_{\text{filtered}} = \max\{\varrho_i | i = 1, 2, 3\} \quad . \tag{3.15}$$

The maximum function avoids that (sparsely seeded) streaklines are not reduced to very small property values in regions of large uncertainty, and therefore smeared-out streaklines are maintained. In general, the $\max()$ function does not provide a constant overall brightness but tends to increase the image brightness. Therefore, this variant is primarily designed for sparse representations with a few, clearly separated streaklines.

The left image of Figure 3.7 shows an example of a single dye pattern injected into a uniform flow field. The underlying uncertainty begins a few steps away from the injection point and then stays constant. The exact shape of the streakline is visible in the error-free part of the flow that changes to a symmetric expansion

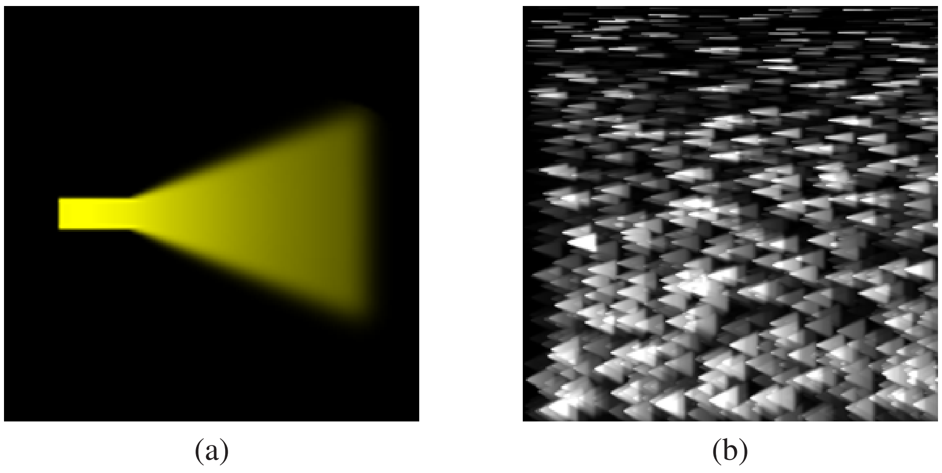


Figure 3.7: Image (a): Cross advection with a single dye pattern in a laminar flow field $\mathbf{v}(x, y) = (1, 0)$ and with piecewise constant uncertainty. Image (b): Sparsely injected particles into the same flow with increasing uncertainty from top to bottom. (Images courtesy of Botchen *et al.* [12], ©2005 IEEE).

of both sides. The right image illustrates a sparse injection of random particles, while the uncertainty increases from top to bottom. Since uncertainty magnitude affects the step size of cross transport, streaklines in regions with large errors (bottom) spread more than those in unaffected regions (top).

Cross advection can be considered an image-processing operation that can be directly mapped to GPU fragment operations. The update equations (3.13) or (3.15) work independently on 2D grid cells of the property field. By identifying this 2D grid with a texture, the update of cells (i.e., texels) is reduced to updating the underlying texture through a fragment program. In fact, ping-pong rendering is employed to update textures: Two copies of the property field are held in texture memory; one serves as render target, the other one serves as input texture. After each update, the role of the two textures is exchanged. This cross advection process is one example for an “error advection module” in Figure 3.5.

The main part of the HLSL fragment program of the basic transport mechanism for Equation (3.15) is given in Listing 3.1. The mapping from the input uncertainty measure to the relative step size $\mu$ is implemented by a dependent-texture lookup. The space-variant input uncertainty is held in a 2D domain-filling texture. Three texture lookups in the property texture are performed to evaluate Equation (3.14). In the 2D case, the rotation matrix $M_{\text{rot}}$ can be realized by a component swizzle followed by a simple multiplication. Finally, the texel with maximum intensity is written to the output. In a variant of this fragment program, the $\max()$ function is replaced by a weighted sum to implement Equation (3.13).

```
float4 dir    = tex2D(VecField, TxCoord);   // lookup vector at pos TxCoord
float4 thisTx = tex2D(AdvTx, TxCoord);      // result of previous advection
float  error  = tex2D(ErrField, TxCoord);   // uncertainty measure u(x,t)
float  stSize = tex2D(ErrStSize, error);    // error step size: mu * Delta t
float4 maxintens;

dir.yx        = dir.xy * stSize.xx;         // perform cross advection
dir.x        *= -1.0f;                      // rotate -90
newpos.xy     = TxCoords - dir.xy;
float4 leftTx = tex2D(AdvTx, newpos.xy);
newpos.xy     = TxCoord + dir.xy;           // rotate 180
float4 righTx = tex2D(AdvTx, newpos.xy);

maxintens.x   = max(leftTx.x, rightTx.x);   // find maximum intensity
maxintens.x   = max(thisTx.x, maxintens.x);

Output.rgba   = maxintens.xxxx;             // final output
return Output;
```

Listing 3.1: Main part of the HLSL fragment program for the cross advection approach. (Code courtesy of Botchen *et al.* [12], ©2005 IEEE).

***Error Diffusion*** The second technique implementing the generic filtering process from Equation (3.12) is called *error diffusion*, which is an alternative example of uncertainty visualization. In contrast to cross advection, error diffusion applies an isotropic 2D filter kernel, independent of the direction of the flow. Filtering is space-variant: The amount of smearing is controlled by the uncertainty $u(\mathbf{x}, t)$, which determines the width (i.e. size) of the filter kernel. The larger the uncertainty, the wider the kernel. Typically, a normalized 2D Gaussian filter kernel is used, to maintain a constant overall brightness. While cross advection blurs the streakline sideways to the flow direction, the diffusion filter affects texels not only in direction of the flow but in all directions. Since an error-affected data point exerts influence to all its adjacent points in real data measurements, this filtering process imitates natural diffusion. The maximum function is not recommended as filter function for error diffusion because it would lead to an extreme increase in brightness caused by the larger 2D footprint of error diffusion (as compared to the smaller 1D footprint of cross advection), which means, there is a higher probability of collecting bright contributions than in the cross advection approach.

The diffusion computation is completely detached from the advection step and can be computed separately as shown in Figure 3.5. In this second step, a discretized 2D filter kernel is applied to the previously advected particles. Just as cross advection, error diffusion is an image-processing operation that can be directly mapped to GPU fragment operations. Error diffusion sums the terms within the support of the 2D filter kernel. For a GPU implementation, it is inefficient to use a large filter kernel because this would increase the computation costs dramatically. Therefore, only a discrete, separated $3 \times 3$ Gaussian filter is implemented. A larger filter kernel is achieved by successive application of this $3 \times 3$ filter, where

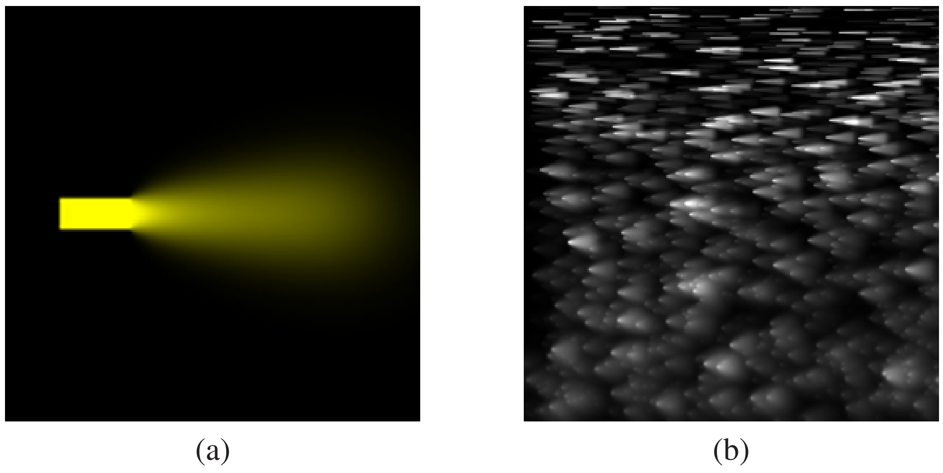


Figure 3.8: Image (a): Gaussian error diffusion with a single dye pattern in a laminar flow with piecewise constant uncertainty. Image (b): Sparsely injected particles into the same flow, with uncertainty magnitude increasing from top to bottom. (Images courtesy of Botchen *et al.* [12], ©2005 IEEE).

the number of filtering steps is determined by the extent of uncertainty. A fine adjustment of filter strength through the uncertainty value is achieved by modifying the entries in the filter mask. The actual filter is constructed from a linear interpolation between an identity mapping and a full Gaussian kernel, where the interpolation weight is determined by the uncertainty value. Linear interpolation guarantees that the integral over the interpolated filter remains constant and normalized. In this way, it is possible to obtain a range of filtering results, all the way from an identity mapping in regions with no uncertainty (which results in exact streaklines) up to a standard Gauss filter in regions with maximum uncertainty (which strongly blurs the streakline in all directions). For the GPU implementation, both the identity mapping and the Gaussian function are held in a floating-point texture. During runtime, a bilinear lookup is performed in this texture that computes the linear interpolation to obtain a modified filter kernel. As demonstrated in Section 3.4.4, GPU implementations lead to interactive visualizations even for very large datasets in the range of $10^6$ grid points.

Figure 3.8 (a) illustrates dye injected into a laminar flow field with constant uncertainty, beginning a few steps away from the injection point. Well recognizable is the one-to-one mapping of the streakline in areas without error influence, which changes to a constant blurring in all directions in the error-affected region. Figure 3.8 (b) shows the Gaussian error diffusion approach applied to sparsely injected particles into the same flow with increasing uncertainty from top to bottom. This picture illustrates that the size and weight of the filter kernel depends on the uncertainty magnitude; hence streaklines in the lower part of the flow are heavily

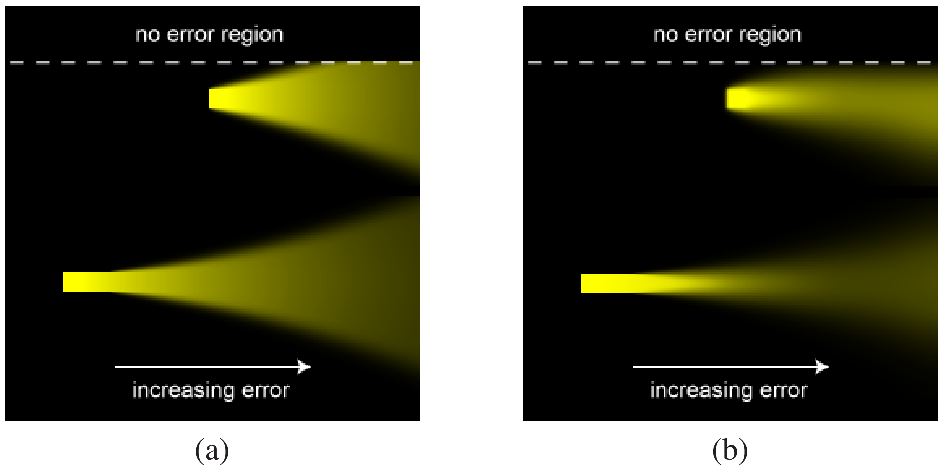


Figure 3.9: Comparison of cross advection (a) and Gaussian error diffusion (b) with two dye patterns injected into a flow with increasing uncertainty from left to right. (Images courtesy of Botchen *et al.* [12], ©2005 IEEE).

blurred while streaklines in the upper part are mapped to identity.

For comparison, Figure 3.9 shows both approaches with two injected dye patterns into a laminar flow field with increasing uncertainty from left to right. Considering the lower dye in the left image, one can see how uncertainty magnitude affects the step size of the cross advection and how the streakline widens due to an increasing step size. Depending on the 2D convolution kernel, the wavefront of a streakline computed with the error diffusion approach runs faster than traditional texture advection or the cross advection approach.

**Multi-Frequency Noise** The generic multi-frequency noise approach [74] is adopted as third technique for a dense vector field representation. Here, uncertainty is used to control the spatial frequency of noise injection. This technique can be directly incorporated into semi-Lagrangian advection by slightly modifying the injected noise. The original 2D noise is replaced by a 3D noise texture whose layers are filled with noise patterns of varying maximum spatial frequency. The first slice contains original white noise; all successive slices contain filtered versions of this white noise with decreasing maximum frequency. Filtering is based on the fast Fourier transform: First, the original white noise is transformed to frequency space, then a low pass filter is applied in frequency space and finally a inverse Fourier transformation back to image space is performed. Since low-pass filtered images lose contrast, histogram equalization is applied to match the contrast of the original image and sharpen the low-frequency structures. To access the different layers, the noise lookup is extended by a third dimension that is controlled by the error value. Figure 3.10 gives an example of four different noise patterns and their

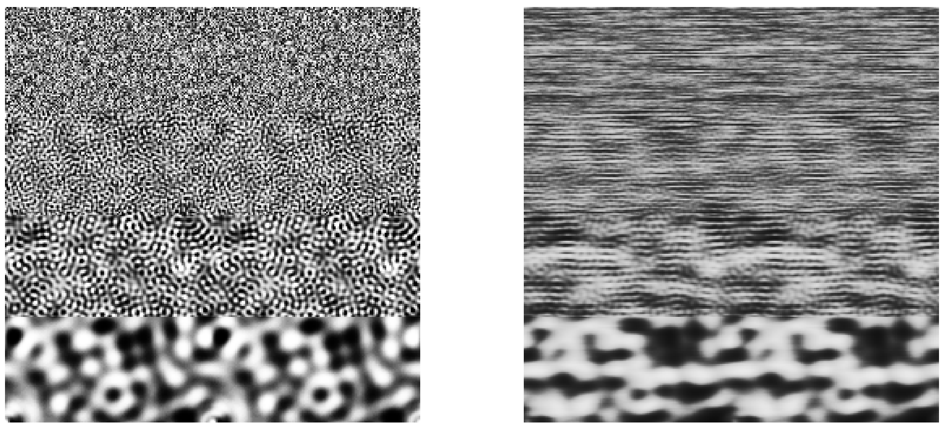


Figure 3.10: Image (a): Four layers of multi-frequency noise. The frequency depends on the uncertainty value. Image (b): Advection of multi-frequency noise in a laminar flow. (Images courtesy of Botchen *et al.* [12], ©2005 IEEE).

application to the visualization of a uniform flow field.

Both post-filtering methods proposed so far are adequate for sparse and dense particle injection patterns. In contrast to them, the multi-frequency approach is not suitable for a sparse particle injection, because low-pass filtering of sparse injected particles would lead to artifacts as known from heavy JPEG compressed images. Further, one would recognize single, isolated and large streaks, but their width would not change depending on uncertainty. Therefore, this approach is not intuitive enough for uncertainty visualization.

### 3.4.3 Color-Based Uncertainty Visualization

Although texture-based uncertainty visualization already leads to an intuitive visual encoding of uncertainty information, additional visual cues can be used to further enhance perception. Besides spatial visual structures, which are exploited by texture-based visualization, color generally plays a dominant role in human visual perception. An advantage is that color, shape and position are highly separable perceptual dimensions [159, p. 180] — color and texture patterns can be easily combined without impairing each other. Therefore, in this section the simultaneous assignment of color and texture patterns is used to propose two approaches that improve texture based uncertainty visualization.

***Color Mapping*** The first approach directly maps $u(\mathbf{x}, t)$—the uncertainty extent to color, by employing a 1D color table. This color table takes the uncertainty as the 1D input parameter and outputs an RGB value. The resulting color is combined with the grey-scale property texture from the advection process by blending

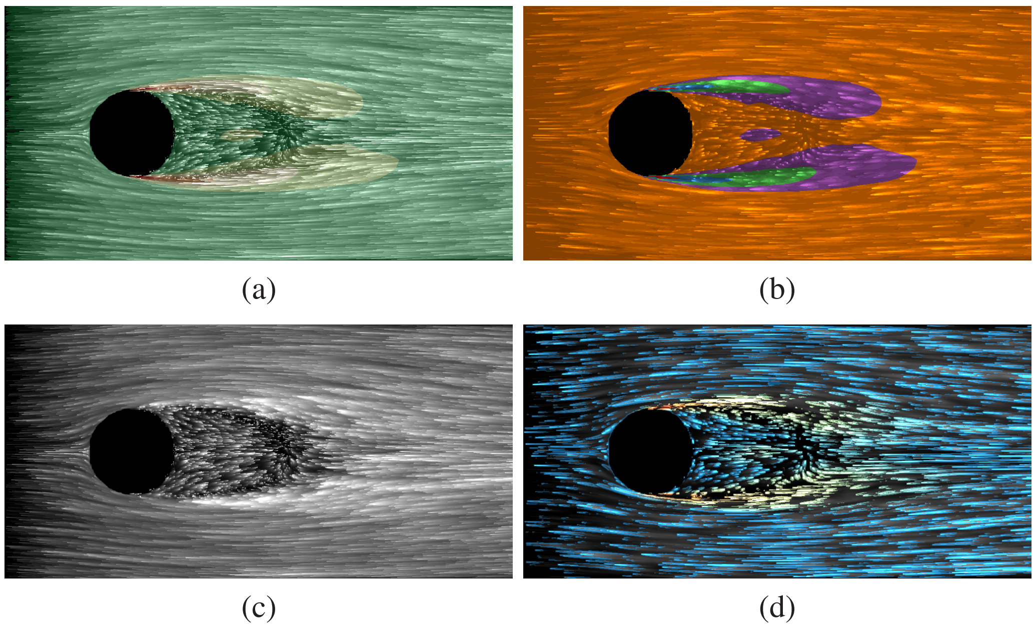


Figure 3.11: Examples of color enhanced texture-based flow visualization techniques showing the 84th layer ($z/D \approx -1.016$) of the LES cylinder dataset. Quantitative color coding (a); and qualitative color mapping with discrete colors (b). Image (c) shows uncertainty based cross advection (b); and the uncertainty-edges technique combined with a spectral (blue to red) color map is illustrated in (d). (Images courtesy of Botchen *et al.* [13], ©2006 ISFV).

or modulation. This algorithm can be performed in one render pass and needs no implementation as additional module, which makes the realization simpler and implicates higher performance.

A popular way of using a color table is to display quantitative data. It is important that the color table is designed in a way that supports the accurate perception of quantitative data, for example, by using a perceptually uniform color scale. More details on the design of color tables are discussed by Ware [158]. Another application of color tables uses color to separate regions—essentially labeling those regions. In this sense, color is mainly used to encode nominal information; color can be extremely effective as a nominal code [159, p. 123]. Each color represents an interval of uncertainty values. The advantage of such a discrete color coding is that regions can be more easily recognized than with a quantitative color scale. The discrete color coding is particularly useful when a qualitative understanding of the visualization is more important than reading off quantitative data.

Examples for the specific color coding techniques are shown in Figure 3.11. Image (a) illustrates qualitative color coding of uncertainty, and image (b) in Fig-

```
1  float4 lft    = tex2D(AdvTx, TxCoord1);    // get the intensities for all
2  float4 rgt    = tex2D(AdvTx, TxCoord2);    // neighbors around TxCoord0
3  float4 up     = tex2D(AdvTx, TxCoord3);    // from the advected texture
4  float4 lo     = tex2D(AdvTx, TxCoord4);
5  float4 uplft  = tex2D(AdvTx, float2(TxCoord3.x-PxSize.x, TxCoord3.y).xy);
6  float4 uprgt  = tex2D(AdvTx, float2(TxCoord3.x+PxSize.x, TxCoord3.y).xy);
7  float4 lolft  = tex2D(AdvTx, float2(TxCoord4.x-PxSize.x, TxCoord4.y).xy);
8  float4 lorgt  = tex2D(AdvTx, float2(TxCoord4.x+PxSize.x, TxCoord4.y).xy);
9
10 float4 row    = uplft + 2*up + uprgt       // weight the upper and lower row
11                 - lolft - 2*lo - lorgt;    // for edge detection
12 float4 column = uprgt + 2*rgt + lorgt      // weight the left and right row
13                 - uplft - 2*lft - lolft;   // for edge detection
14 float4 isEdge = abs(row) + abs (column);   // sum up the values to one final
15 float  res    = dot(isEdge, vIdent);       // scalar representing the edge
16
17 float  errval = tex2D(ErrTx, TxCoord0);    // get error extent at this point
18 float  color  = tex2D(AdvTx, TxCoord0);    // get color from advected texture
19
20 Output        = color;                     // assign original advected color
21 if(res > Threshold) {                      // if a strong edge exists
22     Output    = tex2D(TxErrorTF,           // assign uncertainty color
23                 float2(color.x, errval).xy);
24 }
25 return Output;                             // write final color to buffer
```

Listing 3.2: The main part of the HLSL fragment program for the uncertainty edge approach, without the actual advection step. (Code courtesy of Botchen *et al.* [13], ©2006 ISFV).

ure 3.11 demonstrates discrete, quantitative color coding of uncertainty for the LES dataset. More information on the dataset is detailed in Section 3.4.4. For comparison, Figure 3.11 (c) shows the monochrome, uncertainty based cross advection approach applied to the same LES dataset. Figure 3.11 (d) gives an example of the uncertainty edge approach introduced below. This technique combines the monochrome streaklines from traditional texture advection with color coded edges around the heads of the streaklets.

***Uncertainty Edge Detection*** This second color mapping based approach is another useful method for visualizing uncertainties. It is realized as 2-pass rendering algorithm, whereby the important second pass is detailed as HLSL code snippet in Listing 3.2. In the first stage, traditional texture advection is employed as described in Section 3.4.1. In the second stage, an edge-detection filter—namely a 3×3 Sobel operator (Listing 3.2, Lines 1–15)—is applied to the result of stage one, extracting all edges, which generally are located around the injected particle and along the side of its short streakline. In the same step, a color mapping of the uncertainty value is applied (Listing 3.2, Line 22). If no edge exists at the evaluated region, the result from stage one—the color of the original particle traces—is used instead. An advantage of this technique is that it does not modify or filter the original streaklines, computed in the first stage and, thus, additional information (e.g., pressure or density) could be encoded. A slight disadvantage of this technique is that the edges can clutter the final visualization. Therefore, it is better suited for a sparse particle injection, where streaklines are clearly separated. Figure 3.11 (d) demonstrates uncertainty edge detection for the LES test dataset.

### 3.4.4 Application Cases

The introduced techniques are now illustrated on several test datasets. The laminar water channel dataset shown in Figures 3.12 and 3.13 was measured by a 3D S-PIV method. This dataset contains one time step of water streaming through the test channel that was designed for studying laminar-turbulent boundary layer transitions in a water channel. The measurements, conducted by the Institute for Aerodynamics and Gasdynamics at the Universität Stuttgart, consist of a $81 \times 45 \times 9$ hexahedra-based dataset which contains one time step of water inflow, streaming through the channel, forming vortices in the current. For the final data, the experiment was conducted 25 times for the 9 separated 2D layers and averaged to one velocity field. Further information on the acquired original data can be found in [99].

The wall-mounted finite cylinder datasets illustrated in Figures 3.11 and 3.14 were computed on an unstructured grid with 12.3 million grid points, with LES and DES methods. The simulation for the original datasets was performed on the same grid with a Reynolds-number of 200,000. The final data is averaged, using 3083 single time steps. Computation timings for the cylinder datasets are given for a cluster computer with 42 IBM pSeries 690 PCs with 1.3 GHz. For the 12.3 million grid points, it took the cluster 3.1 minutes for LES and 5.0 minutes for DES per simulated time step. More details on the original datasets and its acquisition can be found in [43]. The visualization of the DES and LES simulations is an example of comparative visualization, i.e., the magnitude of the vector difference between distinct simulation results serves as uncertainty measure $u$. Since our GPU visualization techniques are designed for Cartesian grids, to exploit the SIMD (single instruction multiple data) architecture of modern graphics hardware, it was necessary to resample the original dataset on an equidistant grid. Therefore, we placed a cube with the dimensions $x \in [-1.5, 4.5]$, $y \in [-1.5, 1.5]$, and $z \in [0.0, -3.0]$ in the interesting region of the original dataset, and resampled it with $256 \times 128 \times 128$ sample points.

Figure 3.12 directly compares the three filtering based uncertainty approaches by using sparse and dense particle injection represented by a white noise pattern. The images show the fifth layer of the water channel dataset. Clear streaklines are generated with traditional semi-Lagrangian texture advection in Figure 3.12 (a) using alpha blending as exponential filter along flow direction. Streaklines only widen in divergent parts of the vector field and due to numerical diffusion. Im-

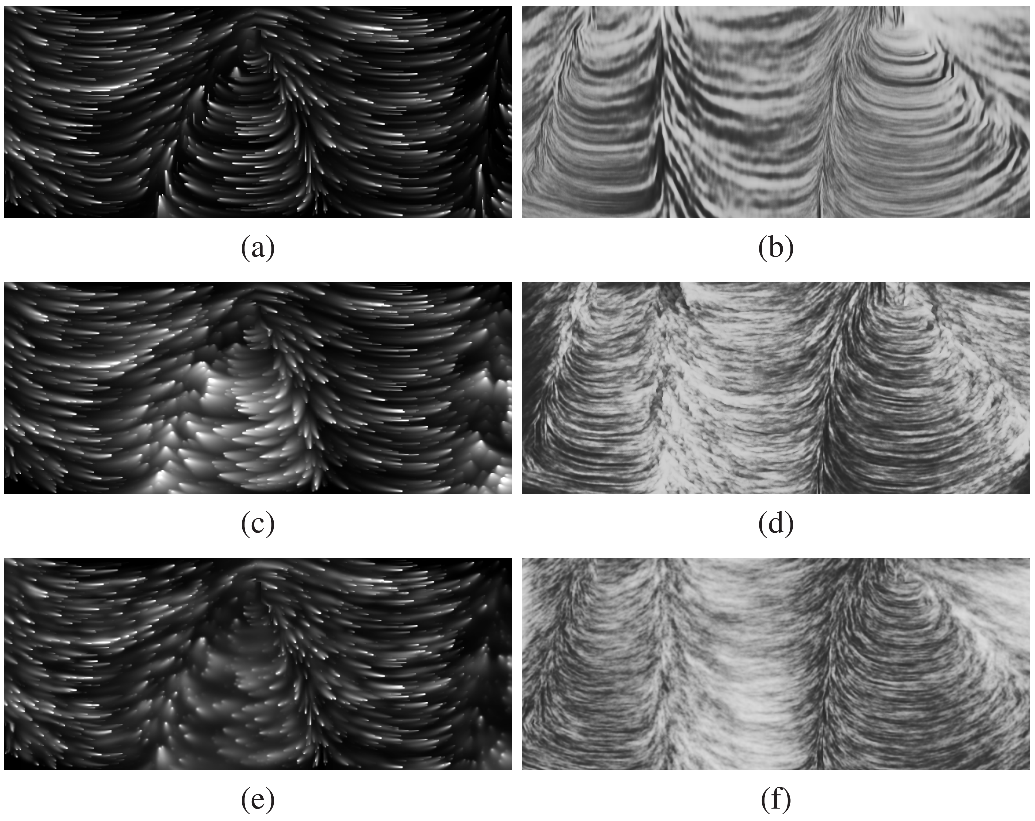


Figure 3.12: Visualizing the 5th layer of the PIV dataset. The left side of the upper six images shows: semi-Lagrangian texture advection (a); sparse representations of cross advection (c); and Gaussian error diffusion (e). The right side illustrates all three approaches using a dense representation: multi-frequency noise (b); cross advection (d); and Gaussian error diffusion (f). (Images courtesy of Botchen *et al.* [12], ©2005 IEEE).

ages (c,d) show the flow visualized with the cross advection approach, for sparse and dense particle injection respectively. In regions with marginal uncertainty, the streaklines remain clear but in uncertainty-affected regions they become blurred perpendicular to the flow by an extent that depends on the uncertainty value. The same applies to sparse and dense particle injections of images (e,f), generated by Gaussian error diffusion. With both techniques, the user can still see structures of the flow in error regions, though the spatial frequency has been reduced. Furthermore, even the orientation of the flow is distinguishable due to the OLIC-like [160] structure of the streaklines. As anticipated, the spatial frequencies strongly decrease in regions of large error, irrespectively of the method being pre-filtered or post-filtered. All techniques produce similar results and eliminate structures of the streaklines in error regions. These approaches can be applied for dye patterns in the same way as for dense particle textures, which enable the engineer to

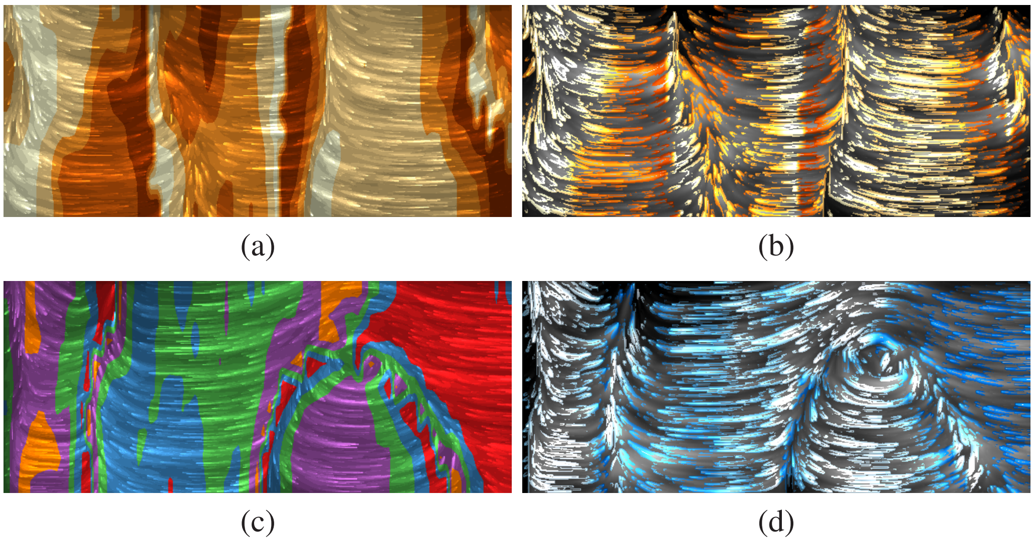


Figure 3.13: Illustration of the 6th layer of the PIV measured water channel dataset, rendered with a sequential (orange) color map and blended with the texture-advection result (a); and with applied uncertainty edges (b). The 8th layer of the PIV dataset. Image (c) shows a combination of cross advection and a discrete, qualitative color map; and in image (d) uncertainty edge detection combined with a sequential color map. (Images courtesy of Botchen *et al.* [13], ©2006 ISFV).

interactively release dye in interesting regions to explore features of the flow.

Figure 3.13 shows results for the color based approaches. Images (a,b) illustrate the sixth layer and images (c,d), the eighth layer in of the measured dataset with different color maps. The left images apply a color map to the complete texture-based flow visualization, i.e., to the whole domain. In contrast, the right images only apply color coding to uncertainty edges; here, uncertainty is only shown in regions where particles are visible, freeing image space in the other areas. The unmanipulated streaklines could be used to display an additional variate (e.g., pressure). Images (a,b) and (d) in Figure 3.13 use a sequential color table that maps the degree of uncertainty from low (white) to high values (highest color saturation). In image (c) the qualitative color mapping represents five different regions of uncertainty: high uncertainties are represented by red or blue colors, whereas regions with no or little uncertainty are mapped to orange and purple color and mid-range values are displayed as green color. In this way, the viewer's attention is drawn to regions with strong deviations. All visualizations show that the areas of high uncertainty lie in regions where the flow moves in the direction of the third dimension, i.e., a weakness of this PIV method is the accuracy in the third dimension, which is plausible since the measurements are performed on 2D layers.

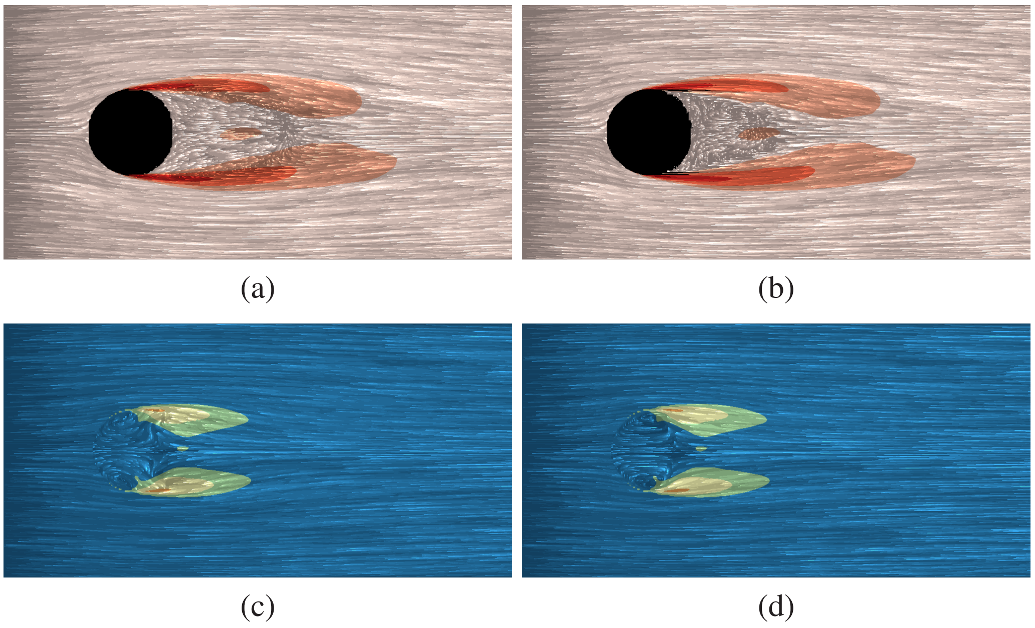


Figure 3.14: Comparing of the LES (left) and DES (right) datasets, encoded by sequential red colors (a); and with a spectral (blue-red) color map. The top row shows is the 84th layer ($z/D \approx -1.016$), the bottom row shows the 40th layer ($z/D \approx -2.055$), located right on top of the cylinder. (Images courtesy of Botchen *et al.* [13], ©2006 ISFV).

The visualization of the DES and LES simulations is an example of comparative visualization, i.e., the difference between different simulation results serves as uncertainty measure. All images of the simulated datasets shown in this paper use the magnitude of the velocity difference vector as uncertainty extent, whereas other values like pressure, density or temperature can be used instead, if they are a matter of particular interest. High uncertainties appear in regions with vortex separations near the surface or in shear layers behind the cylinder. Figure 3.14 shows a comparison of two different layers of both simulations. Images (a) and (b) show the difference of velocity magnitude in the computed shear layers behind the cylinder. A red color table is applied, sequentially enhancing saturation with increasing uncertainty. This kind of mapping is used in most applications since color and uncertainty are intuitively coupled. Nevertheless, the use of a diverging color map, as shown in the right image, can emphasize the visualization. Images (c) and (d) of Figure 3.14 illustrate vortex separations near the upper edge of the cylinder with a spectral color map. Noticeable are the small differences in flow directions of both datasets. While blue regions with very low uncertainties fade to the background, the red-colored regions with high uncertainties are emphasized. These uncertainties are due to the different simulation approach that is chosen near boundary regions for DES and LES.

| viewport | I | II | III | IV | V |
|---|---|---|---|---|---|
| 1024×512 | 901 | 429 | 507 | 822 | 539 |

Table 3.1: Rendering performance in frames per second, measured on a Pentium IV with 3.4 GHz and an NVIDIA GTX 7800 GPU. Techniques are (I) conventional texture advection; (II) cross advection; (III) error diffusion; (IV) uncertainty color coding; and (V) uncertainty edges, respectively. Note that the timing of the multi-frequency noise approach equals technique (I), as it works on a pre-filtered input texture.

The implementation of texture-based uncertainty visualization is based on C++, Direct3D 9.0, HLSL, and FX files. Respective performance numbers are documented in Table 3.1. Rendering speed depends linearly on the number of texels. Cross advection, error diffusion and uncertainty edge detection have similar performance since all three techniques need an additional render pass and more texture lookups. Due to single pass rendering and half as many lookups, the multi-frequency approach and the color mapping approach are nearly twice as fast.

## 3.5 Combined Feature Visualization

The preceding sections presented 2D flow visualization techniques with the ability to embed one or two additional flow features – i.e., uncertainty, velocity magnitude or pressure – into the visualization. This section aims at visualization techniques for three-dimensional flow fields with the capability to include multiple flow features into the final illustration. For three-dimensional vector fields the challenge to produce a proper visualization is more complex. Due to the additional dimension, the final illustration can suffer from cluttering and occlusion. Further, if these fields are given on anything other than a regularly structured grid, it can make the process of computation and analysis highly complicated. This fact leads to the need to reduce the large data flood and present only desired details of the data to the viewer. One possible approach for this purpose is the use of a higher level of abstraction to represent the data as done by flow topology methods [95], although these methods are mostly limited to work on singularities of the underlying field. Another approach is to partition the flow field based on user-defined, combined properties [126], and to display the flow structures only in regions that fulfill these conditions.

The proposed multi-field flow feature visualization system, as shown in Figure 3.15, builds on the latter idea of reducing the amount of data for visualization, by merging several features to one user-defined criteria. Therefore, multiple features are combined using first-order fuzzy logic (FOFL) operators to create one characteristic set of all input features, which is then used as a basis for further

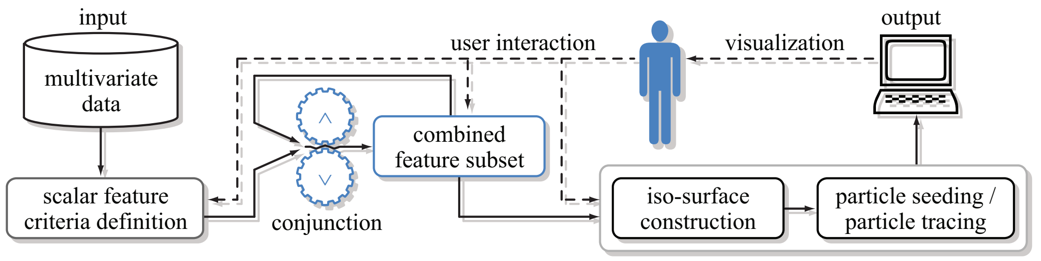


Figure 3.15: Work flow of the multi-field feature combination and visualization system. (Image courtesy of Botchen *et al.* [11], ©2008 ISFV).

computation. The system allows the user to define and combine multiple feature criteria as logic point predicates and display the resulting characteristic set as isosurface with geometric primitives. For this purpose, the system is subdivided into three layers; (1) feature definition and extraction, (2) logical combination of feature sets and (3) visualization of the structures created by the feature sets as isosurfaces in combination with the surrounding flow behavior as particle tracelines.

### 3.5.1 Definition of Feature Criteria

As a first step for feature based visualization, proper feature extraction methods are essential. Thus, an important task is to find an appropriate definition that leads to a good detection and thereupon to an adequate representation of the desired feature. There exist a multitude of features for flow fields and scalar fields that are supported by the system. An overview of flow features is given in Section 3.3. The system extracts the features in a semi-automatic pre-processing step from the given dataset. This extraction results in a normalized scalar field, i.e., the fuzzy domain of the feature. In principle, any possible criterion can be defined, as long as it can be mapped to a scalar field.

For an interactive analysis, the question of what is or is not considered to be a feature eventually refers to the user and depends on what part of the data the user is interested in. Hence, on the resulting scalar fields from the semi-automatic extraction, the user is enabled to interactively define and combine several desired feature criteria for illustration during analysis. It should be noticed that features can be either global or local; the former are influenced by all values of the field, whereby the latter represent a local quantity. However, all features are a characterization of a set of points. The definition of a feature is usually closely coupled to the dataset to be analyzed and depends on the requirements of the analysis.

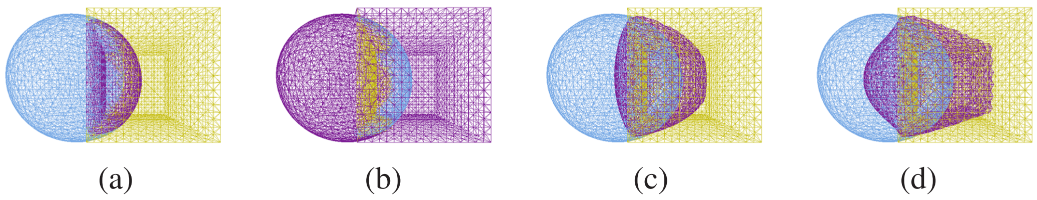


Figure 3.16: Images (a,b) illustrate the results of the $C^0$-continuous min/max-logic for an artificial isovalue dataset consisting of a sphere overlapped by a cube. The conjunction $\wedge$ is shown in (a), and the disjunction $\vee$ in (b). Images (c,d) show the results of the $C^\infty$-continuous multiplicative logic, with the conjunction $\wedge^*$ in (c), and the disjunction $\vee^*$ in (d) respectively. (Images courtesy of Botchen *et al.* [11], ©2008 ISFV).

### 3.5.2 Using First-Order Fuzzy Logic

In the second stage, the system implements predicates to formulate functions on the characteristic feature sets with logical operators $\{\neg, \wedge, \vee\}$. The difficulty of using standard FOL is its binary nature, meaning that a predicate can either be true or false. Yet, most feature extraction methods do not classify the feature in a Boolean way, but rather give a fuzzy version of a characteristic feature set that tells about the extent of the feature. Therefore, fuzzy point predicates are employed to indicate whether a feature exists to a certain extend at a given location of the dataset, meaning that we can describe a continuous connected region as feature. Predicates of valence 1 express statements like high pressure exists and are used to operate and extract features from the data. Therefore, all implemented predicates operate on a normalized range and the continuous FOFL fulfills the following three conditions:

1. It is consistent with binary Boolean logic $P(x) = \{\text{true, false}\}$.
2. The De Morgan's rules hold for all possible values.
3. Conjunctions are steady functions.

For a feature criterion $\xi$ given on the domain $D_\xi$, a fuzzy predicate $\mathcal{P}_\xi$ and a combined characteristic set $CS$ are defined as follows:

$$\mathcal{P}_\xi : D_\xi \rightarrow [0,1] \quad , \qquad CS = \bigcup_{\mathcal{P}_\xi(D_\xi)>0} D_\xi \quad .$$

Here, $\xi$ may represent $\lambda_2$, $s_H$, $\|v\|$, $\|\omega\|$, $p$, $\rho$ or $u$ and the fuzzy value represents the extent of a feature at a given point. Two different logic approaches – min/max logic and multiplicative logic operators – are tested and evaluated to give the positive and negative aspects of those operators, when applied to flow visualization.

***Min-Max Logic*** The operators for min-max logic are defined as follows:

$$\begin{aligned}\mathcal{P}_{\xi_k} \wedge \mathcal{P}_{\xi_l} &= \min\{\mathcal{P}_{\xi_k}, \mathcal{P}_{\xi_l}\} \quad , \\ \mathcal{P}_{\xi_k} \vee \mathcal{P}_{\xi_l} &= \max\{\mathcal{P}_{\xi_k}, \mathcal{P}_{\xi_l}\} \quad , \\ \neg\mathcal{P}_{\xi} &= 1 - \mathcal{P}_{\xi} \quad .\end{aligned}$$

***Multiplicative Logic*** The second type of operators we use in our system implements multiplicative logic, leading to the following definition of operators:

$$\begin{aligned}\mathcal{P}_{\xi_k} \wedge^* \mathcal{P}_{\xi_l} &= \sqrt{\mathcal{P}_{\xi_k} \cdot \mathcal{P}_{\xi_l}} \quad , \\ \mathcal{P}_{\xi_k} \vee^* \mathcal{P}_{\xi_l} &= \neg(\neg\mathcal{P}_{\xi_k} \wedge^* \neg\mathcal{P}_{\xi_l}) \quad , \\ \neg\mathcal{P}_{\xi} &= 1 - \mathcal{P}_{\xi} \quad .\end{aligned}$$

From a geometrical point of view, the min-max logic is equivalent to set operations on isosurfaces and is in general $C^0$-continuous. This is adequate for geometry construction algorithms that are applied later on, since they are commonly $C^0$-continuous too. However, from the combinational point of view, the results of min-max logic might not be satisfactory enough. One of its characteristics - i.e., the stronger or weaker feature dominates the result – can completely neglect the other one and thus, lead to a loss of information. Here, the use of multiplicative logic can be advantageous, because of its $C^\infty$-continuous nature. This means that the resulting set of a combination covers a region that is influenced by all participating features. The effect of the different behavior for the two logics are illustrated in Figure 3.16. The example shows the concatenation of two artificial feature sets representing a sphere (blue) and a cube (yellow). Both features increase from their barycenter. The isosurfaces was built with an isovalue of 0.5 and the result of the respective operation is colored in purple. An important aspect of the multiplicative logic is the fact that the operators are not associative and therefore

$$\begin{aligned}(\mathcal{P}_{\xi_k} \wedge^* \mathcal{P}_{\xi_l}) \wedge^* \mathcal{P}_{\xi_m} \neq \mathcal{P}_{\xi_k} \wedge^* (\mathcal{P}_{\xi_l} \wedge^* \mathcal{P}_{\xi_m}) \quad \text{and} \\ \mathcal{P}_{\xi_k} \wedge^* \mathcal{P}_{\xi_k} \neq \mathcal{P}_{\xi_k} \quad .\end{aligned}$$

In other words this means that the individual fields are not weighted equally by combination. Here, the last accumulation is the one with the strongest weight and thus, the order in which the different fields are combined needs to be considered.

### 3.5.3 Interactive Combined Feature Visualization for Analysis

The third stage provides visualization by giving a structural overview of the extracted feature sets in combination with an illustration of the surrounding flow behavior as streamlines. Depending on the underlying dataset, different aspects

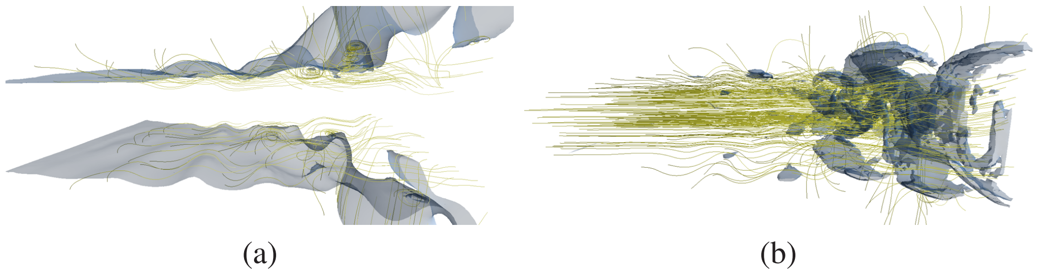


Figure 3.17: The images illustrate the structural results for feature extraction on the ejection nozzle dataset for low velocity magnitude $CS = \mathcal{P}_{||v||} = 0.1$ (a) and high vorticity magnitude $CS = \mathcal{P}_{||\omega||} = 0.9$ (b). (Images courtesy of Botchen *et al.* [11], ©2008 ISFV).

of the data can be of interest to the user. In general, engineers have a basic idea about what kinds of features are important for them and how to define and combine them to obtain a reasonable characteristic feature set. However, an accurate representation of the desired feature set is often achieved by repeatedly tuning the parameters — usually done by adjusting the threshold value $c$, or the length of the traced lines and thus, for an interactive analysis there should be the possibility to customize the feature sets to the users needs. The system enables the interactive modification of parameters for feature extraction. Based on the characteristic feature set, a structural overview is build with 2D geometric primitives and 1D tracelines.

The extracted features are given as point set which can be simplified and described quantitatively in order to be visualized by geometry. The algorithm detects the hull of the feature set, given by the threshold value $c$ to build the isosurface by means of $\{\forall T\ \exists G : CS_T = c\}$. Since the system works on tetrahedral grids, this implies that for all tetrahedra $T$ a geometric primitive $G$ is constructed, if the isosurface runs through it. The construction of the geometry is computed with the Marching-Tetrahedra algorithm [153], an adapted version of the more common Marching-Cubes algorithm [21; 90], Once the surface grid is computed, it is used as source for particle tracing. As seeding strategy, we place a particle on the barycenter of every grid polygon, to cover the whole feature domain. Outgoing from their initial seeding position, the particles are traced with the second order Runge-Kutta method. A bonus of this elementary approach is that it avoids the problems of occlusion and clutter that can occur, when seeding particles in the whole domain and thus, the user can directly explore the flow behavior in regions of interest by a sparse seeding on the chosen feature domain. On the other hand, as shown in Figure 3.17 (a,b) for different velocity magnitudes, the choice of a single non combined feature set can cause the particle tracing to miss important parts of the flow.

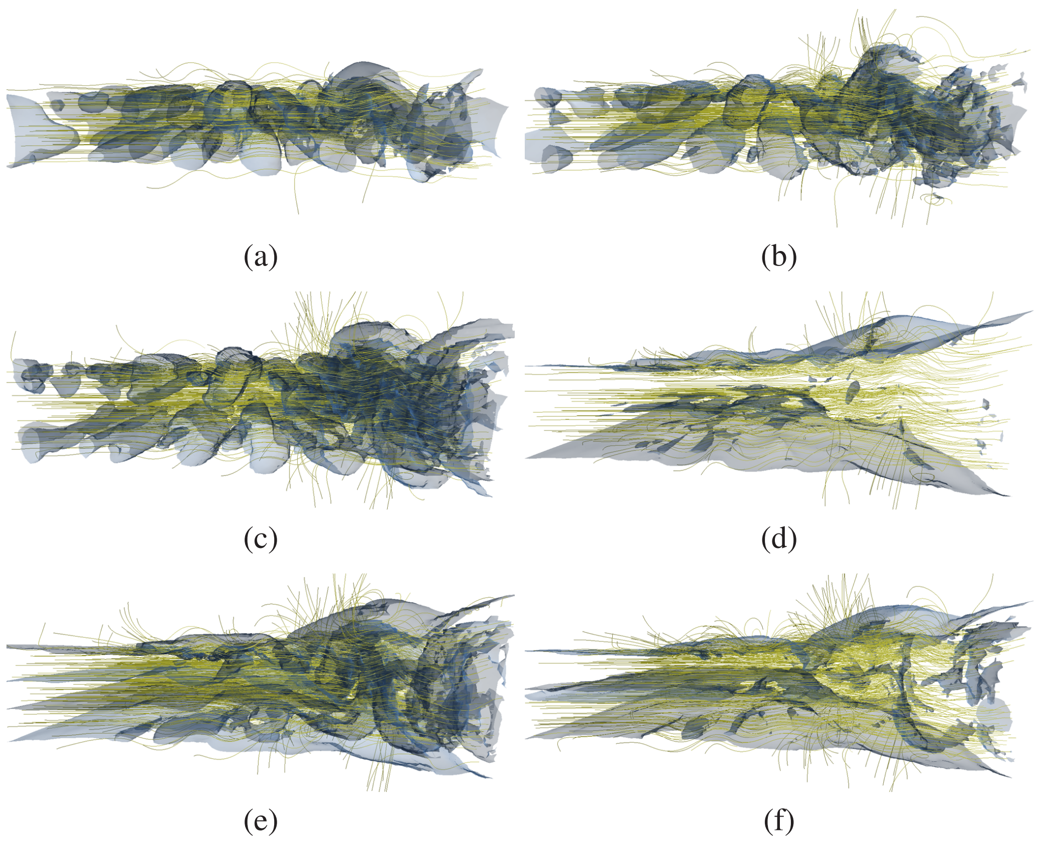


Figure 3.18: (a) $CS = \mathcal{P}_p = 0.1$, (b) $CS = \mathcal{P}_{\lambda_2} \wedge^* \mathcal{P}_p = 0.1$ (c) $CS = \mathcal{P}_{\lambda_2} = 0.1$, (d) $CS = \mathcal{P}_{s_H} \wedge^* \mathcal{P}_{\|v\|} = 0.1$ (e) $CS = \mathcal{P}_{s_H} = 0.1$ (f) $CS = (\mathcal{P}_{\lambda_2} \wedge^* \mathcal{P}_p) \vee^* (\mathcal{P}_{s_H} \wedge^* \mathcal{P}_{\|v\|})$. (Images courtesy of Botchen *et al.* [11], ©2008 ISFV).

Further evaluation of the system was performed on three datasets coming form different engineering application scenarios. Figure 3.18 shows the spatial evolution of a liquid sheet, ejected at Reynolds number 4,000 from a diverging ejection nozzle. The DNS volume-of-fluid simulation of the liquid was done with the focus to find the most instability enhancing parameters and to give detailed information on the possible outcome concerning the spreading rate and the disintegration mechanism depending on the character of the nozzle. The dataset was computed by the Institute of Aerospace Thermodynamics, at the Universität Stuttgart. It was simulated on a Cartesian grid with $480 \times 384 \times 192$ resolution and 253 time steps. The dataset used for testing shows time step 220, converted to 64,000 tetrahedral cells.

As presented in [127], the nature of flow through converging or diverging channels is strongly influenced by the angle and the length of the nozzle, basically leading to acceleration or deceleration of the flow. Though the diverging nozzle

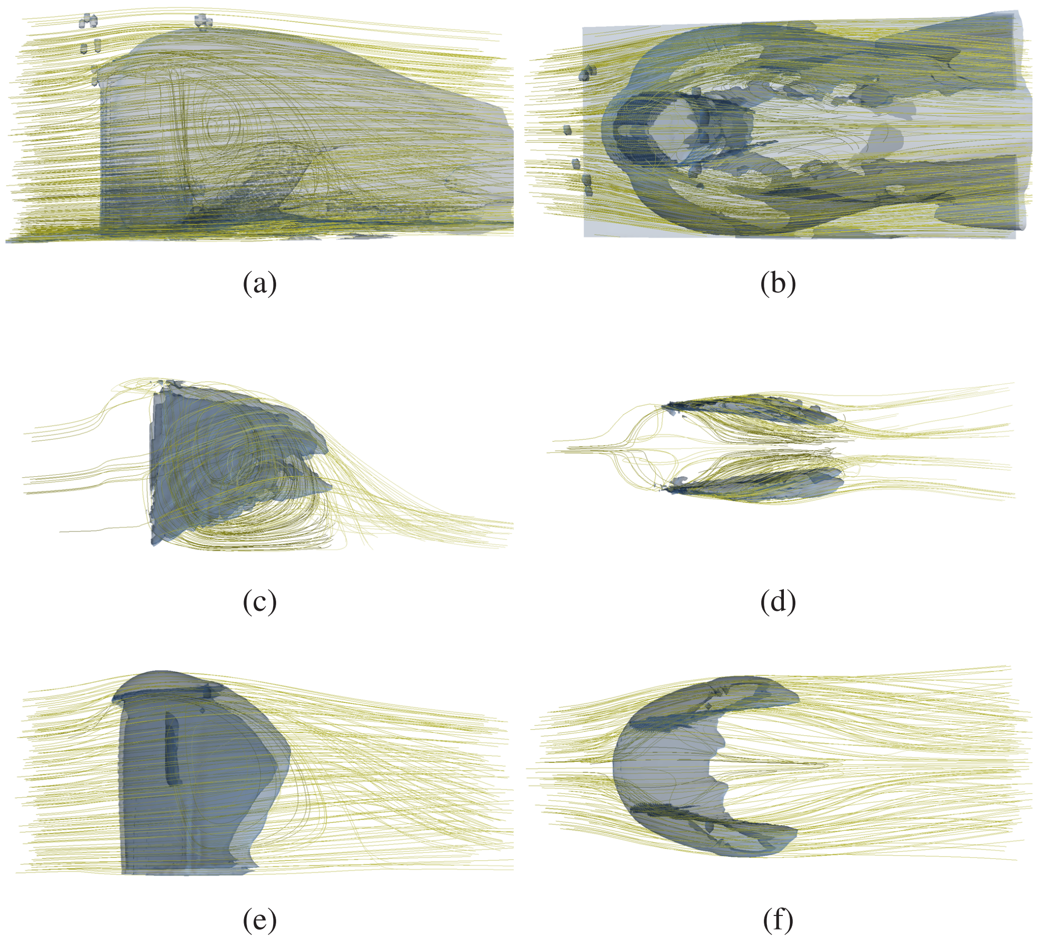


Figure 3.19: The wall mounted cylinder illustrated with combined feature sets: (a,b) $CS = \neg\mathcal{P}_{s_H} \wedge^* \neg\mathcal{P}_u = 0.95$, (c,d) $CS = \neg\mathcal{P}_{\lambda_2} \wedge^* \mathcal{P}_u = 0.9$. (e,f) $CS = (\neg\mathcal{P}_{s_H} \wedge^* \mathcal{P}_{\|v\|}) \vee^* (\neg\mathcal{P}_{\lambda_2} \wedge^* \mathcal{P}_u \wedge^* \neg\mathcal{P}_{\|\omega\|}) = 0.9$. (Images courtesy of Botchen *et al.* [11], ©2008 ISFV).

type represents only a very small technical modification compared to a parallel nozzle, a very high level of kinetic energy flux of $\epsilon = 1.55$ can be achieved, leading to a typical sinusoidal wave instability.

The fuzzy predicates used to create the characteristic sets for images (a-d) are based on $p$, $\|\omega\|$, $\lambda_2$ and $s_H$, respectively. In image (e) the combination $CS = \mathcal{P}_p \bigwedge \mathcal{P}_{s_H}$ was used and for image (f) the characteristic set is defined as $CS = \mathcal{P}_p \bigwedge \neg\mathcal{P}_{\lambda_2}$, $CS = \mathcal{P}_p \wedge \neg\mathcal{P}_{\lambda_2} \wedge \mathcal{P}_{s_H} \wedge \mathcal{P}_{\|\omega\|}$, whereby both operations were performed with multiplicative logic. The results indicate that the increasing wave instability from left to right implicates a gain of $\lambda_2$ as well as $s_H$, while the $p$ slightly decreases at the end of the diverging jet.

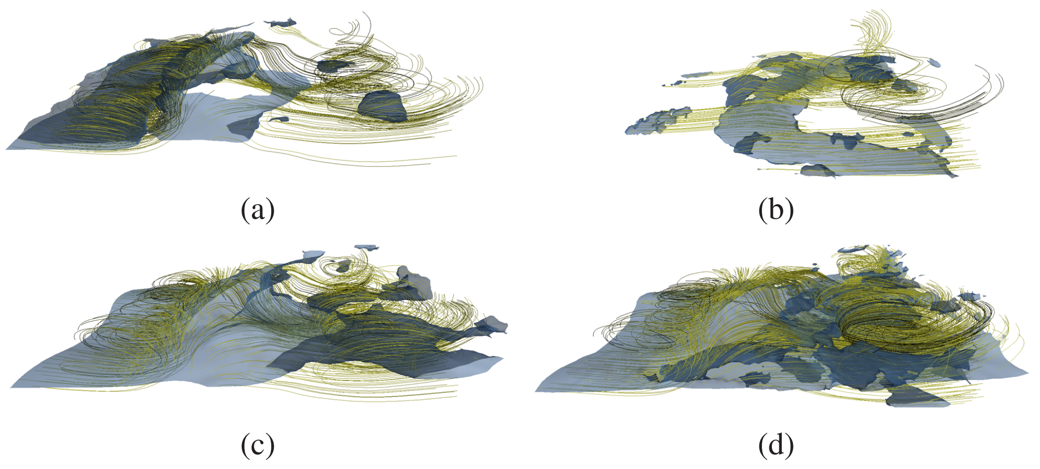


Figure 3.20: Laminar water channel with rendered characteristic sets: (a) $CS = \mathcal{P}_{\|v\|} \vee \mathcal{P}_{s_H} = 0.05$, (b) $CS = \mathcal{P}_{\|v\|} \wedge \mathcal{P}_{s_H} 0.05$, (c) $CS = \mathcal{P}_{\|v\|} \vee^* \mathcal{P}_{s_H} = 0.05$, (d) $CS = \mathcal{P}_{\|v\|} \wedge^* \mathcal{P}_{s_H} 0.05$. (Images courtesy of Botchen *et al.* [11], ©2008 ISFV).

The 12.3 million grid points of wall mounted cylinder Figure 3.19 were resampled with 64,000 tetrahedral cells for the LES and DES datasets. The visualization of the simulations is an example of comparative visualization, i.e., the difference between different simulation results serves as uncertainty measure $u$.

The multiplicative disjunction of negated shear and negated uncertainty results in an isosurface that excludes the uncertain regions but includes the high shear near the bottom, as can be seen in the top view of Figure 3.19 (a,b). A combination of high $\lambda_2$ and high uncertainty is illustrated in Figure 3.19 (c,d) leads to nearly complementary results. Note that $\lambda_2$ and shear need to be inverted, as by definition, a smaller value covers more of the vortical region or the region under shear stress. With more sophisticated combinations as in images (e,f) it becomes clear that regions where all features are combined have a large extend are in the direct neighborhood of the cylinder.

Figure 3.20 gives combined feature set examples for the laminar water channel dataset that was resampled from original dataset with 16,000 tetrahedral cells, since the visualization system was designed for tetrahedral grids. Image (a) in Figure 3.20 shows the characteristic set $CS$ for $\mathcal{P}_{\|v\|}$ and in (b) the extracted isosurface for $\mathcal{P}_{s_H}$ respectively. The illustration of the individual features clearly shows that the relative velocity at the inlet region on the left side is higher than on the right side, where a higher shear value dominates. Logical combinations of those two features are presented with min/max disjunction $CS = \mathcal{P}_{\|v\|} \vee \mathcal{P}_{s_H}$ in (d) and min/max conjunction $CS = \mathcal{P}_{\|v\|} \wedge \mathcal{P}_{s_H}$ in (c). In images (e) and (f) the examples of multiplicative disjunction and conjunction show that the continuous results of multiplicative logical operations lead to a structural feature representation that adequately covers the region to unveil the flow properties of interest.

As can be seen from the resulting images, a sophisticated user defined combination of multiple desired features can give a good structural overview of all regions where these features appear, as well as an adequate amount of seeding geometry that can be adjusted by the user during analysis.

# Chapter 4 Flexible Multi-volume Visualization

Single data volume rendering methods have made significant progress in research and are deployed in many fields of application today [7; 37; 92]. Extensions for single-pass raycasting on GPUs have been proposed [144] and several algorithms have been developed dealing with continuative topics, such as volume clipping [79; 164] and volume deformation [28; 84]. A comprehensive overview on state-of-the-art volume rendering techniques is given in [36]. However, many applications in scientific and medical visualization operate on multiple datasets – derived from different imaging modalities or numerical simulations – rather than only on a single input dataset. The simultaneous analysis of multiple correlated datasets or several different regions of particular interest within a single complex volume is a more sophisticated problem. In this case, a prerequisite for a comprehensive analysis and a proper understanding of the volume data is not only the interactivity of the visualization technique but also an adequate visual representation. Especially for preoperative planning tasks in medical surgery, it can be of great benefit to see several types of data simultaneously by spatially superimposing them, and thereby enable the surgeon to identify anatomical landmarks and critical structures. For example, multi-volume visualization helps the surgeon to picture the precise procedure of an operation and to get the required orientation that is necessary to perform skin incisions or bone cover removal at the optimal location before performing the actual surgery. The combined visualization of several volumes bears various challenges [169], such as to establish visual correspondences while maintaining distinctions among multiple volumes. A general approach for blending several volumes was designed by Cai and Sakas [19]. This approach splits the rendering process in three fundamental rendering pipelines for multi-volume data intermixing. Recent work on multi-volume rendering describes a CPU-based raycasting algorithm for direct rendering multiple overlapping volumes [49], by handling single volume areas and regions where several volumes intersect separately. The compositing of multiple volumes is maintained by sequentially applying the over operator given in Equation (2.10) for each individual volume. Other techniques apply non-photorealistic rendering styles [16], to achieve traditional three-dimensional illustrations as used in educational books. A hybrid algorithm for mixing volumes and translucent polygonal objects [81], and graph based schemes [25; 105] were presented.

In this section, the presented concept for multi-volume rendering is related to the later two approaches, but with the focus on the applicability to GPU-based rendering. The system is based on a graph presentation that wraps complex GPU shading algorithms in single graph nodes, it is able to handle multiple volume datasets that can be combined in one scene, and allows for the rapid development of new visualization applications. The rendering framework is possible to assemble various complex volume rendering styles from predefined pieces of code "on-the-fly" and assign them to parts of one or more volumes. This flexible render graph system can be freely configured from several modularized nodes, leaving the user on an abstract level, while designing a multi-volume illustration.

The described work is based on two publications [122; 123], and was carried out in collaboration with Friedemann Rößler from the Universität Stuttgart, Germany. He must be credited for parts of the implementation and the conceptual design. Before detailing the multi-volume rendering methods for slice-based rendering and raycasting, an overview on medical imaging modalities employed to acquire the used datasets for this work is given.

## 4.1 Medical Imaging Techniques in a Nutshell

Medical imaging refers to modern techniques and processes used to create images of the human body for clinical purposes or medical science. The images are recorded with a computer and are usually stored with 12 bit resolution. The results are used for the study of normal anatomy and physiology or functional analysis of the brain. Different modalities are applied depending on the application area. The commonly used techniques are described in this section.

### Computer Tomography

CT or computed axial tomography (CTA) including conventional, helical and electron-beam forms, is an x-ray based medical imaging method employing tomography [151]. The word "omography" is derived from the Greek tomos (slice) and graphein (to write). The device is equipped with an x-ray sensing unit which rotates around the body to create two-dimensional, cross-sectional images. CT provides clinically relevant anatomic information and is used to study aortic disease, pericardial disease and cardiac masses, among others. It is relatively noninvasive, and has comparatively low short-term and long-term risks.

### Computer Tomography Angiography

CTA is an extended method of CT, using a small dose of contrast material to produce detailed pictures [146]. The iodinated contrast medium is injected through a peripheral vein by a small needle or catheter, to visualize arterial and venous blood flow through major vessels in the body. Just like CT, it is a noninvasive outpatient procedure that produces two-dimensional, cross-sectional images, helping physicians to diagnose and treat medical conditions, with the focus on the heart, lungs, kidneys, arteries, or veins of the body.

### Magnetic Resonance Imaging

MRI or also called nuclear magnetic resonance (NMR) imaging [100], is a procedure that uses powerful magnets placed around a horizontal tube like scanning device to form a magnetic field. The body part has to be placed in the exact isocenter of the magnetic field to be scanned. The generated electromagnetic field is used to align the nuclei of hydrogen atoms in the body and then, record this orientation with point wise, precisely tuned burst of radio waves using a computer. Hit by the radio wave, each distinct material such as tissue, bone or vessel, reflects a brief, unique radio signal of its own. Unlike CT and CTA, MRI is non-ionizing and has no known biological hazards, it is intrinsically three-dimensional. It can produce high-resolution images of the heart's chambers and large vessels without the need for contrast agents. MRI is an acceptable technique for evaluating diseases of the aorta such as dissection and aneurysm, heart muscle diseases and cardiac masses such as intracardiac tumor. It is also preferably used for brain imaging, to outline the affected part of the brain and help define the problems created by stroke. MRI supports surgeons in terms of pre-operative and post-operative analysis.

### Functional Magnetic Resonance Imaging

Functional MRI or short fMRI is a type of specialized MRI scan. It measures the haemodynamic response related to neural activity in the brain or spinal cord of humans or animals and is one of the most recently developed forms of neuroimaging. The ability of fMRI is to observe both the structures itself and also which structures participate in specific functions of neural activity, detected by a blood oxygen level dependent signal [109]. It dominates the functional brain mapping field due to its low invasiveness, lack of radiation exposure, and relatively wide availability. This new ability to directly observe brain function opens an array of new opportunities to advance our understanding of brain organization, as well as a potential new standard for assessing neurological status and neurosurgical risk.

## 4.2 Multi-volume Rendering

For rendering a single volume dataset, there exist a number of GPU-based techniques, e.g., raycasting, splatting, and texture slicing, which numerically evaluate the continuous volume rendering integral as in Equation (2.10). The basic idea of these techniques is to first sample the volume at certain sampling positions, then map the sampled values to color and opacity, and finally compose the resulting color values in correct depth order.

If several intersecting volumetric objects should be visualized in a single image, it has to be decided how the different objects contribute to the final image. For this multi-volume rendering problem Cai and Sakas [19] determined three levels of volume intermixing. For *image level* intermixing, the volumes are rendered independently and the resulting images are combined by a compositing rule which may take the opacity and depth value of the pixels into account. For *accumulation level* intermixing, the visual contributions of the volumes are combined step by step. At each sampling point the sample values of the different volumes are mapped independently to colors and opacities and then these color values are accumulated to a single sample color. For *illumination level* intermixing the sample color is not intermixed from colors and opacities, but directly computed by a special multi-volume illumination model.

While illumination level intermixing allows producing physically inspired results like X-ray images, accumulation level intermixing is the most common approach for illustrative multi-volume visualization, because it provides the possibility of applying independent transfer functions and shading styles to the different volumes. Furthermore, in contrast to image level intermixing, it leads to a correct depth cueing of the volumes. If accumulation level intermixing is used, the GPU-based multi-volume rendering problem is two-fold. On the one hand, a volume rendering algorithm has to be applied which can handle several intersecting volumes simultaneously. On the other hand, the volume intermixing is performed on a per-sample level, which implies that for each combination of shading styles and transfer functions a specialized GPU shader has to be provided. Further, depending on the chosen volume rendering approach the implementation challenges highly differ by means of evaluating and compositing multiple volumes.

The following sections present a framework with an integrated solution for these tasks on the example of two direct volume rendering approaches, i.e., slice-based multi-volume rendering and multi-volume raycasting. This multi-volume visualization framework provides full flexibility for generating meaningful representations of the investigated datasets. It facilitates to intuitively build up a render graph on an abstract graphical level, completely decoupling the operator from the complex process of information handling and shader generation.

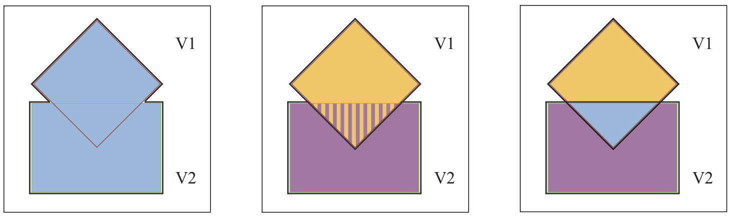


Figure 4.1: Three types of multi-volume slice accumulation on the example of two overlapping proxy slices: Merge (left); Separate (middle); and Intersect (right). Each different colored region is processed by an individual shader. The grey square illustrates the intersection layer of the multi-volume slice. (Images courtesy of Rößler *et al.* [122], ©2008 IEEE).

### 4.2.1 Slice-based Multi-volume Rendering

While the scene description of the framework is independent of the applied rendering technique, the actual GPU rendering algorithm needs to be adapted to the applied rendering method. For the slice-based approach, the volumes in the multi-volume scene are primarily transformed into camera space and then the volume bounding boxes are equidistantly sliced along the viewing direction. This slicing leads to *multi-volume slices*, each containing coplanar proxy slices of the different volumes in the scene. The sampling distance between the multi-volume slices is chosen in relation to the volume with the smallest voxel size. For rendering, the multi-volume slices are processed in back-to-front order and their previously accumulated color contribution is blended with the *over* operator (Equation (2.11)) into the frame buffer. Since the system allows the assembly of shader programs for any combination of volumes in the scene, three different techniques for the multi-volume slice accumulation (see Figure 4.1 respectively) are considered:

***Merge*** This method merges the geometry of all existing proxy slices into a single hull, which is not necessarily convex. Then this hull is tessellated with *gluTesselator* of the OpenGL Utility Library (GLU) and the resulting triangles are rendered with a single shader that accumulates the contributions of all $n$ volumes within the multi-volume slice, requiring no shader switches at all.

***Separate*** The separation technique is based on the strategy to handle each proxy slice separately. Here, no expensive merge of the volume slices is necessary. A single shader for each of the $n$ volumes is assembled, resulting in $n$ shader switches per multi-volume slice.

***Intersect*** The third technique goes one step further by handling each possible combination of intersecting volumes with a different shader. This method needs to divide the regions of intersecting volumes into single bounding polygons and needs to tessellate them. Regarding the shader switches, the upper limit is $2^n$ due to the maximum number of possible combinations of $n$ overlapping volumes.

A further discussion of the advantages and disadvantages of the three accumulation techniques and some comparative performance results for different scenarios can be found in Section 4.5. Independent of how the three slice accumulation techniques handle the different volumes, each technique implements accumulation level intermixing. This means that the visual contributions of the volumes are combined step by step, where the compositing of the different volume samples is often performed by standard alpha blending with the recursive *over* operator given in Equation (2.12). A problem of this operator is that the resulting color and opacity depends on the order in which the volumes are applied. This is similar to the multiplicative Fuzzy Logic described in Section 3.5.2, were the last accumulated value has the strongest weight. For this reason other operators have been proposed which calculate a weighted sum of the single contributions. An example is the *inclusive opacity* operator of Cai and Sakas where the color values are weighted with their normalized opacity:

$$c_{out} = \sum_{i=1}^{n} \frac{\alpha_i}{\alpha_{sum}} c_i \quad , \qquad \text{with} \qquad \alpha_{sum} = \sum_{i=1}^{n} \alpha_i \quad . \tag{4.1}$$

Depending on the used multi-slice accumulation algorithm, different sets of GPU shader programs have to be generated. This is performed on the fly by the shader generator (see Section 4.4), at the first time a certain combination of volumes appears and then the shader program is stored in a map for later re-use. So, only if the render graph configuration changes or another slicing technique is chosen, the shader programs are discarded and re-generated during the rendering of the next frame.

### 4.2.2 Multi-Volume Raycasting

While texture slicing and volume splatting were especially developed for GPU-based rendering, raycasting does not fit directly to rasterization hardware. Nevertheless, since raycasting is the most native way of volume rendering and leads to superior results, there have been developed several raycasting approaches which strive to exploit the parallel rendering capabilities of a GPU. The basic idea of these approaches is to render the front faces of the volume's bounding box and then cast the viewing ray for each rasterized fragment. On older graphics hardware this had to be done in multiple rendering passes [83; 125], but latest GPUs provide loops and dynamic branching and by this they allow the implementation of the whole ray traversal in a single shader [144].

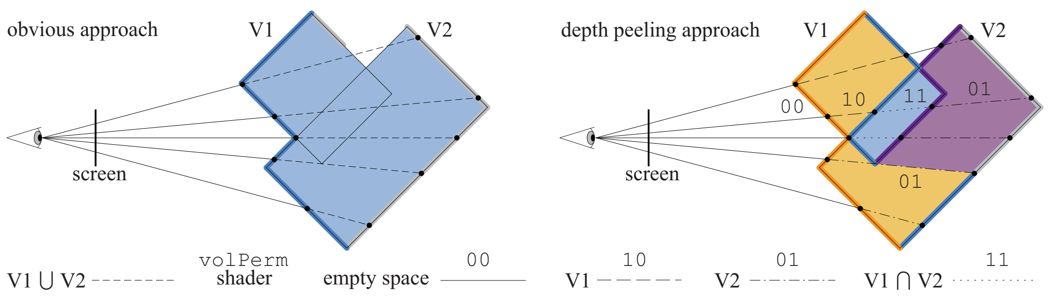


Figure 4.2: Two approaches for multi-volume raycasting of a two-volume scene (*V1, V2*): On the left side the first (*dark blue*) and the last (*grey*) intersection points of the viewing rays with *V1* and *V2* are computed and then these rays are traversed by a single multi-volume shader responsible for both volumes. For the approach on the right side the scene is segmented into three layers by depth peeling. Each layer is rendered with optimized shaders which take only the covered volumes (*volPerm*) into account. The first layer (*orange*) consist of two disjoint regions of *V1* and *V2* which are treated by two different shaders. (Images courtesy of Rößler *et al.* [123], ©2008 IEEE).

The straightforward approach for raycasting of multiple volumes on the GPU can be realized by extending the single pass method for a single volume to a three pass rendering method (see Figure 4.2 (left)). First the front faces of all volumes' bounding boxes are rendered with activated depth test and the depth function set to less. For each viewing ray this yields the first entry point to a volume in the scene. The coordinates of these entry points are stored in a texture. In the second pass, the back faces of the bounding boxes are rendered with the depth function set to greater. This generates the exit points from the furthermost volumes along the rays. Now that the entry and exit points for the union of all volumes are known, the whole multi-volume scene can be visualized in a third pass by rendering a screen-filling quad to initialize the rays and applying a single fragment shader. This shader reads for each pixel the pre-computed entry and exit points and traverses the viewing ray between these points in front-to-back order. At each sampling position along the ray, the shader evaluates the color contributions of all volumes in the scene and accumulates them to a single sample color, which is then blended to the ray's output color. The major disadvantage of this approach is that for each sampling point along each ray each volume is evaluated, even if the sampling point lies outside a volume's bounding box, or the ray traverses empty space. Alternatively, it can be tested if a sampling point lies inside the volume before the evaluation is done, but this introduces a huge number of expensive branching operations especially for complex scenes.

For these reasons a more sophisticated multi-volume raycasting technique (see Figure 4.2 (right)) is desired that requires a slightly different algorithm work flow.

```
1  void raycastingCPU() {
2    activateShader(firstLayerGPU);          // generate first depth layer
3      for each volume in scene {
4        volBit = 1 << volNum;
5        renderFrontFaces(volBit);
6      }
7
8    list prevPermList, curPermList;         // init lists of permutations
9    curPermList.add(0);
10
11   while (layerNum < maxNumOfLayers) {     // loop over depth layers
12     activateShader(nextLayerGPU);         // generate next depth layer
13     for each volume {
14       volBit = 1 << volNum;
15       renderBoundingBox(volBit);
16     }
17
18     prevPermList = curPermList;           // check previous permutations
19     curPermList.clear();
20     for each prevPerm in prevPermList     // perform raycasting for volPerm
21       for each volPerm in singleBitFlip(prevPerm) {
22         activateShader(raycastingGPU, volPerm);
23         renderScreenFillingQuad();
24         if (anyFragmentWritten())
25           curPermList.add(perm);
26       }
27   }
28 }
```

Listing 4.1: Pseudocode for the raycasting procedure on the CPU. The shaders (*red*) are given in listings 4.2 and 4.3. (Code courtesy of Rößler *et al.* [123], ©2008 IEEE).

The implementation of this work flow leads to several challenges that are best understood by following the pseudocodes in Listings 4.1, 4.2 and 4.3. The algorithm first divides the multi-volume scene into depth ordered segments of intersecting volumes and then applies to each of these segments an optimized shader that only takes the currently involved volumes into account. The segmentation of the scene is done by depth peeling. As shown in Listing 4.1 (line 6), the depth peeling starts with rendering the entry points of the rays in the same way as it is described for the approach above. Then the volume bounding boxes are rendered once again with depth test set to less and a special shader applied (Listing 4.2, nextLayerGPU), which peels away the first entry points by comparing their stored z-values with the z-values of the currently rendered fragments. This generates for each viewing ray the second intersection point with a volume bounding box. The following intersection points are computed in the same way by taking the previous intersection points as new start points and again applying the nextLayerGPU shader. The highest possible number of intersections per ray for $n$ volumes is $2n$, since there can be maximally $n$ entry points and $n$ exit points.

The described depth peeling algorithm generates layers of ray segments that

```
 1  in:  vec3 curPos; uint volBit; bool frontFace;
 2  out: vec3 pos; uint volPerm;
 3
 4  void firstLayerGPU() {
 5    pos = curPos;                          // write position and
 6    volPerm = volBit;                      // volume bit vector
 7  }
 8
 9  void nextLayerGPU() {
10    vec4 prevPos; uint prevPerm;           // read previous layer values
11    readPreviousValues(prevPos, prevPerm); // from textures
12
13    if (curPos.z < prevPos.z)              // discard fragment in
14      discard;                             // front of previous layer
15
16    if (frontFace)                         // compute current permutation
17      curPerm = prevPerm | volBit;         // on enter layer
18    else
19      curPerm = prevPerm & ~volBit;        // on exit layer
20
21    pos  = currentPos; volPerm = curPerm;
22  }
```

Listing 4.2: Pseudocode for the computation of a ray segment layer on the GPU. (Code courtesy of Rößler *et al.* [123], ©2008 IEEE).

are bounded by two consecutive intersection points. So, each segment traverses a constant set of overlapping volumes. However, these sets of volumes can differ across a ray segment layer, as can be seen in Figure 4.2 (right), where the orange layer consists of two disjunct regions of volume V1 and volume V2. This means that there is no single shader which can be applied. Instead, a ray segment layer has to be further divided in screen space into regions of equal ray segments. To easily determine which volumes are intersected by a ray segment, it is possible to exploit the integer arithmetic capabilities of NVIDIA's G8 series GPUs. In the depth peeling step a bit vector is applied to each ray segment which encodes the intersected volumes – i.e., the current volume permutation. Since $32$-bit integer values are used, the total number of volumes in a scene is limited to $32$ which is sufficient for common scenes. The permutations are stored in an integer texture which is initialized with $0$. The permutation of the current ray segment layer is computed by incrementally changing the permutation of the previous layer. For each bounding box the id of the corresponding volume – encoded as bit vector `volBit` – is given as uniform to the depth peeling shader. If the currently rendered bounding box face is a front face a rendered fragment represents a point where the volume is entered, so the new volume permutation `curPerm` is computed from the previous permutation `prevPerm` by appending `volBit` with bitwise OR "|", (see Listing 4.2 (line 18)). At back faces the viewing ray is leaving the volume. Here the new volume id is subtracted by merging the previous permutation with the bitwise complement "$\sim$" of `volBit` by bitwise AND "&", (Listing 4.2 (line 19)).

```
in: uint shdrPerm, float smplDist;
out: vec4 outCol;

void raycastingGPU() {
  vec3 pos, strtPos, endPos, step; vec4 smplCol;

  if (shdrPerm != getSegmentPerm())             // discard if shader and
    discard;                                    // ray segment are unequal

  readStartAndEnd(strtPos, endPoint);           // compute sampling step
  step = norm(endPos − strtPos) ∗ smplDist;     // along the ray

  initInterpolatedVars();
  initFrontVars();

  pos = strtPos;
  outCol = vec4(0,0,0,0);
  while (pos.z < endPos.z) {                    // ray traversal
    copyFrontToBackVars();
    computeOtherVars();

    smplCol = vec4(0,0,0,0);
    for each volume in shdrPerm {               // accumulate single
      smplCol = accum(smplCol, volCol[i])       // sample colors
    }
    outCol += (1.0−outCol.a) ∗ smplCol;         // perform blending

    iterateInterpolatedVars();

    pos += step;                                // next sampling position
  }
}
```

Listing 4.3: Pseudocode for raycasting of a ray segment layer on the GPU. The green functions are replaced by generated code due to the render graph configuration and due to the current volume permutation. (Code courtesy of Rößler *et al.* [123], ©2008 IEEE).

Basically, a ray segment layer is rendered several times by specialized shaders for each possible permutation of volumes (Listing 4.3). To avoid unnecessary computations for not affected ray segments each shader first loads the ray segment's volume permutation and tests it against the permutation considered by the shader itself. If they are not equal, the execution of the shader is directly discarded. Since all ray segments that pass the same overlapping volumes usually cover connected regions and dynamic branching is efficiently done on current GPUs for coherent fragments, the overhead of these tests is relatively low. However, the number of shaders which have to be executed per layer is $2^n - 1$, which is the total number of permutations minus the zero permutation, where no volume is covered. This number becomes quite large even for small numbers of volumes. But it can be remarkably reduced by exploiting the fact that the volume permutations covered by the current ray segment layer depend on the permutations covered by the preceding layer. At the segment border of a single viewing ray the corresponding volume permutation changes only at a single bit, because either a new volume is entered or an old one is left. For the whole segment layer this means that only those volume permutations have to be tested that can be generated from the permutations covered by the previous layer by single bit flips (Listing 4.1 (lines 22-32)). To determine which permutations have been covered by a ray segment layer hardware supported occlusion queries are used. For each tested shader a query is started which returns whether any fragment was written to the frame buffer and by this if any ray segment has covered the corresponding permutation.

## 4.3 The Render Graph Framework

The graphical user interface (GUI) of the system consists of three views (see Figure 4.3 (b)). The major *render view* shows the visualization results due to the actual scene and render graph configuration and provides interactive manipulation of the camera position with the mouse. The two other views are placed above each other on the left hand side of the render view. The lower *render graph view* presents the render graph as a hierarchical tree. The graph in a whole can be manipulated by appending and removing nodes. It is also possible to insert nodes at a higher level of the graph, because removed branches are stored in a clip-board and can be re-inserted later. If a node of the render graph is selected, the above *render node view* shows an individual dialog for the manipulation of the nodes' individual parameters. Changes in the node or graph view are directly mapped to the underlying render graph and the resulting effects to the multi-volume visualization are shown immediately in the render view.

The implementation of the multi-volume visualization techniques (e.g., volume shading, clipping and non-photorealistic rendering methods) was embedded into the above mentioned framework, which supports dynamic shader generation for both slice-based multi-volume rendering and multi-volume raycasting, to ease the construction of multi-volume scenes for the user. Further on, it is possible to extend the system to new techniques by implementing and integrating new render nodes. The design of this object oriented system is based on C++, OpenGL and GLSL. The framework consists of four major components: the *scene description*, the *render graph*, the *multi-volume renderer*, and the *shader generator*. The fact that accumulation level intermixing permits an arbitrary combination of shading styles and transfer functions leads to a large number of possible visualization algorithms for a given multi-volume scene. Further on, the volume intermixing is performed on a per-sample level, which implies – in the context of GPU-based rendering – that for each of the visualization algorithms a specialized GPU shader has to be provided. Usually this complexity is coped with by providing a set of fixed shaders for predefined combinations of volumes and shading techniques.

The so called *render graph* was developed in order to overcome these restrictions and to provide full flexibility in GPU-based multi-volume rendering. This graph describes a complex multi-volume shading algorithm by the combination of several *render nodes*. Thereby, each render node describes a certain part of the whole shading algorithm and the final shader code is automatically generated by the system due to the actual graph configuration. Unlike a classical scene graph, which permits the creation and manipulation of complex scenes, the render graph describes the visualization of a given multi-volume scene on the level of shading a single multi-volume sample, independent of the finally applied volume rendering technique. There exist three basic types of render nodes which represent different stages of the shading process.

### 4.3.1 The Scene Node

The root of the entire render graph is always defined by a single *scene node* that represents the interface between the external description of the scene objects (i.e., the camera, light sources and volumes) and the graph itself. Therefore, this node collects the required information from these objects and passes it on to its children.

### 4.3.2 Structural Nodes

Starting from the scene node, all volumes are initially treated equivalent, regarding the shading process. To allow a separate handling of different volumes as a whole or just parts of them, *structural nodes* are introduced. These nodes do not directly contribute to the shading result; rather they provide capabilities to dynamically control the evaluation of the render graph by branching and manipulation. Three kinds of structural nodes are supported:

***Splitter Node*** The *splitter node* is used to divide the handling of the volumes into several branches. Therefore, an arbitrary number of groups can be created, where each group contains one or more volumes. Moreover, a volume can be placed in several groups simultaneously. Every group results in a new branch of the render graph. Thus, it is possible to define different rendering styles for different volumes or to combine several rendering styles for a single volume. This can be considered as a branching on object level.

***Conditional Node*** In contrast to the splitter node, a *conditional node* performs a subdivision of the volume objects themselves. This means, during the rendering process only the branch is chosen for which the condition is true. These conditions are normally evaluated on basis of the actual fragment position and could for example describe the selection of a segmented structure or the clipping against an implicitly given geometry like a plane.

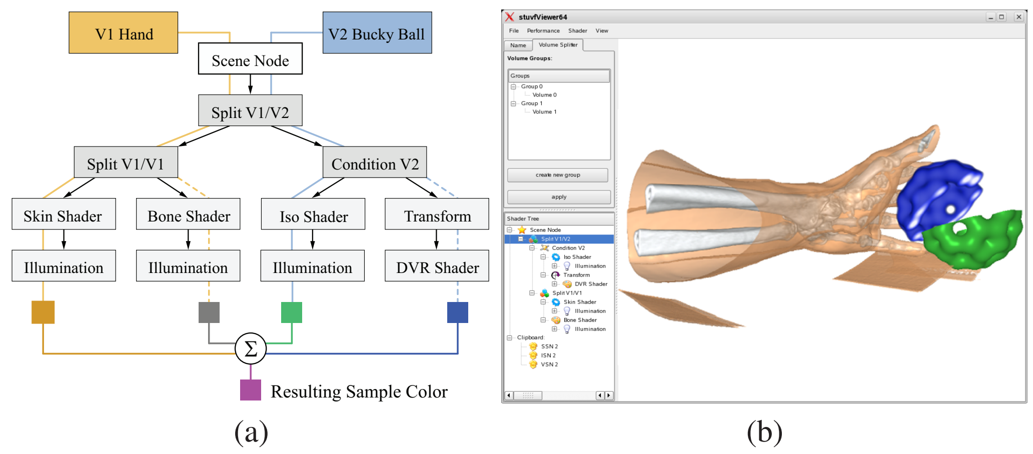


Figure 4.3: The abstract render graph structure (a) represents a scene of two volumes with different rendering styles applied. The resulting image of the applied render graph to the dual-volume scene (b). (Images courtesy of Rößler *et al.* [123], ©2008 IEEE).

***Transformation Node*** To spatially separate whole volumes or previously subdivided parts of them, it is possible to insert a *transformation node* into the render graph. This node implements an affine transformation which is applied to all volumes that are assigned to the actual branch. Thereby, volume displacement can be realized.

### 4.3.3 Shader Nodes

The third kind of nodes are the *shader nodes*, which exclusively implement low-level shading operations to compute the resulting image. The shader nodes can be placed anywhere in the render graph, and several shader nodes can be cascaded on a path from the root down to a single leaf of the graph. In this case, the successor either overwrites or manipulates the result of its preceding shader node.

### 4.3.4 A Render Graph Example

The abstract functionality of the render graph and its nodes gets clearer on the structural example for the computation of the resulting sample color in correspondence to Figure 4.3 (a). Render nodes are represented by grey boxes. The colored lines describe the paths of the volumes, the black arrows indicate the parent-child relationship of nodes. Applying this graph to the given dual-volume scene, with volume *V1* as the hand dataset and volume *V2* as the bucky ball, results in the image shown in Figure 4.3 (b).

Starting at the scene node, which handles the attached volumes, their path leads through the graph in a top-down manner. The first splitter node *Split V1/V2* divides the paths of both volumes into two branches, where the hand volume takes the left branch and the bucky ball takes the right one. Volume *V1* hits another splitter node *Split V1/V1*. This node virtually splits the single path of the volume into two independent branches that both work on the same hand volume, but lead to different shader nodes, indicated by the continuous lines and the dashed lines. The *Skin Shader* node in the left branch is responsible for the semi-transparent isosurface rendering of the skin, while the *Bone Shader* node in the right branch performs a direct volume rendering of the bone structure in Figure 4.3 (a). Both nodes are succeeded by illumination nodes that manipulate the previously calculated fragment sample color with lighting computation.

Investigating the right branch of node *Split V1/V2*, volume *V2* encounters a conditional node *Condition V2*. This node splits the bucky ball into two halves using a clipping plane. The left branch hits an *Iso Shader* node, followed by an illumination node, resulting in a lighted isosurface. The right branch runs into the transformation node *Transform*, translating and rotating this half, before the direct volume rendering node *DVR Shader* serves the unlighted color for this path. Finally, the contributions of the different branches are mixed according to the defined accumulation operation.

## 4.4 Dynamic Shader Generation

The goal of dynamic shader generation is to convert the abstract representation of the render graph into an especially adapted GPU-based shader program. Here, the high-level shader language GLSL [124] can be used for hardware accelerated rendering. The advantage of GLSL (and other high level shader languages) is that it permits the generation of structural C-like code which is automatically optimized by the compiler with respect to the current hardware configuration.

The basic idea of the shader generation approach is that a single render node acts as a container that stores all needed information and dependencies to perform its desired task. Therefore, it provides a set of output variables that either act as input for succeeding render nodes or as final output value for the actual volume sample. For each of these output variables the following information has to be given:

1. *Name and type:* A unique name and a data type to permit correct access by other render nodes.
2. *Shader code part*: Predefined code that implements the computation of the output variables.
3. *Input variables*: Output variables from previous render nodes on which the computation of the actual output variable is based on.
4. *Externals*: External parameters and textures that are needed for the output computation. They are passed to the shaders as uniform variables.

5. *Shader type*: An output variable can either be calculated per vertex in the vertex shader and interpolated by the graphics hardware, or it has to be calculated per fragment in the fragment shader.
6. *Scope*: A variable can either be valid for the whole scene, for a certain transformation, or for a specific volume.

Which output variables a render node serves, depends highly on its type (see Section 4.3). The scene node for example provides all information of the given multi-volume scene to the other nodes of the graph. This is e.g., the camera matrix or the position of the light source. Additionally it provides the sample position and the volumes' scalar values, gradients and curvatures at this position. To facilitate complex shading algorithms like pre-integration or isosurface shading, these values are also provided for the succeeding sample position along the viewing ray.

A shader node generally computes the sample color for a single volume. There are two major types of shader nodes: Those which compute the resulting color directly from the actual volume sample, e.g., direct volume rendering or isosurface shading; or those which manipulate the previously computed sample color to apply for example illumination or ghosting effects.

Structural render nodes do not directly contribute to the rendering result, which means that they usually do not provide any output variable that can be used by a succeeding node. Nevertheless, condition nodes have to provide a Boolean condition variable for each outgoing branch, which indicates, if the related branch should be evaluated due to the applied condition. Furthermore a condition node may manipulate the actual volume gradient with respect to the distance to the clip surface as proposed in [164], to get correct illumination results in succeeding computations.

### 4.4.1 Two-pass Shader Assembly

Based on the definition of output variables, the related shader code and the dependencies on input variables, it is possible to generate a specific shader program for computing the final color of a multi-volume sample. Therefore, the shader generation process is divided into two passes. The first pass evaluates the graph and determines all output variables which have to be computed for the requested sample color. This information is stored in the so-called *variable state*, which is a structural copy of the render graph that holds only the currently used variables and links to the original render graph nodes. In the second pass the pre-computed variable state is used to combine the associated shader code parts for the final shader program.

***1st Pass*** The render graph is traversed in depth-first order to collect the variables, and for each render node an associated *variable state node* is created. When a leaf node of the render graph is reached, it is tested whether it can provide the sample color and if so, this variable is stored in the related variable state node. Furthermore, the applied input variables are stored in a list of required variables. On the way back to the root of the graph, this list is re-investigated for each passed node and servable variables are replaced by their associated input variables. If the actual render graph configuration defines a valid shader program, the list of required variables will be empty in the end.

Branches of the render graph – originating from condition and splitter nodes – are evaluated independently on the way down. At the backward traversal the different lists of required variables are re-merged to a single one. In addition, at splitter nodes it is determined which volumes are investigated at the different branches. Since the sample colors only have to be computed for still active volumes, this information is additionally stored in the related variable state at the leaf node and propagated to the required input variables. At conditional nodes, for each outgoing branch the related conditional variable is added to the required variables list and then processed just like the others.

A transformation node plays a special role in the variable gathering pass. All variables with volume or transformation scope that are computed on the succeeding branch, have to be adjusted due to the defined transformation. The same has to be done for the required variables on the way up to the root. Additionally, if the same volume is examined multiple times on different branches with different transformations, it effectively has to be rendered multiple times at different positions. Thus, even the preprocessing – e.g., the slicing for slice-based volume rendering – is affected. To cope with this fact, all volumes that are actually active on a transformation node's branch are shallowly cloned, which means that the clones point to the original volumes, and additional transformation matrixes are attached. Furthermore, the active volumes at the outgoing branch of a transformation node are replaced by their related clones. In the subsequent processing steps of shader generation and rendering, all volumes in the scene – originals and clones – are treated equivalent.

***2nd Pass*** While the definition of render nodes and the generation of variables do not depend on a certain rendering technique, the code generation pass produces shader programs that are especially adapted to the applied rendering algorithm. As described before, the system supports slice-based volume rendering (see Section 4.2.1) and multi-volume raycasting (see Section 4.2.2), and creates a corresponding vertex/fragment shader pair for the rendering of either a single multi-volume slice, or a multi-volume ray segment. It is possible to generate a single shader program that processes all volumes in the scene at once, or separate shader programs for single volumes or programs for any combination of them.

To assemble a shader program for *slice-based multi-volume rendering* for a given set of volumes, the pre-computed variable state is traversed in depth-first order. At each variable state node the shader code parts that are associated with the stored variables are taken and, depending on the variable's shader type, either added to the vertex or to the fragment shader. If a variable that is computed in the fragment shader, depends directly on a variable in the vertex shader, this input variable is served automatically to the fragment shader by a varying variable. However, the number of varying components that can be used in a single GPU program is limited and depends on the graphics hardware. To overcome this restriction, the number of potential varying components is counted before assembling the shaders and, if the limit is exceeded, vertex shader variables are computed in the fragment shader as well. If a variable has either transformation scope or volume scope, its code segment is defined only once by the render node, but is appended to the shader several times for each requested transformation and/or volume respectively. To ensure the distinction of the different computations, the variable names are additionally extended by a unique per-volume postfix.

If there are branches in the render graph, the shader code is assembled independently for each branch and finally combined. In order to avoid unnecessary computations, the code parts of variables of previous render nodes are always added to the shader as late as possible. If a variable is used in all outgoing branches of a structural node, it is placed before branching, but if it is only needed for a single branch, it is computed inside exclusively. Conditional branches are evaluated, if the associated conditions – represented by Boolean condition variables – are satisfied. This is realized by nesting the branches inside *if*-statements. If a transformation node is placed somewhere below a conditional branch and the branching condition depends on the transformation, for each leaf of the outgoing subtree, the condition has to be evaluated as an independent if-branch in the shader code. After the traversal of the variable state graph, code for the accumulation of the color contributions of the different volumes at all branches is added to the fragment shader, to compute the final multi-volume sampling color. The generated GPU program is now ready for direct use in slice-based multi-volume rendering, which is presented in the next section.

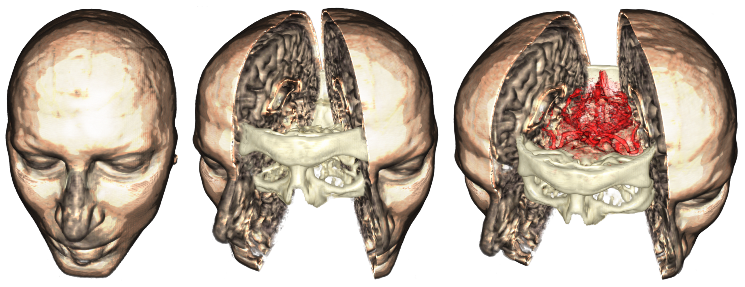
The shader code for the slice-based approach computes a multi-volume sampling color per fragment on a slice, which is then blended to the framebuffer. On the contrary, *multi-volume raycasting* is implemented as single pass algorithm [144], computing several multi-volume samples along a ray segment that have to be blended accumulated the shader, before blending to the buffer. For this purpose, an additional loop needs to be added to the shader to realize the ray traversal. To assign each ray to an unique combination of volumes, the volume bit vector of the ray segment is compared to the shader's bit vector at first, in terms of depth peeling. Then, start and end point of the ray segment is read from the previously computed textures. Finally a loop is executed which traverses the ray segment from start to end point with a predefined sampling distance. The code inside the loop is dynamically generated due to the render graph configuration, just as it is done for the slice-based approach. In the end of the ray traversal loop, code for the accumulation of the color contributions of the different volumes is added to compute a single multi-volume sampling color, which is then blended to the output color of the ray segment.

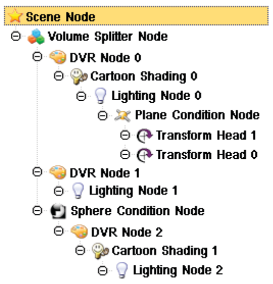
The generated shader code can be further optimized. First, several of the output variables – e.g., the volume texture coordinates – don't have to be computed at each sampling point by their complex predefined operations, but be interpolated instead from initially computed sample values. For this, an initialization step before the ray traversal loop is added, where the variable values for the first and the second sampling point along the ray segment are pre-computed and the sample-to-sample step size is calculated by subtracting these two values from each other. Inside the loop the step size is used to generate a new variable value from the previous one by incrementation. A further potential for optimization is given by the fact that several volume shading algorithms, like pre-integration or implicit isosurface rendering, do not calculate a color for a single sample position but for the whole slab between two samples. This means that some variables have to be computed for the current sampling position and for the following one as well. Instead of re-computing both values for each sampling step it is sufficient just to compute the new value for the current back sample and to copy the value for the front sample from the back sample of the previous step. Additionally, the front value has to be initialized before the ray traversal.

## 4.5 Application Cases

A typical scenario for multi-volume rendering is medical visualization and illustration. On the one hand, several scans with different imaging modalities are often taken of a single patient, to get comprehensive information about anatomical structures and functional processes. On the other hand, a single dataset usually contains a couple of different structures which can best be emphasized by different transfer functions and shading techniques. Thus, to demonstrate the applicability and flexibility of the presented multi-volume rendering framework, various techniques are applied to several different medical use cases, which are detailed in the following. The visualization results of the five example setups and the corresponding render graphs are shown in Figures 4.4, 4.5 and 4.6 respectively.

***Diagnosis*** The first use case is an example in the area of neuroradiological diagnosis for the detection of malformations of cerebral blood vessels. Therefore a CTA scan of the patient's head is taken, which is a CT technique, where a con-





```
Scene Node
⊖ Volume Splitter Node
    ⊖ DVR Node 0
        ⊖ Cartoon Shading 0
            ⊖ Lighting Node 0
                ⊖ Plane Condition Node
                    ⊖ Transform Head 1
                    ⊖ Transform Head 0
    ⊖ DVR Node 1
        ⊖ Lighting Node 1
    ⊖ Sphere Condition Node
        ⊖ DVR Node 2
            ⊖ Cartoon Shading 1
                ⊖ Lighting Node 2
```

(a) (b) (c) (d)

Figure 4.4: Example setup I: Combination of a CTA (Computed Tomography Angiography) dataset and a related MRI (Magnetic Resonance Imaging) dataset of a human head. The MRI dataset provides the skin and brain tissue. It is vertically cut and the two halves are moved away from each other to get insight to the inner structures. The CTA dataset contains the skull and the vessels which are rendered with different transfer functions. Images (a-c) show three stages of an interactive multi-volume visualization session and image (d) represents the render graph which corresponds to the final configuration in (c). (Images courtesy of Rößler *et al.* [122], ©2008 IEEE).

trast agent is injected to emphasize the vessel structure in the resulting images. In addition, an MRI scan that accentuates the brain tissue is used to get the patient specific relationship between the blood vessels and the anatomical structure of the brain. Note that the CTA and the MRI scans are co-registered.

Figure 4.4 shows an example visualization (setup I) of this two-volume scene. Images (a-c) illustrate different visualization steps during the render graph configuration, and image (d) shows the render graph that corresponds to the final result in (c). The goal of the visualization is to present the intracranial brain vessels in relation to the surrounding skull and in the context of the brain structure. Therefore, the visualization path of the two volumes is first split into one branch for the MRI volume and two branches for the CT volume by a *splitter node*. Then, the MRI volume, which contains the skin and the brain tissue, is rendered with a *direct volume rendering (DVR) node* and the surface structure is emphasized by the combination of a *cartoon shading node* and a *lighting node* which performs Phong shading. To get insight into the inner structures, the MRI head is divided vertically by a *plane condition node* and the two halves are rotated and moved away from each other by two *tranformation nodes*. The first branch of the CT volume is responsible for the visualization of the skull. For this purpose a *DVR node* with a transfer function that extracts the bone tissue and a standard Phong shading is applied. On the second CT branch the vessel structure inside the skull is extracted. Primarily, a so-called *sphere condition node* is applied, which approximates the brain volume by a sphere and cuts away all vessels outside this sphere. Then, a *DVR node* with a transfer function for the vessels is attached and

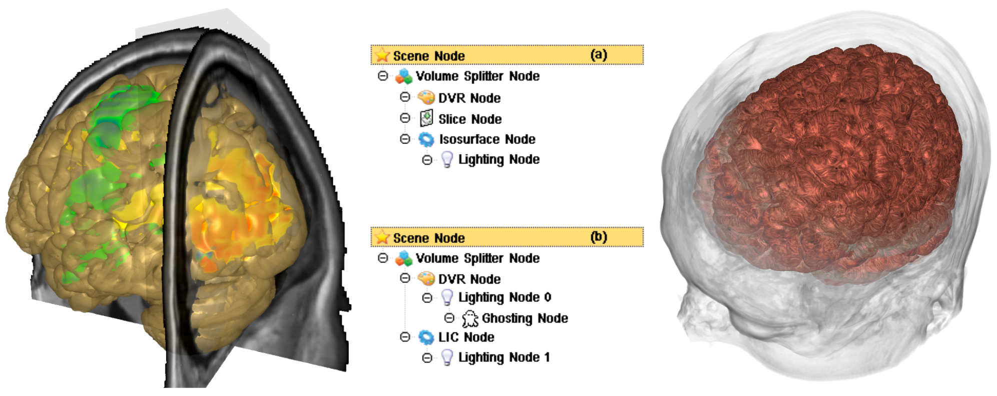
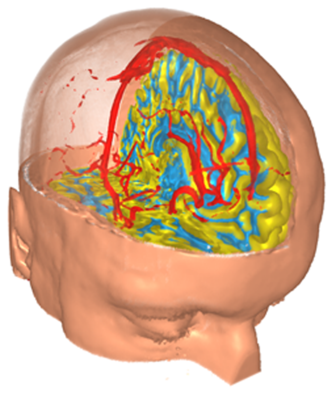


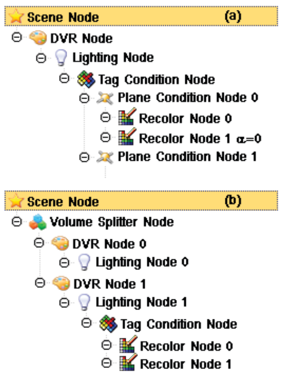
Figure 4.5: Example setups II and III with the corresponding render graphs: setup II in image (a) shows a DVR shaded fMRI dataset combined with the anatomical brain MRI data rendered as illuminated semi-transparent isosurface and two 2D slices of the whole head as context information, setup III in image (b) shows the combination of an illuminated DVR shaded MRI head with a ghosting method applied to see the inside. The interior brain is rendered as illuminated isosurface with a 3D LIC computation applied, to emphasize the curvature. (Images courtesy of Rößler *et al.* [122; 123], ©2008 IEEE).

finally the vessels are emphasized by cartoon and Phong shading.

***Analysis*** Another application area of medical multi-volume visualization is the analysis of functional processes inside the human body that are measured by functional imaging methods, e.g., fMRI (functional MRI). This is a special MRI technique that measures the activation of the brain. It is for example used by cognitive neuroscientist to study the relationships between cognitive tasks and the involved brain regions. To get an impression of the localization of the functional processes, the functional brain images are usually fused with a corresponding anatomical reference dataset.

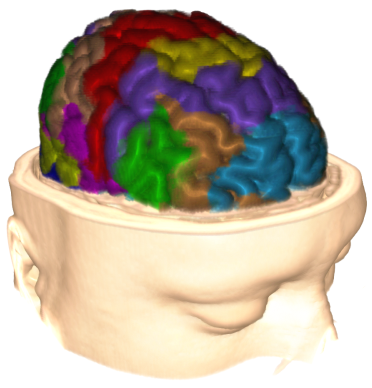
An example of fMRI volume visualization is given in Figure 4.5 (a) (setup II), which shows a multi-volume scene consisting of an fMRI activation dataset, a corresponding anatomical MRI dataset of the head, and a third dataset which contains the explicitly segmented brain of the anatomical MRI volume. To visualize the relation between the three datasets, three different shading techniques are applied. The activation volume is rendered with DVR and a special transfer function that color codes the activation values. The brain dataset is visualized by an illuminated semi-transparent isosurface, to show the brain surface without occluding the activation inside. Finally, a single orthogonal slice of the complete MRI volume is extracted by a *slice node* to provide the anatomical context of the scene.





```
Scene Node                      (a)
⊖ DVR Node
    ⊖ Lighting Node
        ⊖ Tag Condition Node
            ⊖ Plane Condition Node 0
                ⊖ Recolor Node 0
                ⊖ Recolor Node 1 α=0
            ⊖ Plane Condition Node 1

Scene Node                      (b)
⊖ Volume Splitter Node
    ⊖ DVR Node 0
        ⊖ Lighting Node 0
    ⊖ DVR Node 1
        ⊖ Lighting Node 1
            ⊖ Tag Condition Node
                ⊖ Recolor Node 0
                ⊖ Recolor Node 1
```



(a) (b)

Figure 4.6: setup IV in image (a) shows a MRI dataset of a head that is segmented into different anatomical regions like skin, brain tissue and vessels. These regions are differently colored and partly cut away by two clip planes. For setup V in image (b) the upper half of another MRI scanned head is cut away, to lay open the brain, which is segmented and colored due to a functional brain atlas. (Images courtesy of Rößler *et al.* [122; 123], ©2008 IEEE).

***Illustration*** Illustrative volume rendering techniques become more and more important in medical volume visualization (e.g., [16; 53]), because they permit us to emphasize significant information in the datasets, while nonrelevant information is suppressed. While the major application of illustrative volume rendering is the creation of illustrations for presentation and education, it can also be used for diagnostic analytic purposes.

Figure 4.5 (a) and Figure 4.6 (a,b) show three illustrative medical multi-volume visualizations that were generated with the framework. All visualizations present a two-volume scene of a MRI dataset of a human head and the explicitly segmented brain. In the first example (Figure 4.6 (b)) (setup III) the whole MRI volume is shaded with DVR and illuminated with Phong shading. Additionally, a *ghosting node* is appended, which subsequently increases the transparency of the fragments with respect to the center and radius of a predefined sphere. By this, the inside brain becomes visible, which is rendered as an illuminated isosurface with an additional 3D LIC (line integral convolution) computation on the surface to emphasize the curvature. This is a flow visualization technique, which is applied to the previously computed curvature field of the brain dataset.

In the second illustrative example in Figure 4.6 (setup IV), a simulated MRI dataset of the BrainWeb database [1] is visualized in combination with a corresponding segmentation volume. The segmentation volume assigns to each voxel a unique tag, indicating the tissue type which the voxel belongs to. First, the whole MRI dataset is shaded and illuminated with a grey value transfer function applied.

| | | setup I | setup II | setup III | setup IV | setup V |
|---|---|---|---|---|---|---|
| raycasting | | 10.8 | 30.4 | 12.3 | 20.5 | 11.5 |
| slice-based | merge | 16 | 123 | 7.7 | 30.5 | 24 |
| | separate | 40 | 201 | 8.4 | 30.5 | 28 |
| | intersect | 11 | 414 | 8.5 | 30.5 | 34 |

Table 4.1: Performance of all three multi-volume slicing techniques on a $512^2$ viewport given in frames per second (fps). Measurements have been performed on an NVIDIA GeForce 8800 GTS graphics board with 512 MB memory.

Then, a *tag condition node* is attached which takes the segmentation node as input. This node permits the definition of tag groups and for each group a new outgoing branch is generated. In the given example, branches for skin, skull, grey matter, white matter, and vessels are defined. To each of these branches *recolor nodes* are attached which multiply the previous grey values with a pre-defined color. In addition, skin, skull, grey matter, and white matter are partly clipped away by several *plane condition nodes*, each consisting of two clip planes. While skull and brain are completely removed by setting the alpha value to zero, the clipped skin is still rendered semi-transparent to give a feeling of the whole head's anatomy. The equal timings for the different slice-based approaches in Table 4.1 can be ascribed to the fact that this dataset consists of only one volume and a tagged volume that is used for branching and applying separate rendering styles; and therefore, all rendering operations can be accomplished in one single shader for all approaches, which respectively leads to the same rendering performance.

In the final illustrative example of Figure 4.6 (b) (setup V), the brain is subdivided into several functional regions due to a given brain atlas. Therefore, the MRI head is again shaded with DVR and illuminated with Phong, but with another transfer function as in the first example. The upper half of the head is cut away by a *plane condition node*, which is placed in front of the *DVR node*. The brain is initially shaded with DVR, with a grey value transfer function applied and also illuminated. Then, a *tag condition node* is attached which takes an additional tagged volume – the brain atlas – that assigns an unique region ID to each voxel. The tag condition node permits the definition of tag groups and for each group a new outgoing branch is generated. In the given example, *recolor nodes* are attached to each of theses branches which multiply the previous grey values of the fragments with a pre-defined color.

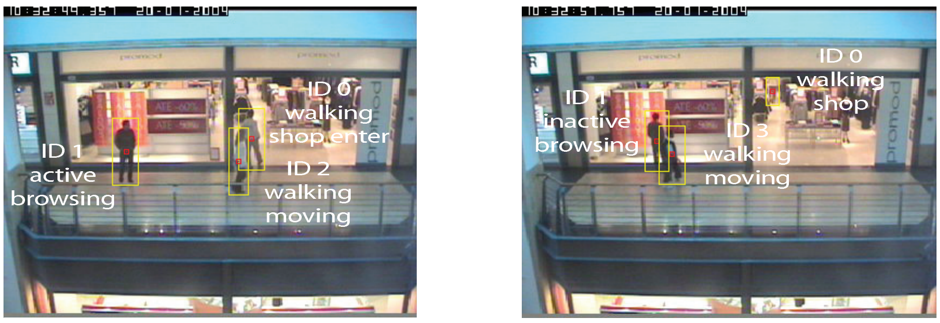
The rendering performance of the system was tested for each of the five example setups. Thereby, CTA and MRI datasets used for setup I have both a resolution of 256x256x120 voxels, the head and the brain dataset for setup II, III, and V have a resolution of 181x217x181 voxels, and the fMRI activation volume in setup II has a resolution of 40x48x34. Table 4.1 shows the measured results with respect to the presented multi-volume slicing techniques. Depending on the complexity of the applied render graph and the total number of volumes in the scene the advantages of the different techniques are accentuated. As can be clearly seen, the separation method dominates for most cases in terms of performance, but with the significant drawback of the restriction of the accumulation functionality to standard GPU blending operations. If more sophisticated accumulation functions are required, the two other slicing techniques are the only choice, which have the disadvantage of high cost for the additional tessellation. Nevertheless, the intersect method can even outperform the separate approach if the bounding boxes of all volumes are co-aligned, as can be seen for setup III and setup IV. This is caused by less effort for rasterization, since each sample is only processed once. For setup I (a) and setup II to V, merge is slower than intersect due to two reasons. First, each volume is sampled for the whole proxy geometry and unneeded samples are computed in non-overlapping regions. Second, it has to be tested for each sample whether it belongs to a volume or not. In setup I (c) the advantage turns over to merge, because of the exponentially raising effort for tessellating the overlapping proxy geometries, needed by the intersect approach. Summarizing, the choice of the slicing technique highly depends on the graph configuration and the desired quality of the visualization result.

Regarding the system's complexity, the effort for shader generation has also to be taken into account. It is linear with respect to the number of volumes and the number of render nodes, because each node has to be processed twice for each volume. Since the total number of volumes and render nodes is rather small, the generation time is minimal in contrast to the rendering performance. Another aspect is the complexity of the generated shader programs, which is also linearly increasing with the number of volumes and render nodes. Additionally, it depends on the single complexities of the applied shading algorithms, e.g., the LIC computation in setup II is very expensive and thus, highly effecting the frame rates. Nevertheless, the performance tests in the context of neuro sciences and neuro surgery have shown that the system provides interactive framerates even for complex scenarios. So, it fits well to a wide range of medical problems and allows clinical users to create meaningful and comprehensive visualizations in an intuitive way.

A minor drawback of this multi-volume rendering system is its scalability for a large number of intermixing volumes in a scene. One point is the use of the 32-bit integer to encode all possible volume permutations, which limits the total number of simultaneously processed volumes to 32. As the computation expense increases linear with the number of volumes, this would cause the system to lose the capability of interactive rendering. Another weak point is the limited hardware memory of $512$MB on current graphics hardware. Common volumes have a size of $128$MB, assuming they consist of $256^3$ voxels in integer precision. This reduces the number of volumes stored in hardware memory to four. However, for common multi-volume scenes, as they appear in medical applications this is sufficient.

# CHAPTER 5 MULTI-FIELD VIDEO VISUALIZATION

A video stream consists of a sequence of 2D frames that in general can be considered as 3D volume data, with one temporal and two spatial dimensions. Notionally, a video stream is analogous to many forms of digital signals (e.g., recordings of voice, electrical activity of the heart and seismic waves), except that it is composed of numerous interrelated pixel signals, and is inherently much more complex. Hence, dynamically processing and summarizing a video stream, and cost-effectively presenting a record of a video stream remain a huge challenge in video processing and visualization. One of these major difficulties is the fact that, in most videos, each 2D frame is the projective view of a 3D scene. Hence, a visual representation of a video volume on a computer display is, in effect, a 2D projective view of a 4D spatiotemporal domain. Because the third dimension of the video volume is the temporal dimension, simply visualizing a video volume using traditional volume rendering techniques is often inadequate in terms of extracting and conveying the most meaningful information in a video. For this purpose, automatic annotation [86] or encoding of additional information into the video frames could be considered, which is fundamentally an analytical process. Further, including the moving direction of dynamic objects in terms of flow visualization such as arrows or hedgehogs [34; 75] into the visualization could be of interest for an analyst. Other visual representations for flow rely on characteristic lines, such as streamlines, obtained by particle tracing. A major problem of 3D flow visualization is the potential loss of visual information due to mutual occlusion. This problem can be addressed by improving the perception of streamline structures [62], or by appropriate seeding [51]. All these aspects make video visualization a multi-field research problem that relates to video processing, volume visualization, flow visualization, and human factors in motion perception, which was not contemplated in this combination yet and requires new approaches for appropriate visualization.



(a) object frame (b) optical flow

Figure 5.1: Two frames of the “OneShopOneWait1front” dataset, hand labeled with the provided ground truth information. Both frames show three people located in front of a shop, ID0 entering the shop, ID1 browsing the shop window and ID2 passing by. (Images courtesy of Botchen *et al.* [9], ©2008 IEEE).

The fundament for video visualization was first introduced as a novel technique and application of volume visualization [30], and it was demonstrated that a spatiotemporal video volume can be used to aid the process of video editing [6]. In fact, video visualization reaches out to several other disciplines. A number of researchers have noticed the structural similarity between video data and volume data commonly seen in medical imaging and scientific computation, and have explored the avenue of applying volume rendering techniques to solid video volumes in the context of visual arts [40; 57; 76]. A lot of research is carried out to study video processing in the context of video segmentation [113; 139], and video surveillance [27; 29]. Systems have been developed under aspects of simultaneously wide-angle and detailed-view cameras [110], as well as for multiple spatially-related videos [157] that can be combined with environmental models, or fused to an augmented virtual environment [134]. While such research and development is without a doubt hugely important to many applications, the existing techniques for automatic video processing are normally application-specific, and are generally difficult to adapt themselves to different situations without costly calibration.

The work presented in this section takes a different approach from automatic video processing. As outlined in [152], it is intended to “take advantage of the human eye’s broad bandwidth pathway into the mind to allow users to see, explore, and understand large amounts of information at once”, and to “convert conflicting and dynamic data in ways that support visualization and analysis”. The objective of the proposed approach is to generate an effective multi-field visualization by combining volumetric scalar and vector data in order to extract and convey the most meaningful information in a video. The strategy is to use the capabilities of modern GPUs to synthesize interactive multi-field visualization.

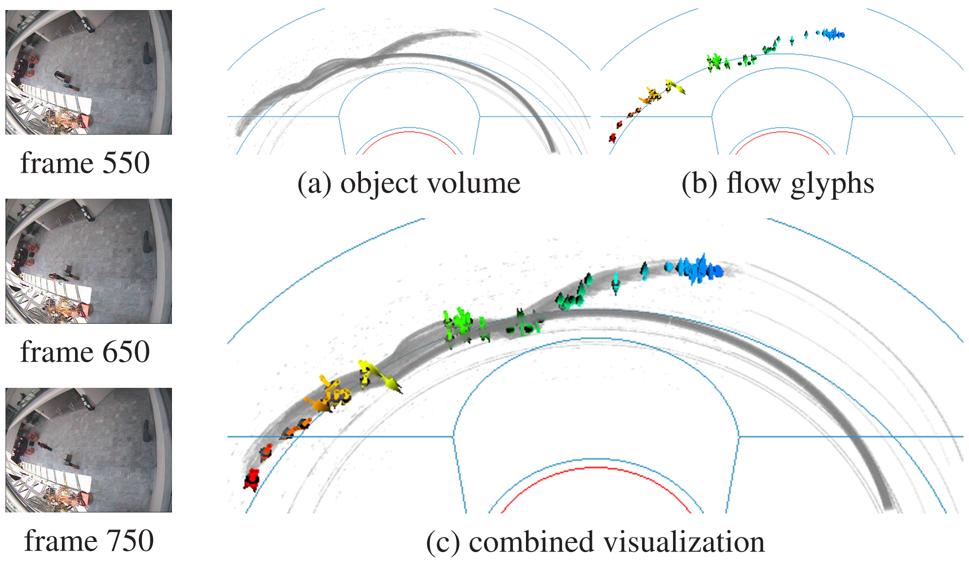


Figure 5.2: Left: Frames selected from the video clip "LeftBox". A woman deposits a box in the scene and leaves. Right: Volume visualization of extracted objects in a video in (a); flow visualization of an estimated optical flow of the same dataset in (b); and image (c) shows a combination of both visualizations. (Images courtesy of Botchen *et al.* [10], ©The Eurographics Association 2006).

Lets consider an example video stream captured by the CAVIAR project [41]. Figure 5.1 shows a presentation based on snapshots together with annotated texts that were hand labeled as the ground truth in CAVIAR. Even without contemplating the difficulties in developing a system that would produce reliable and comprehensive annotated texts dynamically in a variety of situations, the visual record exemplified by Figure 5.1 would require a lot of snapshots and a lot of texts in order for viewers to observe and comprehend the activities and events in the video. Further, it is difficult to recognize the spatial coherence of moving objects, evolving over a time interval of a video, by analyzing separated still images. This problem can be addressed by rendering the video data as large volume, as shown in the "LeftBox" example video clip in Figure 5.2 (a). Although the visualization adequately represents the objects extracted from the background scene, it does not provide sufficient motion features to allow the user to recognize that a moving object (i.e., a person) left a stationary object (i.e., a box) in the scene.

One possible approach to enhance the perception of motion is to estimate and visualize the optical flow in a video as flow glyphs, as shown in Figure 5.2 (b). However, the motion on its own cannot adequately convey the presence of objects in the scene. These observations indicate that the combined use of a volumetric scalar field (for the video data) and a vector field (describing the motion) might result in an effective video visualization. The combined visualization shown in Figure 5.2 (c) separates four stages of the video. In stage one, the person enters the scene with a box, i.e., the person is moving. In stage two, the person stops to deposit the box on the floor. This fact is clearly conveyed through a lack of flow glyphs. In the next stage, the person moves around the box. In stage four, the person exits the scene, but leaves the motionless box on the floor. The combination of volume and flow visualization gives the viewer a better understanding of both the information on location and motion of objects. In the next step, the basic approach on visual signatures is enhanced to a summarization and illustration method that can be used to present a record of video stream dynamically and cost-effectively. This extended method depicts a video stream as a series of continuing video volumes, which displays snapshots at relatively sparse intervals, and highlights automatically recognized actions with a set of visual mappings. This enables viewers to make a dynamic judgment of the semantics of an event, when the event is unfolding itself. The effectiveness of conveying and recognizing visual signatures of motion events in videos is supported by a major user study and a survey on visual effects described in Section 5.5.

The work described in this section was carried out in collaboration with Min Chen, Rudy R. Hashim and Ian M. Thornton from the University of Swansea, Greg Mori from the Simon Fraser University and Sven Bachthaler, Fabian Schick and Daniel Weiskopf from the Universität Stuttgart. So far, the ongoing work led to the following publications [9; 10; 24]. Rudy R. Hashim implemented the tool for the user study and accomplished this study described in Section 5.5 in the computer lab at the University of Swansea. Ian M. Thornton has to be accredited for the help in evaluating the user study. Sven Bachthaler and Greg Mori were responsible for the implementation of the action based video processing filter described in Section 5.2.4. Fabian Schick must be credited for parts of the implementation for the extended framework in Section 5.7.

## 5.1 Concepts and Definitions for Video Visualization

A video $V$ is an ordered set of 2D image frames $\{I_1, I_2, ..., I_n\}$. It is a 3D spatiotemporal dataset, usually resulting from a discrete sampling process such as filming and animation. The main perceptual difference between viewing a still image and a video is that a contemplator is able to observe objects in motion (and stationary objects) in a video. For the purpose of maintaining the generality of formal definitions, it is also necessary to include motionlessness as a type of motion in the following discussions.

Let $m$ be a spatiotemporal entity, which is an abstract structure of an object in motion and encompasses the changes of a variety of attributes of the object including its shape, intensity, color, texture, position in each image, and relationship with other objects. Hence, the ideal abstraction of a video is to transform it to a collection of representations of such entities $\{m_1, m_2, ..., m_k\}$.

Video visualization is thereby a function, $F : V \rightarrow I$ that maps a video $V$ to an image $I$, where $F$ is normally realized by a computational process, and the mapping involves the extraction of meaningful information from $V$ and the creation of a visualization image $I$ as an abstract visual representation of $V$. The ultimate scientific aim of video visualization is to find functions that can create effective visualization images, from which users can recognize different spatiotemporal entities $\{m_1, m_2, ..., m_k\}$ “at once”.

A visual signature $\mathcal{V}(m)$ is a group of abstract visual features related to a spatiotemporal entity $m$ in a visualization image $I$, such that users can identify the object, the motion, or both by recognizing $\mathcal{V}(m)$ in $I$. In many ways, it is notionally similar to a handwritten signature or a signature tune in music. It may not necessarily be unique and it may appear in different forms and different context. Its recognition depends on the quality of the signature as well as the user’s knowledge and experience. With these assumptions and the consideration that the effectiveness of abstract representations is well-accepted in many applications, it is more than instinctively plausible to explore the usefulness of video visualization, for which Daniel and Chen [30] proposed the following three hypotheses:

1. Video visualization is an (i) intuitive and (ii) cost-effective means of processing large volumes of video data.
2. Well constructed visualizations of a video are able to show information that numerical and statistical indicators (and their conventional diagrammatic illustrations) cannot.
3. Users can become accustomed to visual features depicted in video visualizations, or be trained to recognize specific features.

The initially developed video visualization framework described in Section 5.3 has the main aim to evaluate these three hypotheses. With a focus on visualizing objects and motion events in videos, the aspects for the design of this system include:

- To integrate automatic video processing techniques with video visualization in a framework for summarizing video streams visually and dynamically.
- To consider video visualization as a flow visualization problem, in addition to volume visualization.
- To introduce novel notions of visual signature for symbolizing abstract visual features with the focus to depict individual objects and motion events in videos.

- To compare the effectiveness of different abstract visual representations of motion events, including solid and boundary representations of extracted objects, difference volumes, and motion flows depicted using glyphs and streamlines.
- To conduct a user study that results in the first set of evidence for supporting hypothesis (3). In addition, the study provides an interesting collection of findings that helps to understand the process of visualizing motion events through their abstract visual representations.

Integrating these considerations into a visualization framework and evaluating the results with a user study led to several findings that served as a motivation to extend this approach to an enhanced video visualization system which incorporates object actions, as well as object relations in the illustrations to support the process of analysis. The aim of this extended system, described in Section 5.7, is to create a visual representation of a continuous video stream in a manner similar to an electrocardiogram (ECG) and a seismograph. Such visualization should convey both the raw imagery information of the video stream as well as processed information (e.g., extracted actions, recognized objects or detected events). The latter is usually application-specific. This visualization can serve a number of purposes:

1. *Fast temporal overview*. The visualization would make it easy for viewers to gain an overview of a temporal segment of a video without watching the video, or trying to piece together an overview from several disconnected snapshots.
2. *Focus highlighting*. The visualization would highlight specific processed information (as the focus) against the raw imagery information (as the context), and draws the viewers' attention to objects, actions or events that are of interest.
3. *Fault tolerance*. The visualization would enable viewers to identify errors in the processed information since automated vision techniques and statistical analyses are unlikely to deliver 100% accuracy.
4. *Long-term record*. The visualization could be used as a long-term visual record of the video stream as ECGs and seismograph are used.

Typical applications that could benefit from visualizations that provide these features for the analyst are systems including video surveillance cameras which produce a vast amount of video data, as typified by the shopping mall datasets. With the aim to quickly evaluate and analyze video streams that show actions and motion of persons, the contributions of the enhanced system in Section 5.7 consider:

- The implementation a system for the dynamic processing and visualization of action-based video streams, hence demonstrating the technical soundness of this strategy.

- To show a concept of visualizing actions and relations in video streams over a long time-span, based on a collection of short time-span techniques for action detection and relation estimation.
- To incorporate a focus+context design for multi-field visualization of raw and processed information of video streams, based on a set of visual mappings for highlighting multiple attributes including snapshots, object tracking, action classification, object relation and levels of plausibility.
- To introduce a new visual representation of a video stream, called Video-PerpetuoGram, as both a dynamic video summary and a long term abstract record.

Considering these concepts, the realization led to a video processing and video visualization system in Section 5.3 that in the first run was designed to study visual signatures, as evaluated in Section 5.5. With the findings of this study and a survey on visual effects the visualization framework and the rendering techniques were extended to include object actions and object relations into the illustrations as shown in Section 5.7. Before detailing the implementation and evaluation of such a visualization system, the next section gives an overview of applied video processing filters that can be used to extract information from the video streams to provide it for visualization.

## 5.2 Video Processing

An important issue for multi-field video visualization is the adequate extraction of additional information from the video stream that should be highlighted in the actual visualization process. This section describes object related filters for the video processing sub-system. These filters are computed in a pre-processing step and respective results are used at runtime to enhance the recognition of features in the video volume.

### 5.2.1 Video Transfer Function

The first type of filter is related to the visualization of the 3D scalar video volume. Volume visualization, in general, faces the fundamental issue of assigning an appropriate transfer function to the scalar 3D dataset, as described in Section 2.5. For the special case of video volume rendering, the transfer function is primarily used to highlight important regions of the video and to render unimportant regions transparent. This filter assigns opacities derived from a selection of different feature criteria defined in [30]. In a pre-processing step, a feature criterion is evaluated for the dataset and stored as an additional scalar dataset that can be considered as an importance volume. At runtime, the importance volume is assigned opacities, and possibly colors, by means of a 1D transfer function.

### 5.2.2 Optical Flow

In addition to a direct visualization of the 3D video volume, motion characteristics of people or moving objects in the video are of great relevance. One possible approach for the detection of moving objects in a video is to compute the optical flow of an image sequence with a gradient-based differential method [60]. The implementation used in this work is based on a modified version of the gradient-based differential method [5], which results in a velocity field $\mathbf{v} = (u, v)$ per video frame. Lets consider an image sequence as an intensity function $I(\mathbf{p}, t)$, where $\mathbf{p} = (x, y)$ is a position on an object in motion, and $t$ is the time variable. The translation of $\mathbf{p}$ with velocity $\mathbf{v} = (dx/dt, dy/dt) = (u, v)$ is thus

$$I(\mathbf{p}, t) = I(\mathbf{p} - \mathbf{v}t, 0) \quad .$$

A Taylor expansion of the above expression results in

$$I_x(\mathbf{p}, t)u + I_y(\mathbf{p}, t)v + I_t(\mathbf{p}, t) = 0 \quad ,$$

where $I_x$, $I_y$, and $I_t$ are the partial derivatives of $I(\mathbf{p}, t)$. This problem is not well posed with two unknown variables $(u, v)$. It is common to introduce further constraints in order to solve for $(u, v)$. Many proposed methods including [60], associate the above equation with a global smoothness term, and perform a cost minimization over a defined domain $D$:

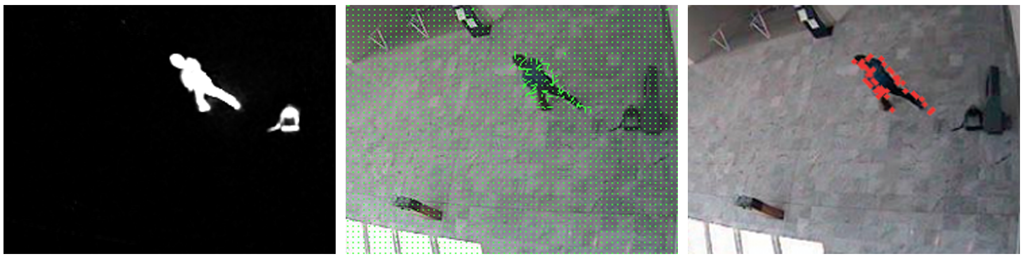
$$\int_D (I_x u + I_y v + I_t)^2 + \lambda^2 \left[ \left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 \right] \mathbf{dp} \quad ,$$

where $\lambda$ indicates the influence of the smoothness term, which, as suggested in [60], is set to 100 in this implementation. The velocity $\mathbf{v} = (u, v)$ is estimated by minimizing the above integral using an iteration process:

$$\begin{aligned} u^{k+1} =& \bar{u}^k - \frac{I_x(I_x\bar{u}^k + I_y\bar{v}^k + I_t)}{\alpha^2 + I_x^2 + I_y^2} \quad , \\ v^{k+1} =& \bar{v}^k - \frac{I_y(I_x\bar{u}^k + I_y\bar{v}^k + I_t)}{\alpha^2 + I_x^2 + I_y^2} \quad , \quad \text{with} \quad u^0 = v^0 = 0 \quad , \end{aligned}$$

where $k$ is the iteration number, $\bar{u}^k$ and $\bar{v}^k$ are the averages of $u^k$ and $v^k$, respectively, in a neighborhood domain.

The computed optical flow fields can be illustrated by the use of flow visualization techniques such as arrow plots and glyphs. Several glyph based flow visualization techniques can be found in literature and the reader is referred to the overview chapter in [165] for background reading on those methods. Another approach is based on the characteristic lines, such as streamlines, obtained by particle tracing. A major problem of 3D flow visualization is the potential loss of



(a) object frame (b) optical flow (c) frame seeding

Figure 5.3: Image (a) shows the difference object in the scene, computed from an empty reference frame. In (b), the optical flow of the frame is shown with green lines. In (c), seeds are generated based on the optical flow shown in image (b). (Images courtesy of Botchen *et al.* [10], ©The Eurographics Association 2006).

visual information due to mutual occlusion. This problem can be addressed by improving the perception of streamline structures [62], or by appropriate seeding as detailed in [51], discussed in the next section.

### 5.2.3 Seed Point Generation

To facilitate the visualization of the previously computed optical flow, it is necessary to determine a set of seed points for particle tracing or for positioning of flow glyphs. The filtering stage that generates seed points is implemented as a CPU program outside the later described video visualization rendering framework (VVR), in order to provide most flexibility in designing the seeding algorithms. Typically, the seeding stage uses the optical flow and the difference object to determine the seed points. Figure 5.3 (c) shows an example frame for seeding.

As detailed in Section 5.2.2, the 2D vector fields $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ are computed based on the intensity object fields $\{I_1, I_2, \ldots, I_n\}$. Then, the filter stage generates from the sequence of text files $\{S_1, S_2, \ldots, S_n\}$ a seed list for every frame. This seeding is designed as a 3-phase algorithm:

1. The algorithm determines a list of all eligible points in $\mathbf{v}_i$, with two control parameters: *grid interval* and *magnitude threshold*. With the *grid interval* parameter, the user can superimpose a grid on all the 2D vector fields and only grid points are eligible to be selected as seed points. With the *magnitude threshold* parameter, insignificant motion with a magnitude less than the threshold is filtered out.
2. The algorithm sorts the list of eligible seed points according to some criteria of visual importance, typically for instance, the magnitude of the motion vector at each point.
3. Finally, the algorithm selects a set of seeds from the sorted list. The user has the option to select *all* points, to select *the first* $N$ points, or to select

*randomly* $N$ points in the list. As the first phase usually produces a large list of seed points, which could lead to slow rendering as well as cluttering the visualization, this selection process allows the list to be trimmed down based on importance.

An example of the seeding process is given in Figure 5.3. Image (b) shows the object extracted from the background, while image (b) shows an optical flow field estimated with two consecutive video frames. Figure 5.3 (c) shows an example of a created seed list that was generated from the optical flow in Figure 5.3 (b), using the above algorithm.

In case of multiple objects appearing in a scene, and in terms of avoiding cluttering and occlusion, it is useful to place only one seed point per object. Commonly, this seed point is placed in the barycenter of the object and the motion direction is then represented by a flow glyph or a traceline.

### 5.2.4 Classifying Actions

Given an input video, the previous filters facilitate to extract appearing objects and to compute their motion direction. A combination of this extracted data leads to different visualizations shown in Section 5.4 and is investigated by the user study in Section 5.5. In addition to that, it can have a great impact on the perception of an observer to classify the objects with their accomplished actions and add this information to the visualization, as shown in Section 5.7. The problem of activity recognition and classification has received a large amount of attention from the computer vision community. Rao *et al.* [119], Gavrila and Davis [45], and Moeslund and Granum [104] review the previous work on activity recognition. Much of it involves tracking at the level of body parts, and hence is inapplicable for the smaller size human figures in lower quality surveillance videos that are used in this work. Other related approaches include Bobick and Davis [8], who derive the Temporal Template representation from background subtracted images. They present results on a variety of choreographed actions across different subjects and views, but require two stationary cameras with known angular interval, a stationary background, and a reasonably high-resolution video. Song *et al.* [140] demonstrate detection of walking and biking people using the spatial arrangement of moving point features. Freeman *et al.* [44] use image moments and orientation histograms of image gradients for interactive control in video games. Developing this theme, Zelnik-Manor and Irani [172] use marginal histograms of spatio-temporal gradients at several temporal scales to cluster and recognize video events. In later work, Shechtman and Irani [135] presented a motion correlation method which handles motion ambiguity due to aperture effects.

***Action Recognition*** Classifying actions can be accomplished by tracking and stabilizing each human figure present in each frame. This gives a figure-centric spatio-temporal volume for each person. Any residual motion within the spatio-temporal volume is due to the relative motions of different body parts: limbs, head, torso etc. This motion can be characterized by a descriptor based on computing the optical flow, projecting it onto a number of motion channels, and blurring. To recognize a similar motion afterwards, previously seen (and labeled) action fragments are stored in a database, and by computing a spatio-temporal cross correlation it is possible to find the one most similar to the motion descriptor of the query action fragment.

The implemented action classification technique builds on the work of Efros *et al.* [35] for analyzing the motion of a human figure, which has proven to be quite effective in discriminating between coarse-level actions, such as running or walking in different directions. In particular, it can make these discriminations from low-resolution video data, of the type which would be commonly found in a surveillance setting. This approach performs action recognition in a nearest neighbor framework. The distance measure used for comparing novel data with stored examples is based on blurred optical flow measurements. Details of the motion descriptor are given in the next section.

The filter was applied to the set of 26 video streams provided by the CAVIAR project [41], which captured different scenarios at the front of a shop entrance in a Lisbon shopping center. These video streams were designed to test computer vision algorithms for object, action and event recognition and classification. They are all accompanied by hand labeled ground truth information such as object bounding boxes and basic action classification.

In order to feed the visualization sub-system with more interesting information for perceptual and semantic reasoning, a slightly more detailed class of actions has been defined, than what is in CAVIARs ground truth annotation. In particular, motion types are associated with directions, and detected actions are provided with a plausibility measurements. Compared to action recognition, detection of human figures is relatively straight-forward in these scenes. Hence, the bounding box information in the ground truth annotation was only used for detecting the objects in motion.

***Motion Descriptor*** The motion analyzing algorithm starts by computing a figure-centric spatio-temporal volume for each person. Such a representation can be obtained by tracking the human figure and then constructing a window in each frame centered at the figure. Any of a number of trackers is appropriate. The main requirement is that the tracking has to be consistent — a person in a particular body configuration should always map to approximately the same stabilized

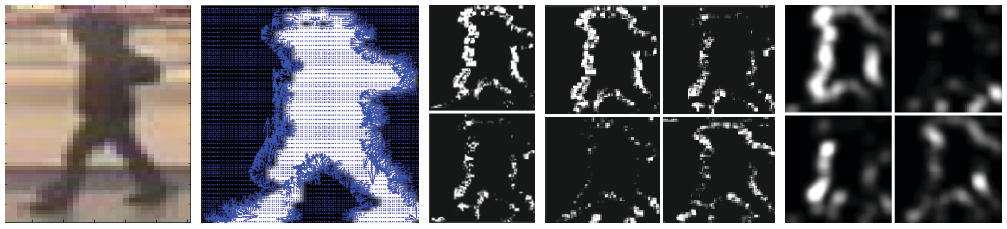


Figure 5.4: Constructing the motion descriptor: (a) Original image; (b) Optical flow; (c) Separating the $x$ and $y$ components of optical flow vectors for $F_x$ and $F_y$; (d) Half-wave rectification of each component to produce four separate channels $F_x^+$, $F_x^-$, $F_y^+$ and $F_y^-$; (e) Final *blurry motion channels* $Fb_x^+$, $Fb_x^-$, $Fb_y^+$ and $Fb_y^-$. (Images courtesy of Botchen *et al.* [9], ©2008 IEEE).

image, but the method used is robust to small jittering.

Once the motion sequences are stabilized it becomes possible to directly compare them in order to find correspondences. Finding similarity between different motions requires both spatial and temporal information. This leads to the notion of the *spatio-temporal motion descriptor*, an aggregate set of features sampled in space and time that describe the motion over a local time period. Computing such motion descriptors centered at each frame will enable the algorithm to compare frames from different sequences based on local motion characteristics.

Given a stabilized figure-centric sequence, the optical flow is computed at each frame using the Lucas-Kanade [93] algorithm (see Figure 5.4 (a,b)). The optical flow vector field $\mathbf{F}$ is first split into two scalar fields corresponding to the horizontal and vertical components of the flow, $F_x$ and $F_y$, each of which is then half-wave rectified into four non-negative channels $F_x^+$, $F_x^-$, $F_y^+$, $F_y^-$, so that $F_x = F_x^+ - F_x^-$ and $F_y = F_y^+ - F_y^-$ (see Figure 5.4 (c,d)). These are each blurred with a Gaussian and normalized to obtain the final four channels, $\hat{Fb}_x^+$, $\hat{Fb}_x^-$, $\hat{Fb}_y^+$, $\hat{Fb}_y^-$, of the motion descriptor for each frame (see Figure 5.4 (e)). Alternative implementations of the basic idea could use more than 4 motion channels — the key aspect is that each channel be sparse and non-negative.

***Action Plausibility Measure*** The spatio-temporal motion descriptors are compared using a version of normalized correlation. If the four motion channels for frame $i$ of sequence $A$ are $a_1^i$,$a_2^i$,$a_3^i$, and $a_4^i$, and similarly for frame $j$ of sequence $B$ then the similarity between motion descriptors centered at frames $i$ and $j$ writes

$$S(i,j) = \sum_{t \in T} \sum_{c=1}^{4} \sum_{x,y \in I} a_c^{i+t}(x,y) b_c^{j+t}(x,y) \quad , \tag{5.1}$$

where $T$ and $I$ are the temporal and spatial extents of the motion descriptor respectively. To compare two sequences $A$ and $B$, the similarity computation will need to be done for every frame of $A$ and $B$ so Equation (5.1) can be optimized in the following way. First, a frame-to-frame similarity matrix of the blurry motion channels (the inner sums of the equation) is computed between each frame of $A$ and $B$. Let us define matrix $A_1$ as the concatenation of $a_1$'s for each frame stringed as column vectors, and similarly for the other 3 channels. Then the frame-to-frame similarity matrix $S_{ff} = A_1^T B_1 + A_2^T B_2 + A_3^T B_3 + A_4^T B_4$. To obtain the final motion-to-motion similarity matrix $S$, the frame-to-frame similarities are added up over a temporal window $T$ by convolution with a $\|T\| \times \|T\|$ identity matrix, thus $S = S_{ff} \star I_T$, where $\star$ is the cross correlation.

Given a novel sequence to be classified and a database of labeled example actions, a motion similarity matrix is constructed first as outlined above. For each frame of the novel sequence, the maximum score in the corresponding row of this matrix will indicate the best match to the motion descriptor centered at this frame. Now, classifying this frame using a $k$-nearest-neighbor classifier is simple: find the $k$ best matches from labeled data and take the majority label.

### 5.2.5 Object Relations

The process described in 5.2.4 also results in an object related action list, where for each object $x$, there is a vector of measurements characterizing the actions of $x$ at a discrete temporal point $t \in \mathbb{N}$, $\mathbf{X}(t) = [x_1(t), a_2(t), \ldots, a_n(t)]^\top$. One can recognize that $\mathbf{X}(t)$ is a discrete multivariate time series and each $x_i(t)$ is its elementary time series. The measurement of $x_i(t)$ can be *nominal* (e.g., names for categorizing actions), *ordinal* (e.g., the importance rank order of $x$), an *interval* (e.g., motion speed of $x$), or a *ratio* (e.g., plausibility measurement). Some of these measurements are grouped together to form composite measurements, such as coordinates, motion directions, bounding box, etc.

One must note that there is no general assumption that elementary time series are independent of each other, their correlation dimensionality is known, or they possess special statistical properties, such as periodicity and persistence. The establishment of such statistical properties from sample videos is beyond the scope of this work. Hence not all tools for time series analysis are readily applicable [80]. However, several principle methods in time series analysis can be adopted in this particular application. They include *filtering*, *moving average*, *cross-correlation*, and *power of time series*, which are to be detailed below.

One of the goals is the feasibility of visualizing complex events, such as the possibly related actions in a scene, by drawing the viewers' attention to the possibility of such relations, rather than informing the viewer of an explicit conclusion which can be very unreliable in general.

As the particular interest lies in real-time processing of video streams, this requires the visualization to be updated dynamically, with only the access to video data in a relatively short time span. Without losing generality, here it is possible to consider only relations between two time series $\mathbf{X}(t)$ and $\mathbf{Y}(t)$ representing the actions of objects $x$ and $y$ respectively.

***Object Relation Filtering*** *Time-invariant relation filters* generate a new time series, $\mathbf{r}_{x,y}(t)$, or $\mathbf{r}(t)$ in short. Each of its elementary time series, $r_i(t)$ is a function of one or more elementary time series of $\mathbf{X}(t)$ and $\mathbf{Y}(t)$, and measures the probability if actions of $x$ and $y$ may be related in a specific aspect. The time-invariance implies that if there is a filter $F$ such that $F\big(a(t)\big) = b(t)$, we also have $F\big(a(t+h)\big) = b(t+h)$ for any $h \in \mathbb{N}$. Nevertheless, there is no restriction as to the linearity and the size of the time window of each filter. Let $P_x(t)$ be a composite time series representing the centroid of an object $x$, $V_x(t)$ be its motion direction, $B_x(t)$ be its bounding box. Below is a set of example filters:

*Closeness* $r_C(t)$. Let $d_{max} > 0$ be a constant, and $D$ be the Euclidean distance function between two points. We have

$$r_C(t) = \begin{cases} 0, & \text{if } D\big(P_x(t), P_y(t)\big) \geq d_{max} \\ 1 - \frac{D\left(P_x(t), P_y(t)\right)}{d_{max}}, & \text{otherwise} \end{cases} \quad . \tag{5.2}$$

*Moving in similar directions* $r_D(t)$. Let $\theta$ be the angle between vectors $V_x(t)$ and $V_y(t)$ which can be obtained trivially. We have

$$r_D(t) = \max\big(0, \cos(\theta)\big) \quad . \tag{5.3}$$

*Moving with similar speeds* $r_S(t)$. Let $\| \ \|$ denote the magnitude of a vector, and $s_{max} > 0$ be a constant. We have

$$r_S(t) = \begin{cases} 0, & \text{if } \mid \| V_x(t)\| - \|V_y(t)\| \mid \geq s_{max} \\ 1 - \mid \frac{\| V_x(t)\| - \|V_y(t)\|}{s_{max}} \mid, & \text{otherwise} \end{cases} \quad . \tag{5.4}$$

*Overlapping of bounding boxes* $r_A(t)$. Let $A$ be an area function, and $\cup$ and $\cap$ denote the spatial union and intersection of two bounding boxes. We have

$$r_A(t) = \frac{A\big(B_x(t) \cap B_y(t)\big)}{A\big(B_x(t) \cup B_y(t)\big)} \quad . \tag{5.5}$$

*Moving towards each other* $r_T(t)$. Let $\theta_{x \to y}$ be the angle between vector $V_x(t)$ and $P_y(t) - P_x(t)$, $\theta_{y \to x}$ be the angle between vector $V_y(t)$ and $P_x(t) - P_y(t)$, and

$\upsilon_{max} > 0$ be a constant. We have

$$r_T(t) = \begin{cases} 0, & \text{if } \theta_{x\to y} \leq 0 \vee \theta_{y\to x} \leq 0 \\ 1, & \text{if } \theta_{x\to y} > 0 \wedge \theta_{y\to x} > 0 \wedge \upsilon \geq \upsilon_{max} \\ \upsilon, & \text{otherwise} \quad , \end{cases} \tag{5.6}$$

with $\upsilon = \cos(\theta_{x\to y})\|V_x(t)\| + \cos(\theta_{y\to x})\|V_y(t)\|$. $\upsilon$ is the combined velocity of $V_x(t)$ and $V_y(t)$ modulated by $\cos(\theta_{x\to y})$ and $\cos(\theta_{y\to x})$ respectively.

***Moving average*** This is an efficient technique for computing dynamic properties of a time series. It can be applied to the elementary time series prior to, or after the filtering. In this work, the *exponential moving average* is employed, which minimizes the need for the system to memorize the records of the previous time span:

$$\begin{aligned} \bar{r}(t_0) &= r(t_0) \\ \bar{r}(t) &= \alpha r(t) + (1-\alpha)\bar{r}(t-1) \quad , \end{aligned} \tag{5.7}$$

where $0 \leq \alpha \leq 1$. In this work, the typically value for $\alpha$ is chosen to be $0.5$.

***Cross correlation*** Some useful indicators of a relation are in the form of *cross correlation*, which evaluating covariance between two random vectors. For example, we can use cross correlation to examine if the corresponding time series of two objects are following the same trend. The *Pearson product-moment correlation coefficient* applies for this requirement. For a time span [t-h, t], and two corresponding time series, $x(t)$ and $y(t)$ (which can be original or resulting from filtering), we have:

$$r_{Pearson}(t) = \frac{1}{h}\sum_{t-h}^{t} z_x(i)z_y(i) \quad .$$

where $z_x(i)$ and $z_y(i)$ are the standard scores of $x(i)$ and $y(i)$ in the time span $[t-h, t]$. One can observe easily that a larger time span will require relatively more computation resources.

***Power of time series*** The power of a time series $\mathbf{E}(x(t), t_1, t_2)$ over a time span $[t_1, t_2]$ indicates the energy of the "activity" during that period. Using the notion of average energy of a time series, gives

$$\mathbf{E}(x(t), t_1, t_2) = \frac{1}{h}\int_{t_1}^{t_2} x^2(t)dt \approx \frac{1}{h}\sum_{t_1}^{t_2} x^2(t) \quad .$$

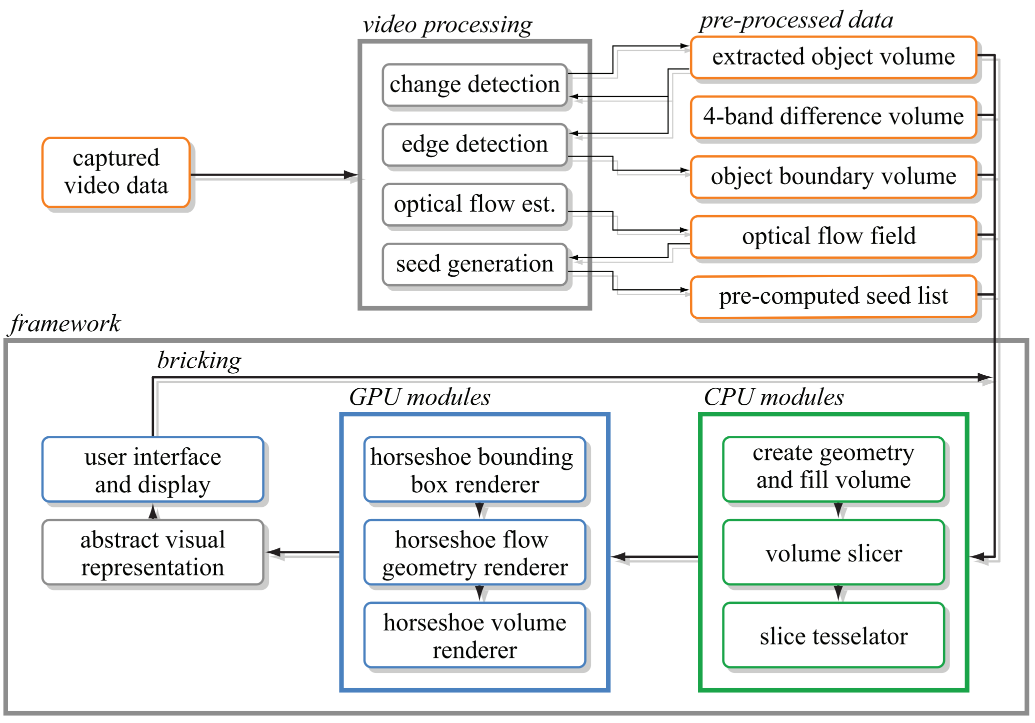


Figure 5.5: The system pipeline for multi-field video visualization.

The power of a multivariate time series, $\mathbf{r}(t)$, is the weighted average of the energy of its individual elementary time series that is defined as

$$\mathbf{E}(\mathbf{r}(t), t_1, t_2) = \frac{\sum_i \omega_i \mathbf{E}\big(r_i(t), t_1, t_2\big)}{\sum_i \omega_i} \quad .$$

## 5.3 The Multi-field Video Visualization Framework

The flow chart in Figure 5.5 shows the overall system architecture of the video visualization rendering (VVR) framework, which includes two major functional sub-systems, namely *video processing* and *video rendering*. The main development goals for this pipeline were: (i) to extract a variety of intermediate datasets that represent attribute fields of a video. Such datasets include extracted object volume, difference volume, boundary volume, and optical flow field; (ii) to synthesize different visual representations using volume and flow visualization techniques individually as well as in a combined manner; and (iii) to enable real-time visualization of deformed video volumes (i.e., the horseshoe view), and to facilitate interactive specification of viewing parameters and transfer functions.

The *video processing* sub-system focuses on the generation of appropriate attribute fields, including extracted object volume, 4-band difference volume, object boundary volume, optical flow field, and seed list. These attribute fields are generated with a collection of filters introduced in Section 5.2 and are used to highlight the different features in the following rendering stage.

The *video rendering* sub-system is the main focus of this section, with the adaption of volume bricking to handle large volume and flow datasets. One modification for video volume bricking is that partitioning of data happens only in the temporal dimension instead of the spatial partitioning commonly used in traditional volume rendering. As shown in Figure 5.5, the bricking process affects most modules in the rendering sub-system through a loop that triggers a dynamic update within each module. Because of the existence of this loop and the logical brick structure, this bricking mechanism supports scalable multi-field visualization, including video spans, glyph geometry for flow visualization, and dynamic streamlines.

The video volume rendering adapts the horseshoe layout for video rendering from [30], because the horseshoe geometry has a number of merits, including a cost-effective space utilization and a provision of four visible sides of a video volume. However, the horseshoe layout requires the rendering of a deformed video volume. A generic way to render deformed volumes is to use raycasting (e.g., ray reflectors [84]), or utilize texture slicing [120], to render deformed volumes in real time. In this approach, a slice-based volume rendering method is implemented and here, a backward mapping is employed to modify the texture coordinates that address the dataset.

For real-time rendering of large video volumes, GPU methods are employed to achieve high frame rates. The visualization framework is built upon an existing slice-based volume renderer [156]. An advantage of this framework is its separation of different visualization aspects into different software components. The framework is implemented in C++, using the Direct3D graphics API and HLSL as shader programming language.

In the following sub-sections the technical details of distorted video volume rendering and optical flow visualization are discussed. The starting point for visualization is volume rendering that shows a scalar field associated with the 3D space-time video volume. In combination with appropriate transfer functions, relevant information of the video volume can be emphasized and uninteresting regions can be made transparent. A challenge for video volume visualization is the interactive rendering of large datasets (see Sections 5.3.2), possibly using a distorted horseshoe geometry as detailed in the next section. The second part of the visualization system provides a representation of optical flow by glyphs or streamlines constructed by particle tracing (see Section 5.3.3).

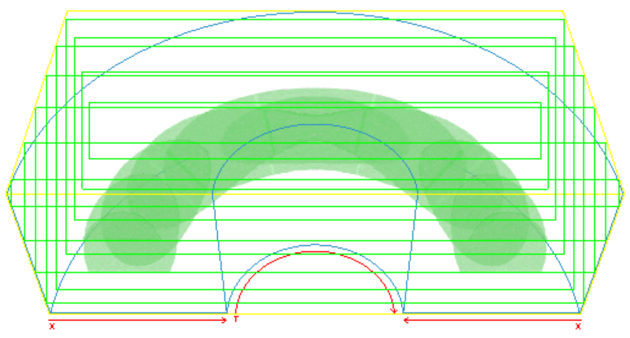


Figure 5.6: Bounding boxes of the P-space (blue) and C-space (yellow). The slice planes (green) are mapped to C-space in the fragment shader. (Image courtesy of Botchen *et al.* [10], ©The Eurographics Association 2006).

### 5.3.1 Distorted Video Volumes

The visible video volume might need to be distorted during rendering. The primary example is the bending into a horseshoe shape [30], as shown in Figure 5.6. A backward-mapping approach for rendering such deformed volumes can be applied: instead of deforming the geometry of the volume, it is easier to distort the associated texture coordinates to obtain the same result [120]. Therefore, planar and view-aligned slices are rendered with modified 3D texture coordinates.

The texture coordinates in computation space (*C*) are described by $(x_C, y_C, t_C)$ in the range $[0,1]^3$. Here, $x$ and $y$ denote the spatial dimensions of a video slice and $t$ denotes the temporal dimension. In contrast, the coordinates in the physical space (*P*) – the object space of the distorted volume – are given by $(x_P, y_P, z_P)$.

For the case of the horseshoe volume, a transformation according to cylindrical coordinates is assumed,

$$(x_P, y_P, z_P) = (-r\cos(\pi t_C), y_s\, y_C, r\sin(\pi t_C)) \quad , \tag{5.8}$$

$$\text{with} \qquad r = r_{\text{min}} + \Delta r\; x_C \quad \text{and} \quad \Delta r = r_{\text{max}} - r_{\text{min}} \quad .$$

Here, $r_{\text{max}}$ and $r_{\text{min}}$ describe the inner and outer radius of the horseshoe, respectively. The parameter $y_s$ provides a scaling factor for the $y$ dimension. Figure 5.7 illustrates the different coordinate systems.

The inverse mapping of Equation (5.8) is used to transform the physical coordinates of the slices to texture coordinates $(x_C, y_C, t_C)$ that address the video volume. The inverted mapping involves inverse trigonometric functions, which are available in GPU fragment programs. Therefore, the volume deformation can be implemented by computing texture coordinates in a fragment program during texture slicing. An example of such a fragment program is provided in Section 5.3.2. Since the video volume is not illuminated, it is possible to omit the transformation

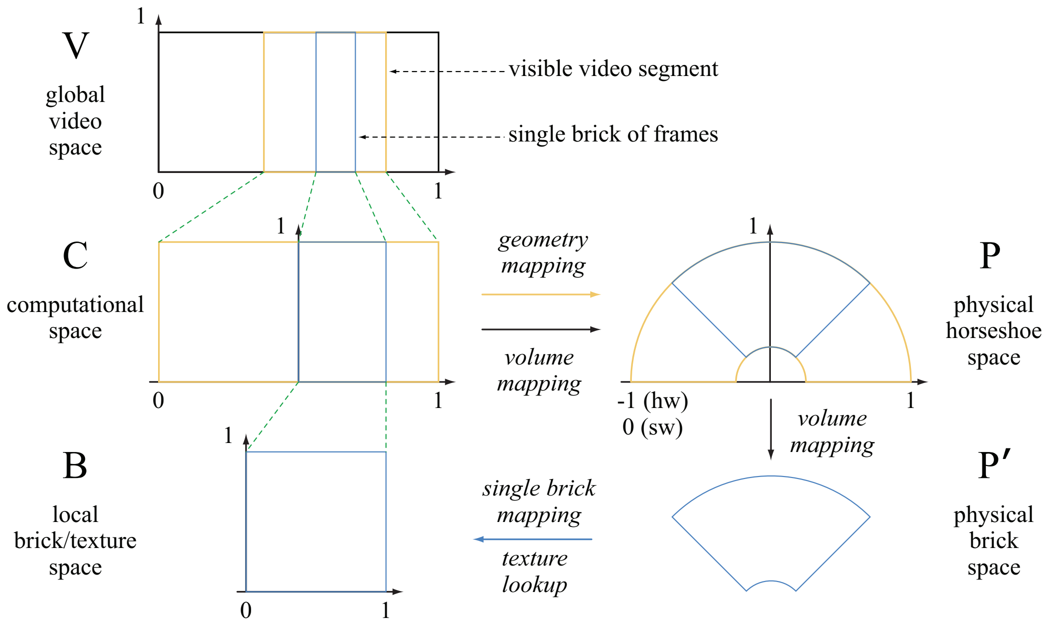


Figure 5.7: Mapping between coordinate systems. (Image courtesy of Botchen *et al.* [10], ©The Eurographics Association 2006).

of volume gradients for appropriate volume shading (see [120] for a description of this type of transformation).

### 5.3.2 Scalable Multi-field Bricking

To visualize a large video dataset that cannot be loaded to GPU memory *en bloc*, it is necessary to subdivide the whole domain into smaller sections that can be handled and processed by the GPU. Therefore, a generic implementation that combines volume visualization and the rendering of flow geometry in scalable user-defined bricks is introduced in this section.

Let the video $V$ be an ordered set of 2D image frames $I_i, i \in \{1, .., N\}$, where $N$ is the total number of frames. The volume is divided into $K \geq 1$ video bricks, where $1 \leq k \leq K$ bricks are rendered at a time. Each brick, $B_j \subseteq V$, contains $m$ image frames, with $B_j \cap B_l = \emptyset$, where $j, l \in \{1, .., k\} \wedge j \neq l$ and the condition $k \cdot m = n$, with $n \leq N$. When the GPU memory cannot handle the data size of $N$ frames, which means that the condition $n < N$ holds, dynamic bricking needs to be applied to process the data. Each logical brick is described by two integer values: the number of the starting frame and the number of frames in the brick. Based on the information given by the logical brick structure, the memory for $k$ texture objects is allocated and each single volume brick is filled with its corresponding video frames. Dynamic bricking is realized by reassigning

```
float  volData, tmpZ; float2 rp;
float3 txCrd = In.TextureCoord0;                      // cartesian coordinates
float3 lkup;                                          // horseshoe coordinates

txCrd.x = ((txCrd.x*2.0) - 1.0f) * (-1.0f);           // transform to P space
rp.x    = sqrt(pow(txCrd.x,2) + pow(txCrd.z, 2));     // map from P to C
rp.y    = atan2(txCrd.z,txCrd.x);
lkup.x  = (rp.x - g_fInRad)/(g_fOutRad - g_fInRad);   // compute radius
lkup.y  = txCrd.y; tmpZ = rp.y/g_PI;                  // compute angle

lkup.z  = (tmpZ - g_vSclCrd.x) * g_vSclCrd.y;         // map from C to B
volData = tex3D(VOLsmp, lkup);                        // perform 3D lookup

Output.RGBColor = tex1D(TFsmp, volData.x);            // apply color values
return Output;                                        // write to buffer
```

Listing 5.1: The complete code of an HLSL fragment program for the bricked, dynamic video spans. (Code courtesy of Botchen *et al.* [10], ©The Eurographics Association 2006).

the pointers in a cyclic way, forming a ring-buffer data structure. Thus, the last texture object contains information that can be overwritten and filled with the frames that newly enter the horseshoe.

Furthermore, a brick-based filter for seed generation, which is a modified version of the one described in Section 5.2.3, needs to be applied. It enables frames in different bricks to share a pre-processed seed point list. The input of this shared list is used by all bricks and a flag indicates for each point whether this seed point is used for constructing geometry for that brick or not. Starting from this logical entity, the whole dynamic bricking structure can be built that consists of $k$ 3D volume textures that are shifted through the horseshoe, and the flow geometry generated from the seed points and the optical flow vectors as detailed below.

The fragment program that renders a single volume brick is given in Listing 5.1. The first line of code scales the texture coordinates to a range of $[-1, 1]$, because this permits us to map the cylindrical horseshoe coordinates between $-\pi$ and $\pi$. This mapping leads to a half circle in the $xz$ plane, as required by the bent horseshoe (see Figure 5.7). The following four lines realize the inverse of the mapping in Equation (5.8), by first computing the radius and angle of the intermediate cylindrical coordinate system, and then mapping them to the coordinate system *C*, which represents the visible part of the video volume. The final mapping takes the coordinates into the local coordinate system of the brick *B*, which is a subset of the visible video volume *C*. With these brick-related coordinates, a 3D texture lookup is performed and a final RGB$\alpha$ value is assigned according to the transfer function.

### 5.3.3 Integrating Optical Flow in Volume Visualization

To combine an optical flow field with the distorted scalar field for the horseshoe video volume, the video volume rendering system allows opaque flow geometry to be added into the scene. The geometry, in the form of arrow glyphs or traced lines, is created on-the-fly by the module *FlowGeometryRenderer* and stored in the according geometry buffers before the actual rendering takes place.

Building the arrow geometry requires two pieces of information: a point $\mathbf{p}$ and a direction $\mathbf{v}$, which are given by the pre-computed seed points $S_i$ and the optical flow vectors $\mathbf{v}_i$, as described in Sections 5.2.2 and 5.2.3. In fact, the original optical flow field is extended from a 2D spatial vector field described by $(u, v)$ to a 3D space-time vector field with an additional component along the temporal dimension: $\mathbf{v} = (u, v, v_t)$. The temporal vector component $v_t$ describes the "velocity" along the time axis of the video volume. So far, only temporally equidistant sampling of the video volume is used. Therefore, $v_t$ is constant for the whole volume and represents the relative speed along the time axis. The user is allowed to define the relative speed $v_t$. All example images in the following sections use $v_t = 0$ in order to focus on the motion within individual frames. Based on this 3D optical flow, for each seed point a reference geometry for glyphs can be copied to the geometry buffer, and shifted and rotated into the proper position and orientation.

As an alternative, particle tracing is used to visualize the trajectory of particles along the flow and to provide information of longer moving structures inside a frame. These lines not only emphasize the distance of a movement, but also can indicate a change in direction. Particle tracing needs more processing steps and is implemented using Euler integration given in Equation (2.5). Trilinear interpolation is used as reconstruction filter for the vector field. The number of computed integration steps is chosen by the user, manipulating the length of the traced lines. The rendering of those lines with dynamic texture mapping, as shown in Figure 5.8, is detailed in Section 5.3.3.

One additional issue occurs when the video volume is distorted. In this case, the original vector field data, which is given in *C* space, needs to be transformed into the physical space *P* in order to obtain correct particle traces or glyph orientations. Similar to the coordinate transformation for the scalar field as discussed in Section 5.1, a transformation rule for vector fields is needed. In general, vectors can be defined as differentials according to

$$\mathbf{dy} = \sum_{i=0}^{2} \frac{\partial \mathbf{y}}{\partial x_i} \mathrm{d}x_i = \sum_{i=0}^{2} \mathbf{e}_i \mathrm{d}x_i \quad .$$

Here, the $\mathbf{e}_i$ serve as basis for the vectors in the space associated with $x_i$. In the

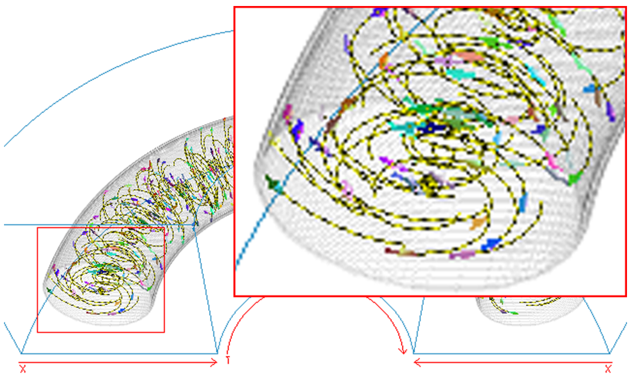


Figure 5.8: Directional textured tracelines in combination with arrow glyphs. The arrows are placed at the seed points in flow direction. (Image courtesy of Botchen *et al.* [10], ©The Eurographics Association 2006).

case of the horseshoe, we have

$$\begin{aligned}
\mathbf{e}_x &= \frac{\partial \mathbf{x}_P}{\partial x_C} = \Delta r(-\cos(\pi t_C), 0, \sin(\pi t_C)) \\
\mathbf{e}_y &= \frac{\partial \mathbf{x}_P}{\partial y_C} = (0, y_s, 0) \\
\mathbf{e}_t &= \frac{\partial \mathbf{x}_P}{\partial t_C} = \pi\ \Delta r(\sin(\pi t_C), 0, \cos(\pi t_C)) \quad ,
\end{aligned}$$

with $\mathbf{x}_P = (x_P, y_P, z_P)$. With these basis vectors, a vector field $\mathbf{v}_C = (v_x, v_y, v_t)$ given in the coordinate system *C* is transformed to the coordinate system *P* by

$$\mathbf{v}_P = \sum_{i=x,y,t} \mathbf{e}_i v_i \quad .$$

***Directional Textured Tracelines*** Lines are 1D primitives that convey information about the orientation and extent of a trace along the flow, but fail to indicate the flow direction. Therefore, animation is added to the tracelines in order to highlight the direction of flow. The idea is to attach an animated 1D texture that moves into the direction of the flow. The texture needs to have some kind of visual structure so that its motion can be perceived. All tracelines shown in combination with horseshoe rendering use a zebra-like texture, as illustrated in the zoomed box of Figure 5.8. This image illustrates a spinning sphere lying in the $xy$-plane and rotating around the $z$-axis. The arrow glyphs rendered at seed-point locations show the flow direction at these certain locations. In contrast, the traced lines provide flow information along a longer distance, covering more locations of the domain.

For texture mapping, each vertex of a line is assigned a texture coordinate, with a range between $[0, 1]$ from the first to the last vertex, respectively. By shifting the local texture coordinate of each vertex with a global parameter $\Delta t$, the texture moves along the line, in direction of the underlying flow field. The 1D texture does not need to be changed for the animation and can thus be computed on the CPU and downloaded to the GPU once.

### 5.3.4 Flow Geometry Bricks

The geometry bricks are similarly to the volume bricks held in a pointer structure that eases the swapping of the bricks for the dynamic rendering of large video data. Unlike the volume bricks, a geometry brick only consists of the logical structure that holds the range information of the currently visible region. The render geometry for arrows and streamlines is constructed for the whole visible horseshoe region (Section 5.3.3) only when needed and directly mapped from *C* to *P* (Figure 5.7). All points that result from particle tracing are stored in a single vertex buffer and rendered as line strip. The arrow geometry is stored in an indexed vertex buffer to avoid redundant vertices. All buffers are rendered as opaque geometry before the semi-transparent volume is displayed with back to front blending. This rendering order allows to accurately mix geometry and volume information by means of the depth test.

## 5.4 Types of Visual Signatures

As described in Section 5.1, a visual signature of an object is given by the spatiotemporal entity $m$. With the various information extracted in the video processing stage, it is feasible to construct different visual signatures to highlight different attributes of $m$. Examples for the construction of different signatures are the following time-varying attributes: (i) the shape of the object; (ii) the position of the object; (iii) the object appearance (e.g., intensity and texture); (iv) the velocity of the motion. Consider an animation video of a simple object in a relatively simple motion. As shown in Figure 5.9, the main spatiotemporal entity contained in the video is a textured sphere moving upwards and downwards in a periodic manner.

To obtain the time-varying attributes about the shape and position of the object concerned, an image processing algorithm extracts the object silhouette in each frame from the background scene. It is also reasonable to identify the boundary of the silhouette, which to a certain extent conveys the relationship between the object and its surroundings (in this simple case, only the background).

To characterize the changes of the object appearance, the difference between two consecutive frames is computed. Figure 5.9 (b) gives an example difference image. A 2D motion field is established to describe the movement of the object between each pair of consecutive frames, as shown in Figure 5.9 (c). Compiling

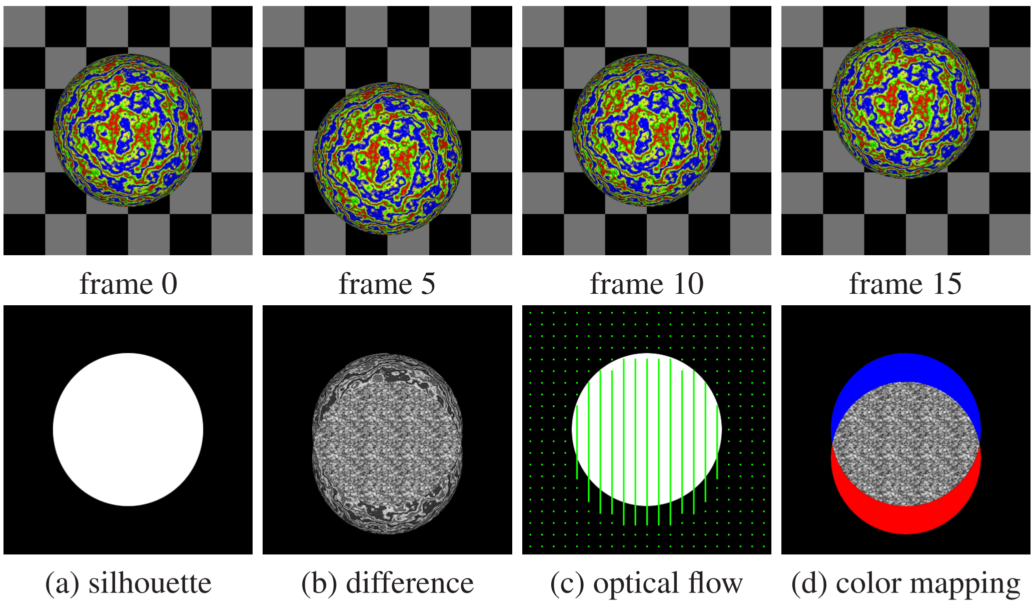


Figure 5.9: The top row shows four selected frames of a simple up-and-down motion, depicting the first of the five cycles of the motion, together with examples of its attributes associated with frame 0 illustrated in the bottom row. (Images courtesy of Chen *et al.* [24], ©2006 IEEE).

all silhouette images as seen in Figure 5.9 (a), into a single volume results in a 3D scalar field that is called an extracted object volume. Similarly, a difference volume can be obtained, which is also in the form of 3D scalar fields. The compilation of all 2D motion fields in a single volumetric structure gives a motion flow in the form of a 3D vector field. Given these attribute fields of the spatiotemporal entity $m$, it is now possible to consider the creation of different visual signatures for $m$.

One can find numerous ways to visualize such scalar and vector fields individually or in a combinational manner. Without complicating the user study to be discussed in Section 5.5, four types of visualization for representing visual signatures are selected. Each type of visual signature highlights certain attributes of the object in motion, and reflects the strength of a particular volume or flow visualization technique. All four types of visualization can be synthesized in real time. For the following discussions, the horseshoe view [30] was chosen as the primary view representation. In comparison with conventional viewing angles, it places four faces of a volume, including the starting and finishing frames, in a front view. It also facilitates relatively more cost-effective use of a rectangular display area, and conveys the temporal dimension differently from the two spatial dimensions.

### Type A: Temporal Silhouette Extrusion

This type of visual signature displays a projective view of the temporal silhouette hull of the object in motion. Steady features, such as background, are filtered away. Figure 5.10 (a) shows a horseshoe view of the extracted object volume for the video mentioned in Figure 5.9. The temporal silhouette hull, which is displayed as an opaque object, can be seen wiggling up and down in a periodic manner.

### Type B: 4-Band Difference Volume

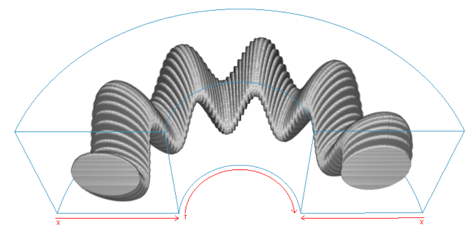
Difference volumes played an important role in [30], where amorphous visual features rendered using volume raycasting successfully depicted some motion events in their application examples. However, their use of transfer functions encoded very limited semantic meaning. For this work, a special transfer function is designed that highlights the motion and the temporal change of a silhouette, while using a relatively smaller amount of bandwidth to convey the change of object appearance (i.e., intensity and texture).

Consider two example frames and their corresponding silhouettes in Figure 5.9. Pixels in the difference volume are classified into four groups as shown in Figure 5.9 (d), namely (i) background ($\notin O_a \wedge \notin O_b$); (ii) new pixels ($\notin O_a \wedge \in O_b$); (iii) disappearing pixels ($\in O_a \wedge \notin O_b$); and (iv) overlapping pixels ($\in O_a \wedge \in O_b$). The actual difference value of each pixel, which typically results from a change detection filter, is mapped to one of the four bands according to the group that the pixel belongs to. This enables the design of a transfer function that encodes some semantics in relation to the motion and geometric change.

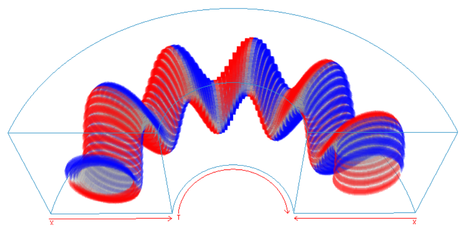
For example, Figure 5.10 (b) was rendered using the transfer function illustrated in Figure 5.9 (d), which highlights new pixels in nearly-opaque red and disappearing pixels in nearly-opaque blue, while displaying overlapping pixels in translucent gray and leaving background pixels totally transparent. Such a visual signature gives a clear impression that the object is in motion, and to a certain degree, provides some visual cues to velocity.

### Type C: Motion Flow with Glyphs

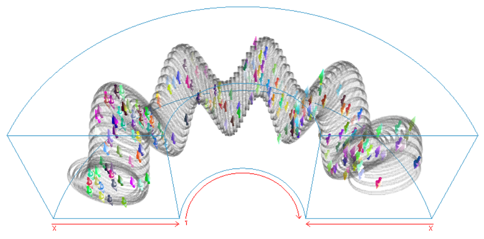
In many video-related applications, the recognition of motion is more important than that of an object. Hence it is beneficial to enhance the perception of motion by visualizing the motion flow field associated with a video. This type of visual signature combines the boundary representation of a temporal silhouette hull with arrow glyphs showing the direction of motion at individual volumetric positions. It is necessary to determine an appropriate density of arrows, as too many would clutter a visual signature, or too few would lead to substantial information loss. Thereby a combination of parameters is used to control the density of arrows,



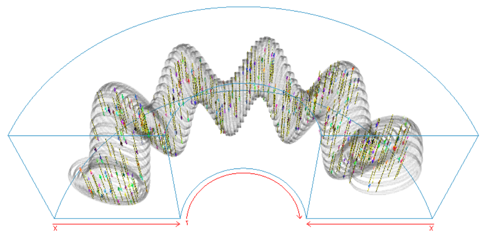
(a) type A: silhouette hull



(b) type B: 4-band difference volume



(c) type C: motion flow with glyphs



(d) type D: motion flow with streamlines

Figure 5.10: Four types of visual signatures of an up-and-down periodic motion. (Images courtesy of Chen *et al.* [24], ©2006 IEEE).

which will be discussed in Section 6. Figure 5.10 (c) shows a type C visual signature of a sphere in an up-and-down motion. In this particular visualization, colors of arrows are chosen randomly to enhance the depth cue of partially occluded arrows by improving their visual continuity.

Note that there is a major difference between the motion flow field of a video and typical 3D vector fields considered in flow visualization. In a motion flow field, each vector has two spatial components and one temporal component. The temporal component is normally set to a constant for all vectors. For the temporal component, experiments with a range of different constants have been conducted and it seemed that in most cases, a nonzero constant would confuse the visual perception of the two spatial components of the vector. Thereby, the temporal components of all vectors were chosen to be set to zero.

### Type D: Motion Flow with Streamlines

The visibility of arrow glyphs requires them to be displayed in a certain minimum size, which often leads to the problem of occlusion. One alternative approach is to use streamlines to depict direction of motion flow. However, because all temporal components in the motion flow field are equal to zero, each streamline can only flow within the $xy$-plane where the corresponding seed resides, and it seldom flows far. Hence there is often a dense cluster of short streamlines, making it difficult to use color for direction indication.

To improve the sense of motion and the perception of direction, a zebra-like dichromatic texture is mapped to the line geometry, which moves along the line in the flow direction. Although this can no longer be considered strictly as a static visualization, it is not in any way trying to recreate an animation of the original video. The dynamics introduced is of a fixed number of steps which are independent from the length of a video. The time requirement for viewing such a visualization remains to be $O(1)$. Figure 5.10 (d) shows a static view of such a visual signature. The perception of this type of visual signatures normally improves when the size and resolution of the visualization increases.

## 5.5 A User Study on Visual Signatures

The discussions in the previous sections naturally lead to many scientific questions concerning visual signatures. The followings are just a few examples:

- Can users distinguish different types of spatiotemporal entities (i.e., types of objects and types of motion individually and in combination) from their visual signatures?
- If the answer to the above is yes, how easy is it for an ordinary user to acquire such an ability?
- What kind of attributes are suitable to be featured or highlighted in visual signatures?
- What is the most effective design of a visual signature, and in what circumstances?
- What kind of visualization techniques can be used for synthesizing effective visual signatures?
- How would the variations of camera attributes, such as position and field of view, affect visual signatures?
- How would the recognition of visual signatures scale in proportion to the number of spatiotemporal entities present?

Almost all of these questions are related to the human factors in visualization and motion perception. There is no doubt that user studies must play a part in the search for answers to these questions. Therefore, an integral part of this work is a user study on visual signatures. Because this is the first user study on visual signatures of objects in motion, focus of this study lies on the recognition of types of motion, with the main objectives:

1. To evaluate the hypothesis that users can learn to recognize motions from their visual signatures.
2. To obtain a set of data that measures the difficulties and time requirements of a learning process.

3. To evaluate the effectiveness of the above-mentioned four types of visual signatures.

**Types of Motion**

As mentioned before, an abstract visual representation of a video is essentially a 2D projective view of our 4D spatiotemporal world. Visual signatures of spatiotemporal entities in real life videos can be influenced by numerous factors and appear in various forms. In order to meet the key objectives of the user study, it was necessary to reduce the number of parameters to be examined in this scientific process. In the same sense, simulated motions have been used with the following constraints:

- All videos feature only one spherical object in motion. The use of a sphere minimizes the variations of visual signatures due to camera positions and perspective projection.
- In each motion, the center of the sphere remains in the same $xy$-plane, which minimizes the ambiguity caused by the change of object size due to perspective projection.
- Since the motion function is known, most attribute fields are computed analytically. This is similar to an assumption that the sphere is perfectly textured and lit, and without shadows, which minimizes the errors in extracting attribute fields using change detection and motion estimation algorithms.

This lead to the following seven types of motion:

1. Motion Case $1$: No motion—in which the sphere remains in the center of the image frame throughout the video.
2. Motion Cases $2-9$: Scaling—in which the radius of the sphere increases by $100\%$, $75\%$, $50\%$ and $25\%$, and decreases by $25\%$, $50\%$, $75\%$ and $100\%$ respectively.
3. Motion Cases $10-25$: Translation—in which the sphere moves in a straight line in eight different directions (i.e., $0°$; $45°$; $90°$; ...; $315°$) and two different speeds.
4. Motion Cases $26-34$: Spinning—in which the sphere rotates about the $x$-axis, $y$-axis and $z$-axis, without moving its center, with $1$, $5$ and $9$ revolutions respectively.
5. Motion Cases $35$, $38$, $41$: Periodic up-and-down translation—in which the sphere moves upwards and downwards periodically in three different frequencies, namely $1$, $5$ and $9$ cycles.
6. Motion Cases $36$, $39$, $42$: Periodic left-and-right translation—in which the sphere moves towards left and right periodically in three different frequencies, namely $1$, $5$ and $9$ cycles.

7. Motion Cases $37$, $40$, $43$: Periodic rotation—in which the sphere rotates about the center of the image frame periodically in three different frequencies, namely $1$, $5$ and $9$ cycles.

The first four types are considered to be elementary motions. The last three are composite motions which can be decomposed into a series of simple translation motions in smaller time windows. Other composite motions were considered for inclusion first – such as the periodic scaling, and combined scaling, translation and spinning – but it was decided to limit the total number of cases in order to obtain an adequate number of samples for each case while controlling the time spent by each observer in the study. For the same reason of reducing parameters to keep the study simple, complex motions such as deformation, shearing and fold-over were not included as well.

### The Main User Study

Both the main and the supplemental user study were designed and conducted with the aim to examine whether users are able to learn to recognize visual signatures of motions, and to assist in the evaluation and the comparison of the effectiveness of different abstract visual representations of videos. The setup included the following:

***Participants*** 69 observers (23 female, 46 male) from the student community of Swansea University took part in this study. All observers had normal, or corrected to normal, vision and were given a 2 book voucher each as a small thank-you gesture for their participation. Data from two participants were excluded from analysis as their response times were more than 3 standard deviations outside of the mean. Thus, data from 67 (22 female, 45 male) observers were analyzed.

***Tasks*** The user study was conducted in 14 sessions over a three week period. Each session, which involved four or five observers, started with a 25 minutes oral presentation, given by Rudy R. Hashim (one of the collaborators in [24]), with the aid of a set of pre-written slides. The presentation outlined the steps of the test, and highlighted some potential difficulties and misunderstandings. As part of a learning process, a total of 10 motions and 11 visual signatures were shown as examples in the slides. The presentation was followed by a test, typically taking about 20 minutes to complete. A piece of interactive software was specially written for structuring the test as well as collecting the results.

The test was composed of 24 trials. On each trial, the observer was presented with between one and four visual signatures of a motion. The task was to identify the underlying motion pattern by selecting one of the four alternatives listed on the screen. Both the speed and the accuracy of this response were measured. As

| | static | scaling | translation | spinning | periodic |
|---|---|---|---|---|---|
| accuracy (%) | 81.2 (4) | 90.3 (2) | 66.7 (3) | 49.4 (3) | 62.2 (3) |
| response time (sec) | 19.8 (2) | 13.6 (1) | 23.8 (1) | 24.8 (1) | 24.4 (2) |

Table 5.1: Mean accuracy and response time related to motion types. Numbers in parentheses are standard errors (*se*) of the means. (Courtesy of Chen *et al.* [24], ©2006 IEEE).

observers were allowed to correct initial responses, the final reaction time was taken from the point when they proceeded to the next part of the trial.

The second part of the trial was designed to provide feedback and training for the observers to increase the likelihood of learning. It also provided a measure of subjective utility that is, how useful observers found each type of visual signature. In this part, the underlying motion clip was shown in full together with all four types of visual signatures. The task was to indicate which of the four visual signatures appeared to provide the most relevant information. No response time was measured in this part.

At the end of the experiment, observers were also asked to provide an overall usefulness rating for each type of visual signature. A rating scale ranging from 1 (least) to 5 (most) effective was used.

***Design*** The 24 trials in each test were blocked into four equal learning phases (six trials per phase) in which the amount of available information was varied. In the initial phase all four visual signatures were presented, providing the full range of information. In each successive phase, the number of available representations was reduced by one, so that in the final phase only one visual signature was provided. This fixed order was imposed so that observers would receive sufficient training before being presented with minimum information. For each observer a random sub-set of the 43 motion cases was selected and randomly assigned to the 24 experimental trials. For each case, the four possible options were fixed. The position of options was however randomized on the screen on an observer by observer basis to minimize simple response strategies.

### The Supplementary User Study

Since the number of visual signatures available in the main user study decreased from one phase to another, it may be difficult to know whether changes in the overall accuracy and response times directly reflect learning. To address this issue, a supplementary user study was conducted, where two visual signatures, types B and C, were made available throughout the 24 trials. It was organized in the same manner as the main study, and involved 40 observers (14 female, 26 male). Among them, 17 also took part in the main user study, hence had some experience

| | accuracy (%) | | | response time (sec) | | |
|---|---|---|---|---|---|---|
| | main | sup-1 | sup-2 | main | sup-1 | sup-2 |
| phase 1 | 66.7 | 68.1 | 75.5 | 30.8 | 24.7 | 26.7 |
| phase 2 | 70.0 | 74.6 | 76.5 | 22.2 | 18.9 | 19.4 |
| phase 3 | 72.0 | 74.3 | 82.4 | 17.5 | 12.0 | 16.9 |
| phase 4 | 63.0 | 71.7 | 78.4 | 13.4 | 11.2 | 10.8 |

Table 5.2: Mean accuracy and response time in each phase. The mean values are listed separately for the main user study, the first- and second-time groups in the supplementary user study. The standard errors (se) of the means listed are all between 1 and 2. (Courtesy of Chen *et al.* [24], ©2006 IEEE).

of video visualization, with a time lapse of 4-5 months. The other 23 were first-time observers, with no previous experience in video visualization.

### 5.5.1 Evaluation of Visual Signatures

Conducting both user studies including over 100 participants produced enough data for a solid and meaningful evaluation. In the data evaluation process, analysis of Variance (ANOVA) was used to explore differences between three or more means, and $t$-tests were used to directly compare two means. By convention, $F$ and $t$ values indicate the ratio between effects of interest and random noise using specific probability distributions. The probability $p$ of obtaining $F$ or $t$ values, given the statistical degrees of freedom indicated in parentheses, is also provided, with values less than 0.05 considered unlikely to occur by chance alone.

***Motion Types*** Table 5.1 gives the mean accuracy (in percentage) and response time (in seconds) in relation to motion types. There were clear differences between the types of motion, both in terms of accuracy ($F(4, 264) = 34.5, MSE = 5, p < 0.001$), and speed ($F(4, 264) = 12.6, MSE = 118, p < 0.001$). The scaling condition gave rise to the highest accuracy, clearly showing that positive identification of motion is possible from visual signatures. Post-hoc analysis showed that this condition did not lead to better performance than the trivial static case, but performance was reliably higher than the other three motion types (all have $t > 6.0, p < 0.001$).

Accuracy levels for the translational motions, including the elementary motion in one direction, and combinational motion with periodical change of directions did not differ from each other, but were both significantly above those for spinning motion ($t > 2.8, p < 0.01$). The difficulty in recognizing spinning motion appears to arise because the projection of the sphere in motion maintains the same outline and position throughout the motion. For example, the temporal silhouette hull of

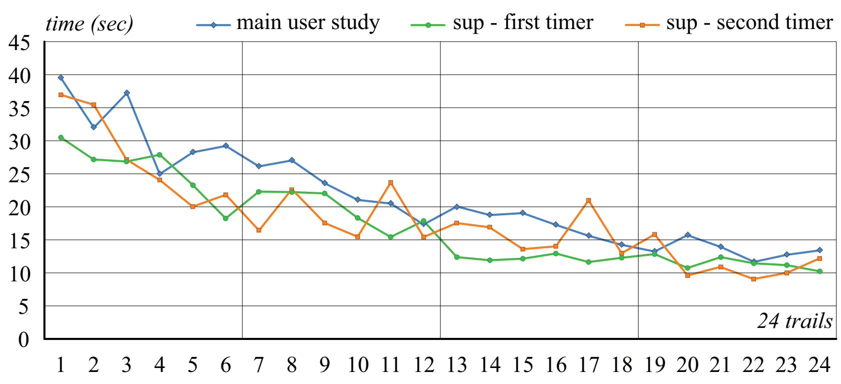


Figure 5.11: The decreasing trend of the mean response time of each trial in both user studies. (Image courtesy of Chen *et al.* [24], ©2006 IEEE).

Motion Case 31, which is a spinning motion, is identical to that of Motion Case 1, which is motionless. This renders type A visual signature totally ineffective in differentiating any spinning motion from the motionless state.

Response times, computed only for correct trials, followed a similar pattern. Here, however, scaling motion did give rise to significantly better performance than the static case ($t(114) = 3.1, p < 0.001$), in addition to the other three moving cases. No other comparisons were significant.

***Phases*** Table 5.2 gives the mean accuracy (in percentage) and response time (in second) in each of the four phases. Although the supplementary study was not divided into specific phases, the data was grouped into 4 and 6 trials for comparison purposes. In the main user study, accuracy levels changed significantly across the four phases ($F(3, 198) = 2.9, MSE = 3.7, p < 0.05$). While there is a clearly increasing trend across the first 3 phases, this main effect appears to be due more to the final drop between phases 3 and 4, the only pair of means to differ significantly ($t(132) = 2.23, p < 0.05$). This drop may be due to the reduction of the number of visual signatures to only one in Phase 4. A single visual signature is often ambiguous, for example, spinning and static cases share the same Type A visual signature in the user studies, so this could have inflated error rates. Another possibility is the lack of a confirmation process based on a second visual signature.

It has to be noted that a similar trend can also be observed in the supplementary study, where types B and C visual signatures were available throughout the session. Here, though, there was no main effect of phase. It seems possible that the generally high level of performance in both of the user studies may well be masking more subtle learning effects in terms of accuracy. Second time

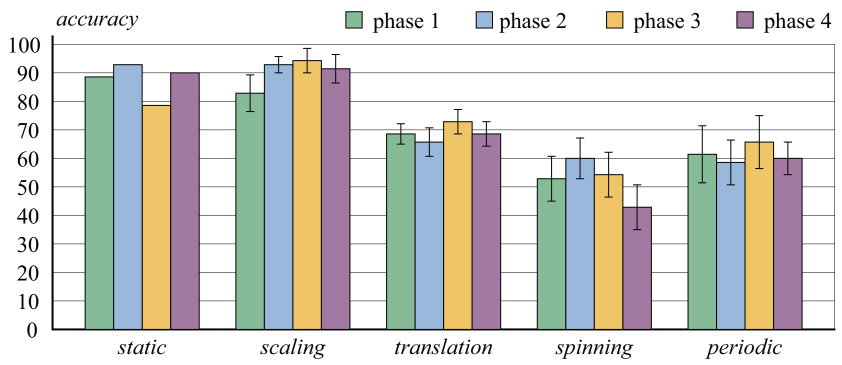


Figure 5.12: The *mean accuracy* (with standard errors), measured in each of the four phases, categorized by the types of motion. (Image courtesy of Chen *et al.* [24], ©2006 IEEE).

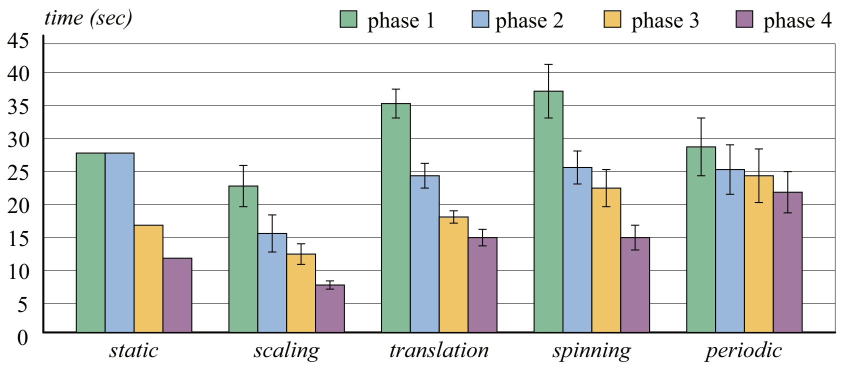


Figure 5.13: The *mean response time* (with standard errors), measured in each of the four phases, categorized by the types of motion. (Image courtesy of Chen *et al.* [24], ©2006 IEEE).

observers ($mean = 78\%, se = 2.6$) performed slightly better than first time observers ($mean = 72\%, se = 2.8$). Although this difference did not reach statistical significance, the trend towards higher performance is still encouraging. Any improvement, after a single prior exposure dating back several months, can provide some motivation to further explore long-term learning effects in this context.

In terms of response time, the story is much cleaner. In the main user study there was a clear effect of phase ($F(3, 198) = 43.5, MSE = 97.8, p < 0.001$), which takes the form of a consistent linear decrease ($F(1, 198) = 121.6, MSE = 97.8, p < 0.001$). Importantly, exactly the same pattern is present in the supplementary study, with a main effect of phase ($F(3, 114) = 35.2, MSE = 45.1, p < 0.01$), driven by a linear decrease in response time ($F(1, 114) = 103, MSE =$

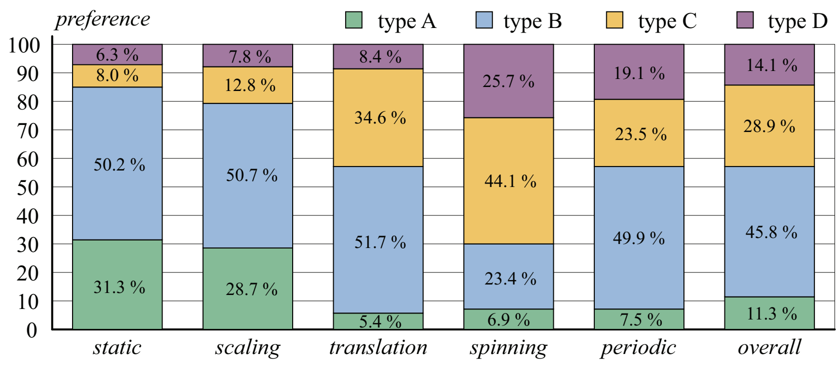


Figure 5.14: The relative preference of each type of visual signature, presented in the percentage term, and categorized by the types of motion. The overall preference is also given. (Image courtesy of Chen *et al.* [24], ©2006 IEEE).

$45.1, p < 0.001$). Thus, within the space of a single experiment, observers improve their performance even when the number of response options remains constant. There were no other significant response time effects in the supplementary study. Figure 5.11 shows this decreasing trend over all trials for both user studies.

For the main study, Figure 5.12 shows the accuracy in relation to each type of motion in each phase. We can observe that the spinning motion seems to benefit more from having multiple visual signatures available at the same time. The noticeable decrease of the number of positive identification of the motionless event in Phase 3 may also be caused by the difficulties in differentiating it from spinning. Figure 5.13 shows a consistent reduction of response time for all types of motion.

***Preference*** Figure 5.14 summarizes the preference of observers in terms of types of visual signatures, which largely reflects the effectiveness of each type of visual signature. Note that the type C visual signature was considered to be the most effective in relation to the spinning motion, while type B was generally preferred for other types of motion. The overall preference (shown on the right of Figure 5.14) was calculated by putting all "votes" together regardless the type of motion involved. This corresponds reasonably well with the final scores, ranging between 1 (least) to 5 (most) effective, given by the observers at the end. The mean scores for the four types of visual signatures are A:2.6, B:4.0, C:3.6, and D:3.1 ($0.14 \leq se \leq 0.16$ ) respectively.

### 5.5.2 Application Cases

Apart from the synthetic datasets that were used to produce the visualizations for the user study, the video visualization system was applied to a set of video clips

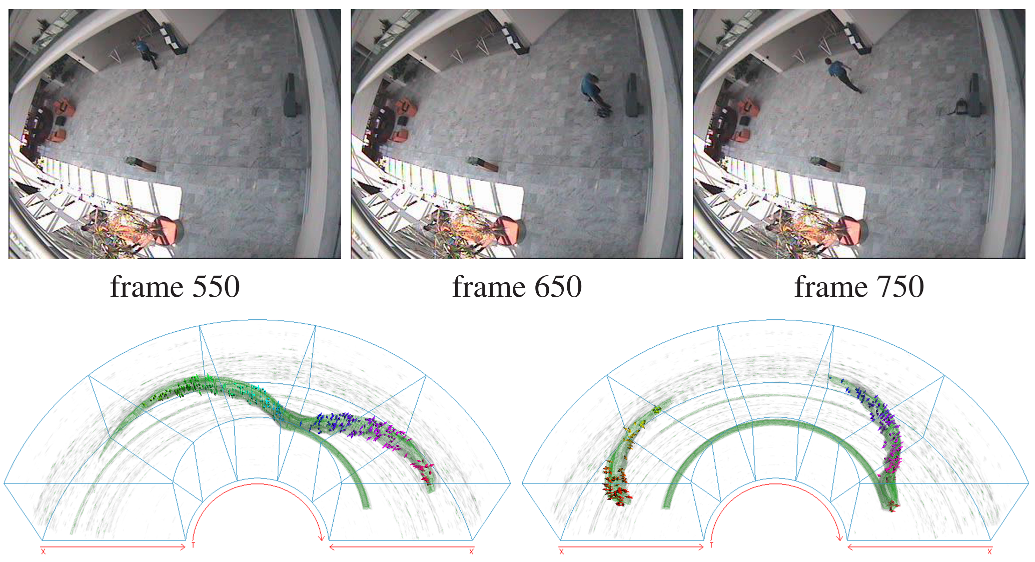


Figure 5.15: The two lower images show the "LeftBag" video rendered as bricked volume horseshoe. The three frames in the upper row present the stages entering, depositing, and leaving. Ensuing reentering and picking up the box can only be seen in the horseshoe visualization. (Images courtesy of Botchen *et al.* [10], ©The Eurographics Association 2006).

collected in the CAVIAR project [41] as benchmarking problems for computer vision. In particular, this collection consists of 28 video clips of the entrance lobby of the INRIA Labs at Grenoble, France, which were filmed from a similar camera position using a wide angle lens. All videos have the same resolution with $384 \times 288$ pixels per frame and 25 frames per second. As all videos are available in compressed MPEG2, there is a noticeable amount of noise, which presents a challenge to the synthesis of meaningful visual representations for these video clips as well as automatic object recognition in computer vision.

The video clips recorded a variety of scenarios of interest, including people walking alone and in group, meeting with others, fighting and passing out, and leaving a package in a public place. Because the camera was located at a relatively high position and almost all motions took place on the ground, the view of the scene exhibits some similarity to the simulated view used in the user study. It is therefore appropriate and beneficial to examine the visual signatures of different types of motion events featured in these videos.

The VVR system is capable of visualizing video streams in real time. With the bricking mechanism, a video stream can be segmented into small time spans, each of which is processed in the video processing sub-system and pushed to the rendering sub-system. The processed multi-field data are then used to update the

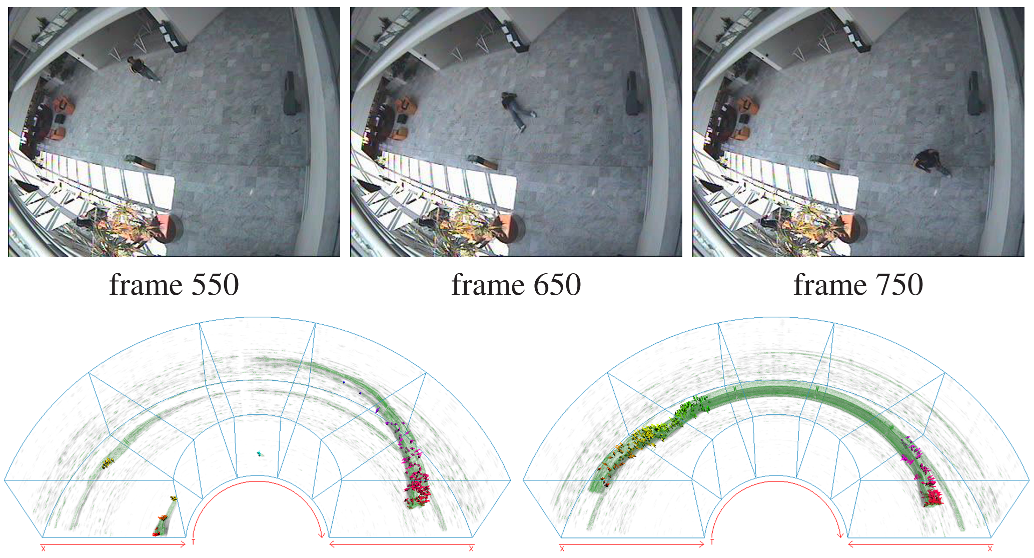


Figure 5.16: Visualization of the “Rest_FallOnFloor” video. The three frames in the lower row show the stages *entering*, *lying*, and *leaving*, which can be clearly seen in the upper horseshoe images. (Images courtesy of Botchen *et al.* [10], ©The Eurographics Association 2006).

visualization. In this way, a continuous video stream can be visualized as either a series of horseshoe images, or one dynamically updated image.

The three images in the top row of Figure 5.15 show the snapshots of three time steps of the “LeftBag” scenario. From the left to the right of the lower images, the horseshoe has been updated four times, i.e., moved by four bricks. The visualization presents a moving object (i.e., a person) that entered the scene and then left an object (i.e., a bag) in the scene before exiting. By observing the glyphs associated with the two objects, the observer can recognize that the object being left in the scene remained stationary until a moving object (in fact the same person) returned and took it away.

Let’s consider the visualization of another video clip shown in the top row of Figure 5.16. In both horseshoe images, a moving object entered the scene and then remained almost motionless for a while before moving again. In comparison with the “LeftBag” video clip, it gets clear that there was only one object. In fact, this particular video shows a drunken man falling on the floor.

In both video clips, each brick covers a time span of about 3 seconds. With the GPU-assisted techniques described in the previous sections, the video visualization rendering system can update the dynamic image for each new brick well below one second. The exact timing for different rendering features is given in Table 5.3. The table demonstrates that flow visualization does not reduce the overall

| viewport | V-S | V+S | V+S+G | V+DS | All |
|---|---|---|---|---|---|
| $800 \times 600$ | 11.04 | 9.63 | 9.63 | 7.40 | 7.20 |
| $1024 \times 768$ | 10.20 | 7.83 | 7.83 | 6.80 | 6.65 |
| $1280 \times 1024$ | 8.64 | 5.47 | 5.47 | 5.13 | 4.56 |

Table 5.3: Performance results in fps for the "LeftBag" dataset with a resolution of 384 × 288 × 1600 pixels. All timings were measured on a PC with 3.4 GHz Pentium 4 and NVIDIA GeForce 7800 GTX (256MB). The table shows six different types of rendering styles: volume without video span (V-S), volume with video span (V+S), volume with video span and geometry (V+S+G), volume with dynamic video span (V+DS), and all rendering features combined. (Courtesy of Botchen *et al.* [10], ©The Eurographics Association 2006).

rendering performance: the video span (i.e., volume) with geometry (i.e., flow) is rendered at the same speed as video span only. This behavior can be explained by the fact that the rendering pipeline of VVR renders the opaque geometry prior to the translucent volume. With depth testing activated, the system makes up for the lost speed for rendering geometry by neglecting parts of the volume occluded by the opaque geometry. The results in Table 5.3 also indicate that the rendering costs are proportional to the viewport size.

Visual signatures of spatiotemporal entities in real life videos can be influenced by numerous factors and appear in various forms. Such diversity does not in any way undermine the feasibility of video visualization, and on the contrary, it rather strengthens the argument for involving the "bandwidth" of the human eyes and intelligence in the loop. The above examples can be seen as further evidence showing the benefits of video visualization.

The outcome of the user study showed that rendering visual signatures are an effective means for conveying objects and their according motions in real-life videos in one illustration. In particular, it showed that human observers can learn to recognize types of motion from their visual signatures. However, the communicated information from those images is rather small and could ideally provide more features about the objects, such as their particular actions or the relations between appearing objects. Further, the user study revealed that the applied deformed horseshoe volume rendering did not only bear vantages. Besides from the better space utilization, the bending of the volume around the time-axis, inverts the direction of the $x$-axis from start frame to end frame. This bending transforms a constant movement e.g., from left to right to a perceived right to left motion, although the actual motion of the object did not change. This fact lead to confusion amongst some attendees. Another negative aspect is the fact that the frames in the middle of the rendered video volume are projected to a minimal space – similar to

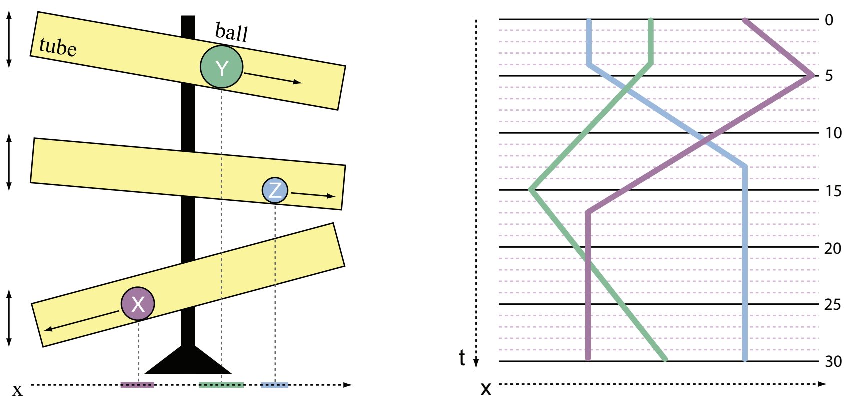


Figure 5.17: The basic scenario of the survey. Left: Differently sized balls rolling in horizontally mounted tubes. Right: The center position of the balls, recorded over a period of time. (Images courtesy of Botchen *et al.* [9], ©2008 IEEE).

looking at the frames from the side – which can lead to a loss of information. Considering these aspects, in the next step the system should be extended to realize a better visual mapping of the video volume, as well as convey more information about the objects in their visual signature.

## 5.6 A Survey on Visual Effects for VideoPerpetuoGrams

Before extending the video visualization framework, a survey of visual effects was carried out to study which kind of additional object attributes can be embedded in the visual signature, and what types of visual effects can be utilized to ensure an appropriate visualization. Section 5.2 exemplifies some typical attributes that may be obtained through video processing, including object identifier, position, size, action type, inter-object relation, and the certainty and error margins of the analytical results. Such processed information may vary in terms of its amount and variety in different applications of video processing. One design aim of the new rendering system is to highlight specific pieces of processed information against the raw imagery information. Designing and selecting a suitable combination of visual mappings for conveying multiple attributes is thus crucial to the effective and efficient use of the design space of visualization.

There are guidelines for using different visual effects, such as color, luminance, scale and symbols, in visualization (e.g., [159] and these expert guidelines provided primary reasoning in the design of the extended multi-field video visualization framework detailed in Section 5.7. In addition to these guidelines, the survey described in this section supported the choice of appropriate visual effects for different attributes and lead to the decision to change the projection of the video volume from the prior perspective horseshoe mapping to a parallel projection of the video cube, rendered with a top view.

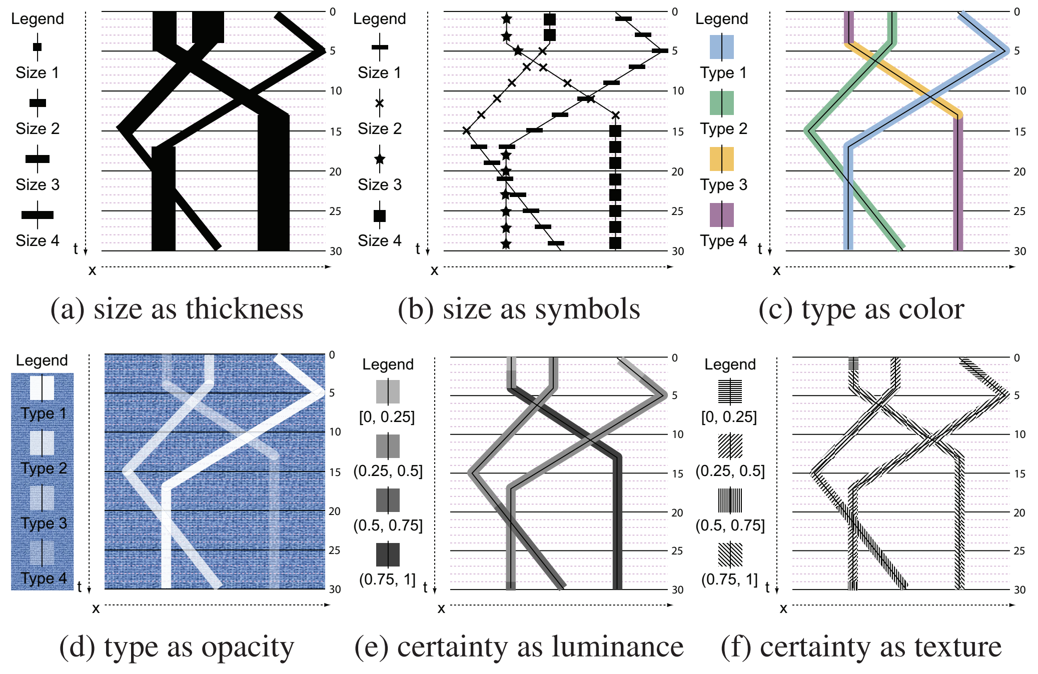
The survey was based on a simple temporal scenario, as illustrated in Figure 5.17, which captures the essence of an action-based video. Assume that $n$ ($n > 0$) balls are placed in $n$ horizontal mounted tubes, whereby each tube can swing slightly up and down independently, causing the balls to roll left or right. The $x$-coordinate of the central position of each ball is recorded over a period of time. Further, each ball is associated with three additional attributes, representing the following three categories of information respectively:

- *Geometric attribute* — a numerical value, such as the size or diameter of the ball.
- *Semantic classification* — normally an enumerated value, such as the type or owner of the ball.
- *Statistical indicator* — a numerical value without an intrinsic geometrical meaning. The recorded information is associated with a statistical value which may be used to indicate the certainty, or importance of the recording. We assume that the statistical indicators fall into the range of [0, 1].

Over the recording period, the three attributes can vary, for instance, the ball can change its type or size. For each attribute, six types of visual mappings are considered, namely *color*, *luminance*, *opacity*, *thickness*, *symbols* and *textures*, giving a total of 18 different attribute-mapping pairings. Figure 5.18 shows six examples of the 18 pairings. To minimize the diversity of the illustrations, each attribute is limited to four nuances. In the first row, the geometric attribute size is mapped onto thickness in (a) and symbols in (b) respectively. In (c) and (d) the semantic attribute type is mapped onto color and opacity respectively. For the opacity example, the noisy background pattern was used to facilitate the perception of different levels of transparency. On a monochrome background, the opacity mapping would have a similar effect as luminance mapping. In the last two images of Figure 5.18, the statistical attribute certainty is mapped onto luminance in (e) and textures in (f) respectively.

To determine the suitability of each pairing, 18 visualization researchers were asked to score the example visualization on a scale from 0 to 10. All voluntarily attending participants are staff members of the Institute for Visualization and Interactive Systems (VIS) at the Universität Stuttgart or the Visualization Research Institute Universität Stuttgart (VISUS). One principal reason for engaging experts in a survey rather than ordinary users through a user study is that the participants of the survey should take into account the following visualization specific criteria:

- *Perceptual effectiveness* — are the visual effects easily recognizable and distinguishable?



(a) size as thickness (b) size as symbols (c) type as color

(d) type as opacity (e) certainty as luminance (f) certainty as texture

Figure 5.18: Example visualizations used for the survey. The images show the three attributes illustrating six different visual mappings. (Images courtesy of Botchen *et al.* [9], ©2008 IEEE).

- *Intuitiveness in association* — can one learn quickly the association between a type of visual mapping and a type of attribute?
- *Visual scalability* — can the scheme extend to a large value range and a large number of balls?
- *Space utilization* — does the scheme require large space or high resolution rendering?

The results of the survey are shown in Table 5.4, whereby the first column of each attribute represents the average score for the chosen mapping and the second column shows the standard deviation. For the three attributes, the best and the second best scores are marked. Nevertheless, it can be assumed that every mapping that scored 5.5 or higher is adequate enough to be used for visualizing the corresponding attribute. The results reveal that color and thickness are the most favored mappings, suitable for conveying changes of most attributes. Opacity and texture are considered to be unsuitable, whilst symbols and luminance are considered only usable for certain types of attributes.

One important use of this survey is to assist in the design of multi-field visualization through an optimized combination of different mappings. For example,

| attribute | type | | certainty | | size | |
|---|---|---|---|---|---|---|
| | $\mu$ | $\sigma(x)$ | $\mu$ | $\sigma(x)$ | $\mu$ | $\sigma(x)$ |
| mapping | | | | | | |
| color | **9.2** | 0.9 | <u>6.6</u> | 2.8 | 4.8 | 2.7 |
| luminance | 4.2 | 2.4 | 5.5 | 2.6 | <u>5.1</u> | 2.1 |
| opacity | 2.2 | 1.6 | 3.5 | 2.4 | 3.5 | 1.7 |
| thickness | 3.7 | 2.6 | **7.3** | 2.6 | **8.1** | 2.4 |
| symbols | <u>6.7</u> | 2.6 | 2.7 | 2.5 | 2.7 | 2.7 |
| texture | 2.8 | 2.0 | 0.8 | 1.1 | 0.8 | 1.1 |

Table 5.4: The table shows the average values $\mu$ of the rating for the six visual effects. The rating ranges from 0 to 10, where 10 is the highest score. The best rated mapping for a property is marked with bold font, the second best result is underlined. In Addition, the standard deviation $\sigma(x)$ for each value is given. (Courtesy of Botchen *et al.* [9], ©2008 IEEE).

if one needs to highlight three attributes simultaneously in the same visualization, one optimal combination is to use color for type, luminance for certainty and thickness for size. However, it may not be straightforward to combine some mappings in the same visualization, for example color with symbols, luminance with thickness and so on. A discussion on how such difficulties are addressed for the development of the extended framework is given in the next section.

## 5.7 The Extended VideoPerpetuoGram Framework

The extended framework for multi-field visualization relies on the survey from the above section in order to provide appropriate visual mappings of video attributes to renderable representations. One issue is that more than just a single video attribute needs to be visualized simultaneously. Possibly interesting video attributes, as extracted by the action recognition stage (Section 5.2.4), include: *geometric information* (the position of an object and its size), *semantic information* (action type and the relationships between extracted objects), and *statistical information* (the plausibility of a recognized action or relationship). In addition to these video attributes, the aim is also to convey the correspondence between the extracted video information and the key frames of the original video stream (as exemplified by Figure 5.20 (c)). Therefore, focus-and-context visualization techniques are employed to combine the display of extracted video attributes with original video frames: the former is the focus, the latter serves as context. The design of the visual representation needs to address a number of challenges:

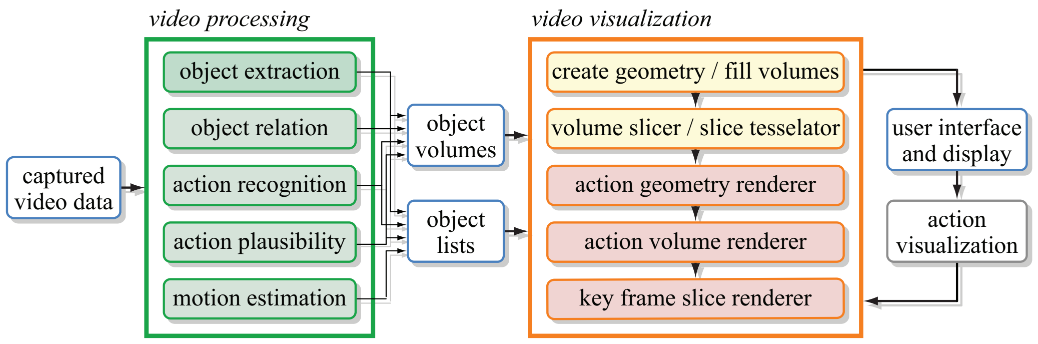


Figure 5.19: The extended system pipeline for multi-field video visualization.

- It must facilitate the continuous depiction, along the temporal axis, of extracted information and key frames for a video stream with an arbitrary length;
- It has to provide a good visibility of motion traces along the temporal axis to facilitate the comprehension of temporal behavior of the objects;
- It should make maximum use of the available screen space in the $xy$-dimension;
- It should try to convey the context information in an intuitively recognizable form, but it should not detract from the focus;
- It should make use of effective different visual attributes to illustrate focus information in different categories.

This section describes an approach that combines such information in an appropriate focus-and-context visual representation, discusses the merits of several typical designs, and presents a combined CPU and GPU implementation to achieve a balanced distribution of the rendering workload.

The technical flow-chart in Figure 5.19 shows the overall architecture of the whole system, which is similar to the previous framework also divided into two substantial sub-systems — *video processing* and *video visualization*. The main objective of this system is to process the input data stream, derive significant information and output a meaningful visualization.

The *video processing* sub-system receives the actual frames of the continuous video stream. The purpose of this sub-system is to extract several different attributes of a video, using image processing filters as described in Section 5.2, with the main focus on object extraction and the classification of the performed action for the recognized objects. Therefore, a recognition filter [35] is adapted and extended it to give a premise about the possible relation of objects in the scene. Details of the individual modules are discussed in Section 5.2.4. The output files contain the object IDs, the center positions, the action with the estimated plausibility, the motion direction and the relation between objects.

With the gained information, the *video visualization* sub-system synthesizes a meaningful visual representation for the given time span of the video stream, using volume rendering in combination with glyph geometry and video snapshots. The main goal is to clearly represent different actions, or motions of people detected in a scene, and to emphasize a possible relation between them, i.e., grouping of several people. Further, the functionality of this system utilizes real-time visualization and an extended frame-by-frame volume bricking mechanism – derived from the previously described volume bricking – to enable the handling of large video data.

***VideoPerpetuoGrams*** For rendering the *VPGs* in Figure 5.21 the projection parameters of the framework had to be changed. A different viewing angle and a sheared volume in $z$-dimension minimizes the problem of overlap and self-occlusion. A inclination of $-49$ degrees in the $z$-axis, with a shear of $45$ degrees in the same dimension, and a modified field of view by $\frac{\pi}{10}$ in combination with a parallel projection generates satisfying results.

This enables the system to continuously render the incoming video stream, writing the visualization result to an output buffer. For every 300 incoming video frames, an image is rendered, where the middle section (including the second and third key-frames) is cut out and appended to the last rendering result, thus generating an endless output stream of a video. Using this technique, it is now possible to see all the performed action sequences and relations that appeared in a video in one continuous illustration at one glance.

### 5.7.1 Illustrating Focus Information

The discussion on visual mapping of the focus information includes the recognized video attributes, in the following order: object position and size; action type; relationship information and plausibility.

***Object Position and Size*** VPG is intrinsically linked to a view of the space-time of the video that uses the vertical axis for time and the horizontal axis for the horizontal spatial dimension of the video frames, as illustrated in Figure 5.20. The second spatial video dimension is essentially mapped to the vertical axis by tilting the video frames. In this way, the mapping of the geometry of space-time is fixed to that interpretation of the axes of the visualization image.

Therefore, the only appropriate mapping of object position is to respective spatial coordinates in the VPG image. Figures 5.20 (a,d) show examples where

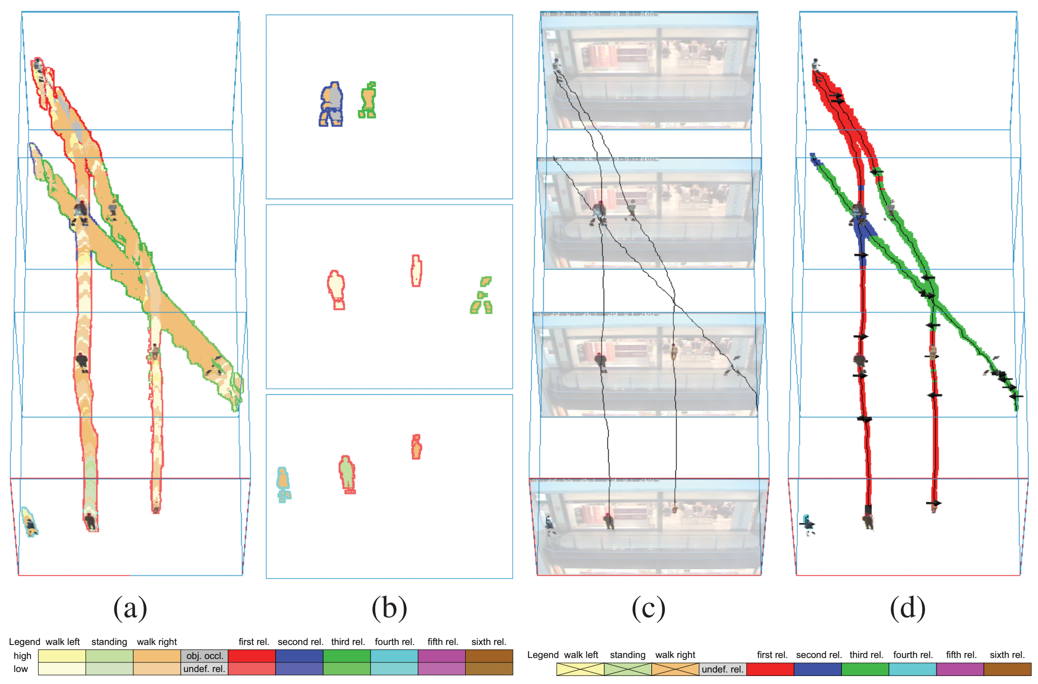

(a)
(b)
(c)
(d)
Legend walk left standing walk right first rel. second rel. third rel. fourth rel. fifth rel. sixth rel.
high
low
obj. occl.
undef. rel.
Legend walk left standing walk right first rel. second rel. third rel. fourth rel. fifth rel. sixth rel.
undef. rel.

Figure 5.20: The “OneShopOneWait1front” dataset. Volume with 300 frames, starting at frame 355 (a). Three object frames (455, 555 and 655) show the object actions and relations encoded as colors (b). The OneStopEnter2front dataset showing object timeline reconstruction (c) and additional action geometry (d). (Images courtesy of Botchen *et al.* [9], ©2008 IEEE).

the object positions are marked by thick bands, Figure 5.20 (c) shows an example that uses only a thin curve to indicate the center of the object’s position.

The expert survey indicates that the object’s size should be mapped to thickness. Figure 5.20 (a) uses this mapping strategy: the thickness of the object’s trail reflects the projected size of the object. For object size, no other visual attribute is used for mapping; the only alternative is that object size is not visualized at all, depending on user preference. However, size can be used to map other attributes adequately, as shown in Figure 5.20 (d), where the extent of the object relation is mapped to the thickness of the trace.

***Actions*** One attribute of the action classification describes the type of action. In the survey, the most highly ranked mapping for action type is the mapping to color. Therefore, color mapping is recommended as an appropriate method to illustrate the different action types. The color map is applied to the trail of the object in order to couple object position and action type. Usually, the number of action types is small, in the range of 10 types, so that the perceptual grouping through color hue can be used to clearly distinguish different actions. Figure 5.20 (b) illustrates the effect of this color mapping for single frames, based on the color table, given as legend under every image (orange: walk to the right; yellow: walk to the left; green: standing; dark grey: overlapping).

The design of the color map is based on previous work and recommendations by Ware [158], Healey [56], and Kindlmann *et al.* [73]. The other highly ranked mapping for action type uses symbols. Figure 5.20 (d) shows an example of the glyph-based visualization of action type. Glyphs are distributed along object trails, indicating the action type at respective space-time positions. The design of the glyphs adopts established icons from flow visualization (i.e., arrows indicating motion direction; additionally, a square icon indicating an object at rest). Depending on the results of the survey, it is recommended using either color or glyphs for action types. In particular, glyphs are useful when color is already needed for the visualization of another video attribute.

***Relations*** Relationship data is another kind of action-related information attached to the video volume. The strength of the relationship between two objects is based on their distance, relative speed, relative direction of motion, size of their bounding boxes, and overlap of their bounding boxes. Typically, relation information should be displayed together with action type, i.e., facing the issue of simultaneous visualization of several attributes. More importantly, both action type and relation are most appropriately visualized by color. Color is particularly suited for relation visualization because color is effective in building visual correspondence and grouping.

To overcome the conflict that two different video attributes are well visualized by color, the following strategies can be considered. The first strategy is to spatially separate the colored image regions, as illustrated in Figure 5.20 (a,b): action type is shown by color attached to the center of the object's trail, whereas relation information is color-coded within a surrounding silhouette line. Figure 5.20 (b) shows that the relation data forms a silhouette around the action type representation. The color table for these relation silhouettes is shown in the legend of the figure. For example, the red silhouette indicates the relation of the first two people entering the scene in Figure 5.20 (a), and walking besides each other. The issue with this strategy is that more visualization space is needed for spatial separation. Therefore, this strategy is appropriate if enough screen space is available.

The other strategy is to use a different mapping for action type. As discussed above, glyphs are a suitable alternative for the visualization of action type. Then, color is available for relation visualization. This strategy is illustrated in Figure 5.20 (d). Here, the colored trace represents the relations of objects, whereby the actions are mapped on different glyphs.

***Plausibility*** One of the reasons for using visualization for video analysis is that computer-vision techniques are not always capable of fully analyzing videos with high certainty. In fact, the action recognition algorithms provide a certainty—or plausibility—attached to recognized action type and relation information. The certainties of both video attributes as statistical attributes are included in this new visualization approach. The expert survey indicates that mapping to thickness or color should be most effective for these statistical attributes. The system supports both alternatives. Figure 5.20 (d) shows how thickness is used for the continuous mapping of the certainty of the relation type.

In contrast, Figure 5.20 (a) illustrates the mapping of all attributes to color only. Here, saturation is used to indicate plausibility: high saturation corresponds to high certainty, low saturation to low certainty. Only two different saturation levels are used in order to allow for an accurate visual discrimination. The plausibility value is mapped to the quantized saturation value by thresholding. The advantage of the saturation mapping is that it can be immediately combined with a hue mapping. In Figure 5.20 (a,b), both action type and relation information are encoded by hue (in the main trail and the silhouette regions, respectively); then, the plausibility values for action type and relation information are mapped to respective saturation values.

### 5.7.2 Conveying Context Information

The above video attributes alone can communicate the performed action of persons and their relations in the scene at any given time, but they are restricted in the amount of detailed information they can convey because visual information about the environment is lacking. Therefore, the spatial context of the surrounding environment is missing. For example, it would be difficult to say where exactly an object is located in the scene at a certain time step. To increase the amount of information communicated to the viewer, and thus enhance the understanding of the events in the scene, the system provides a set of additional visualization options that facilitate the display of surround information. This additional visualization is combined with the focus visualization of video attributes in a focus-and-context approach.

***Snapshots*** Original frames of the video contribute the most information possible. They pull together extracted objects and their surrounding. Similarly, they can indicate important static objects in the scene – i.e., a door or an obstacle – that might cause persons to act in a certain way. The system allows for several video frames to be placed as snapshots at any position in the volume as shown in Figure 5.20 (c) and Figure 5.21. The useful number of snapshots that can be displayed simultaneously depends on the available screen space: the frames should be so far apart that they do not overlap on the final visualization image.

Focus-and-context rendering is achieved by blending the snapshots with a depth-dependent alpha value over the visible volume signature. Depth-dependent blending enhances depth perception. All objects that are visible in a snapshot are rendered with full opacity to stand out from the background.

***Timeline Reconstruction*** The object's path evolving in time is reconstructed by tracing a line through the objects center in every frame where the object appears as shown in Figure 5.20 (c). This is useful to keep track of objects that can occlude each other, if their paths are crossing. To visually enhance the progress of time, which increases from back to front along the $z$-axis (equivalently from top to bottom).

### 5.7.3 Efficient Focus and Context Rendering

For the efficient visualization of action based video volumes, the video visualization application employs CPU and GPU methods to achieve real-time rendering. The visualization framework is built upon an existing slice-based volume renderer [156] that has been modified for video volume rendering [10]. An advantage of this framework is its separation of different visualization aspects into different software components.

This section discusses the technical details of action-based video rendering modules that are used to generate the two presented visualization styles. Further, it exemplifies the accomplished extensions that have been applied to the framework, to produce a continuous video visualization.

***Two-pass Silhouette Rendering*** Constructing the opaque silhouettes around the opaque object traces is managed by a two-pass GPU-rendering procedure. Both information – the silhouette with the object relation and the thereof enclosed region, holding the action type – are stored in every single video frame, as shown in Figure 5.20 (b). By volume rendering both information in one pass, the opaque silhouette would completely occlude the interior action information. As the visualization should show both attributes simultaneously, the approach is to first render the complete volume, only blending the relation silhouettes to the framebuffer. Then, in the second pass, the whole volume is rendered again, but this time only the interior is blended as opaque color to the framebuffer, generating the desired result. The advantage of this technique is the good structured visual result, with the clear separable inner and outer regions. Since the volume has to be rendered twice, a drawback is the bisection of rendering performance, making this technique not very useful for a real-time system that could be built with today's hardware.

***Additional Action Indication*** Regarding the issues of the prior technique, investigation in another combination of visual mappings was performed, guided by the expert survey, to enhance the system. This technique, as shown in Figure 5.20 (d), is implemented as single-pass volume rendering with additional glyphs to indicate the objects action. For this technique, the volumetric representation of the object trace indicates the relation of objects, illustrated with the same color. The plausibility of the relation is mapped on the thickness of the trace, thereby neglecting the size of the objects itself. In the context of this particular work, this choice seems adequate, as the video processing module was designed for tracking only one kind of object, i.e., persons that are nearly equal sized.

The motion glyphs are generated on the CPU and rendered as opaque geometry over the object traces, whereby a square represents a standing person, and arrows indicate a walk left or walk right respectively. This technique has the advantage that it maps both information to different visual attributes and achieves high framerates that are required for a real-time application. However, due to noise in the extracted video frames that can cause many small changes in the action recognition (e.g., multiple changes from left to right caused by a standing person that is slightly fluctuating), this can lead to a multitude of rendered glyphs that occlude each other. This is overcome by thresholding with a user defined value. The threshold regulates the maximum relative change of an object between two frames, required to generate a glyph at this location.

## 5.8 Evaluation of VideoPerpetuoGram Visualization

The system was tested with several datasets that contained a different amount of appearing people and various groupings, thus forming interesting relations. The evaluation of time for all images in the presented figures runs from back to front of each figure. The shopping mall datasets have been recorded with 25 fps, all video volume illustrations in Figure 5.21 represent 44 seconds of the according video. The illustrations have been rendered with the bricked volume approach, meaning that one brick between two snapshots contains 100 video frames, thus representing four seconds of the video.

The VPG in Figure 5.21 (left) represent the "OneShopOneWait1front" dataset. In this scenario, two people enter the scene from the left side, walking to the store. While one person enters the store, the other one is waiting outside in front of the window, before his partner returns, whereupon they leave the scene. In the meantime, two other people cross the scene from left to right with different speeds, one after another. In addition to the object actions in the interior, one can see the relation silhouettes here. Both persons that entered together, walking besides each other, have the same colored silhouette (dark red) at the beginning and a mainly orange colored action trace, indicating that they walk to the right. As they drift apart, the dark red changes to a light red, since the relation plausibility falls under the threshold value. Both crossing people start with a different colored silhouette, causing all other silhouettes to change the color over time. This is due to the fact that their relation changes from one to the other group member, while passing by. The relation of the right person of the former group in Figure 5.21 (left) changes to a relation to the crossing person (green). This is caused by a similar speed and motion direction, as the other group member stopped in front of the window, minimizing the relation plausibility. His relation in turn becomes very strong to the crossing person, as they get very close and their bounding boxes overlap (blue). The second passerby changes the silhouettes from cyan for the first group member to magenta for the second group member. Then, the relation of the two people who formed a group before is increasing again, resulting in a dark red silhouette, as they leave the scene.

Figure 5.21 (middle) shows a very similar scenario as the left image. Again, two people step into the scene together from the left side, one entering the shop, the other one waiting outside. But this time, the third person that enters the scene to confuse the system, appears just a few frames after the others, walking with approximately the same speed in the same direction, thus the person is always resided very close to the group. As well as in the first scenario, the relation filter always scores higher for the two people that entered together, assigning the third person a different color for a differing relation. Only as their paths cross and thus, the bounding boxes overlap, The relation to the crossing person becomes stronger. The red relation silhouettes of the group in Figure 5.21 (middle) is completely disappearing, as one partner enters the store and vanishes inside. There does not exist a relation for one single person. But as he steps outside the shop, the relation silhouette reappears too.

The most complex case that was used to verify the system is the “TwoEnterShopfront2” dataset illustrated in Figure 5.21 (right). In this scenario, seven people appear in the foreground of the scene nearly at the same time, crossing paths, forming groups and splitting up again. In the beginning, two people cross the site from different directions with unequal speeds. They both start with a relation to the third person inside the shop, but their relation score is highest as they approach each other and cross, which is mapped to green. Then, two people enter and cross their ways, shaking hands as they meet, indicated by the magenta relation color. At the same time, a group of two other people appears from the left side, walking besides each other and entering the store. The relation of the meeting people is colored in green. Notice that relation colors that have been used already are locked, but can be reused after a certain time span has passed. Since there is only a limited amount of clearly separable colors, which can be exceeded by the amount of possible relations, a light grey is used as reserve color that is assigned to relations with low plausibility. In fact, only the six highest relations are

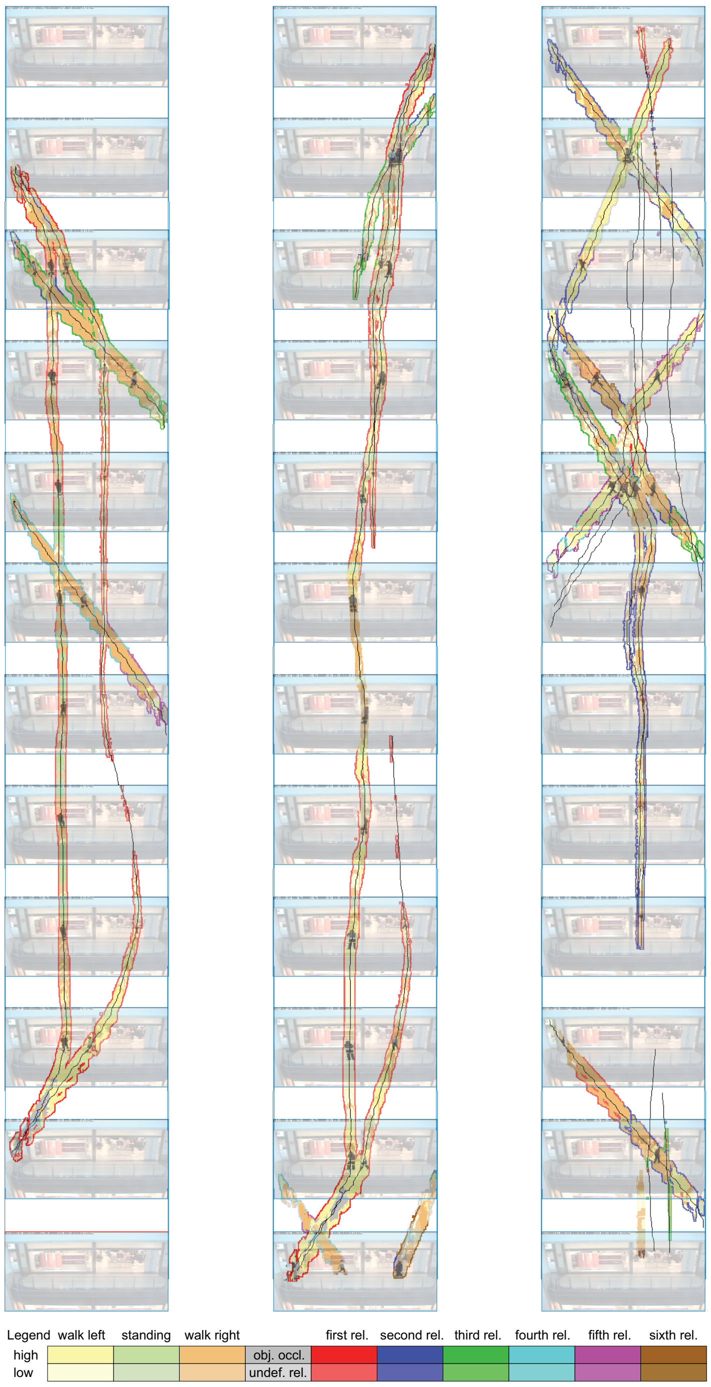


Figure 5.21: *VideoPerpetuoGrams* of the datasets “OneShopOneWait1front”, “OneShopOneWait2front” and “TwoEnterShop2front”. The en bloc visualization represents 44 seconds of the videos. (Images courtesy of Botchen *et al.* [9], ©2008 IEEE).

| viewport | $768^2$ | $1024^2$ |
|---|---|---|
| Volume | 37.4 | 21.6 |
| Volume+Glyphs+Key Frames+Shear | 20.3 | 11.5 |
| Volume+Glyphs+Key Frames+Shear+Bricking | 11.2 | 7.9 |
| Two-pass Silhouette Rendering | 5.6 | 4.0 |

Table 5.5: Performance results, in fps for a video volume with a resolution of $300 \times 384 \times 288$ voxels. All timings have been measured on a desktop PC with an AMD Athlon 64 X2 dual processor with 2.4 GHz, 2 GB of RAM and a NVIDIA GeForce 8800 GT (512MB) graphics card. (Courtesy of Botchen *et al.* [9], ©2008 IEEE).

mapped to color, and all other relations that exist at the same time are indicated by grey. This happens in the middle of Figure 5.21 (right), as many people appear in the scene at the same time, each one having a relation to every other person in the scene.

The performance results for the combination of all presented modules is given in Table 5.5. The table shows that the framework can perform with interactive frame rates, even for a high resolution display. A limitation of the system is the illustration of scenes with a large amount of people appearing at once. In such scenes it can be that most parts of some objects are occluded by other objects, thus leading to a problem for the object tracking, as well as to overlap and occlusion problems for the visualization. Here, the system is as good as the used computer vision algorithm and tracking technique. The object tracking could be improved, by using only the people's heads for the recognition, since those parts of the bodies should be visible most of the time. From the visualization side, the illustration of an object trace could be minimized to a line, for objects that are not of great interest, only using glyphs and silhouettes for the important objects or groups. This could be pre-decided by an algorithm, using the relation probability, or it could be left as an option for the user to select the objects of interest.

# Chapter 6

# Visualization of Encoded Multi-field Data

The size and scale of data that is generated and gathered with nowadays complex simulation and measuring techniques is increasing at an alarming rate, creating a data deluge for analysts wanting to visualize, interactively manipulate, and explore the problem at hand. Moreover, most current visualization techniques are datagrid specific and cannot allow scientists and researchers to interactively visualize various unstructured and scattered large-scale datasets in a single system on their desktop computers. One major issue for the visualization of large datasets is the data transport from the CPU to the GPU and the data storage in the limited graphics memory. To overcome this bottleneck, one possible approach is presented in this chapter. For the visualization of large, structured data streams, a bricking based strategy can be applied to upload only the required subset of data to the GPU memory for rendering, as discussed in the previous chapter. This strategy however is only applicable if the problem domain can be subdivided and the analysis task is of local nature.

For analysis tasks that consider the whole data domain, such as it is common in feature based flow analysis as shown in Chapter 3, or medical imaging as described in Chapter 4, another solution is required that allows the whole information contained in the data to be stored in GPU memory at a once. A feasible approach is data storage reduction by means of data encoding, with the aim to find a data layout that suits on graphics hardware memory and enables the utilization of the programmable fragment processing unit for a simple and interactive on-the-fly decoding during rendering.

This section describes a new technique for procedural encoding and visualization of structured, scattered and unstructured multi-field datasets using Radial Basis Functions (RBF) and Elipsoidal Basis Functions (EBF) with great potential for compactly representing large datasets. One advantage of RBF encoding technique is the reduced storage requirement that allows the compressed datasets to completely reside in the local memory of the graphics hardware, thus, enabling accurate and efficient processing and visualization without data transfer problems. Within computer graphics RBFs have been widely used for surface and functional approximations due to their symmetry property. They have been applied in a broad field of research including surface modeling, geometric modeling, spatial interpolation in GIS applications and visualization of 3D scattered data [22; 46; 55; 129; 154; 173]. Previously, Jang et al. [64] and Co et al. [26] have used RBFs to procedurally encode both scattered and irregular gridded scalar datasets. The approach described here extends their methods and suits for interactive visualization and feature detection of large scalar, vector, and multi-field CFD datasets. The algorithm is well-suited for structured and unstructured data representations and thereby applies well to meshless CFD methods. Further, feature detection techniques, as introduced in Chapter 3, are applied for detecting and rendering features the RBF space by utilizing the functional representation for efficiency.

The primary advantages of RBFs and EBFs include their compact description, ability to interpolate and approximate sparse, non-uniformly spaced data, and analytical gradient calculations. Common choices for the RBFs are thin-plate splines, multiquadrics, and Gaussians. Splines have no adjustable parameters and do not have local support, thus leading to a denser system of equations necessary to solve. Inverse multiquadrics have been proven to have a physically relevant foundation [55]. However, Gaussian RBF and EBF models offer several advantages. They are concise, robust, and have a regular and smooth behavior outside the fitting domain, providing a localized function through which local data features are preserved. The Gaussians also offer the additional advantage of being less expensive to reconstruct on modern graphics hardware.

The work presented in this chapter was carried out in collaboration with Yun Jang, Jingshu Huang and David S. Ebert from Purdue University; Kelly P. Gaither from the University of Texas at Austin; and Manfred Weiler and Andreas Lauser from the Universität Stuttgart. Yun Jang must be specially credited for the RBF and EBF encoding of all datasets, while Manfred Weiler was responsible for the basic implementation of the visualization framework and Andreas Lauser helped with further code tuning. The presented methods were published in the Computer Graphics and Applications journal [161] and in the Computer Graphics Forum journal [63].

## 6.1 Radial Basis Functions

Radial basis functions are circularly-symmetric functions centered at a single point. RBF encoding creates a complete, unified, functional representation of the data field throughout three-dimensional space, independent of the underlying data topology, and eliminating the need for the original data grid during visualization. Unlike other data compression techniques like iterated function systems [4] or wavelets [106], the compact representation of the basis function brings the advantage of a simple evaluation, as well as a smoothing of the first and second derivative, in case of noisy data. RBFs play an important role in a broad research area. Researchers have proposed several different polynomial functions to approximate the construction of *soft objects* [171], or *solid geometry* applied to implicit surfaces [71]. Recently, Chen [23] compared several different radial basis functions that are suited to combine point clouds and conventional volume objects. Jang et al. [64] and Weiler et al. [161] presented techniques for spatially-limited spherical Gaussian radial basis function encoding and visualization of volumetric scalar, vector, and multi-field datasets. To provide a greater variety of functional approximation and modeling options, researchers proposed to use ellipsoids instead of RBFs. Ellipsoids have been used in many research areas, such as segmentation [111], object detection [47], and filtering of noisy data [133]. In the area of data fitting, Calafiore [20] proposes an approximation of $n$-dimensional data using ellipsoidal primitives and showed that a "distance-of-squares" geometric error criterion gives stability for Gaussian noise. This approach, however, is limited to several hundreds of points because of numerical problems. Li et al. [118] proposed a fitting method using ellipsoids for implicit surfaces with a limited number of data points. Recently Huang et al. [61] showed a shape-based approach for thin structure segmentation and visualization in biomedical images using an ellipsoidal Gaussian model.

Mathematically, for given data samples $p_j = (\mathbf{x}_j, \mathbf{y}_j)\ ,\ j = 1, ..., n$, where the values $\mathbf{x}_j \in \mathbb{R}^3$ are the spatial locations and the values $\mathbf{y}_j \in \mathbb{R}^k$ are the data values that exist at the corresponding spatial locations, the data values can be approximated with a function $f(\mathbf{x})$ defined as weighted sum

$$f(\mathbf{x}) = \sum_{i=1}^{N} \lambda_i \phi\left(\mathbf{x}, \mu_i, \sigma_i^2\right) \quad , \tag{6.1}$$

where $N$ is the number of basis functions, $\lambda_i$ the weight, $\mu_i$ the center, and $\sigma_i^2$ the variance of a single basis function $\phi$. With this mathematical expression, $\lambda$, $\mu$ and $\sigma^2$ are optimized to find the best approximation of the original data values.

Depending on the application there exist several types of basis functions $\phi$ that could be employed, ranging from multiquadric functions to biharmonic or triharmonic splines. This chapter exclusively applies spherical and ellipsoidal Gaussian functions because of their above stated advantages. The strategy of data encoding and decoding for visualization is described in the following sections.

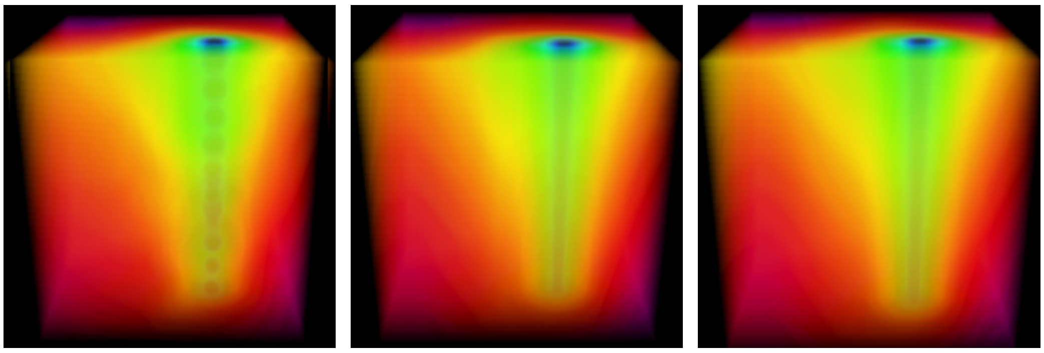


Figure 6.1: Visual comparison of basis functions using an oil reservoir data set. The left image is the result using the spherical Gaussians, the middle image using the axisaligned ellipsoidal Gaussians and the right image using the arbitrary directional Gaussians. (Images courtesy of Jang *et al.* [63], ©The Eurographics Association 2006).

### 6.1.1 Spherical and Ellipsoidal Gaussians

As previously mentioned, the radial basis function is a radially symmetric function where the RBF volume has a spherical shape. For volume visualization, several researchers [64; 161] have used spherical Gaussians as basis functions. Using a Gaussian RBF, the data approximation function is of the form

$$f(\mathbf{x}) = \sum_{i=1}^{N} \lambda_i e^{\frac{-\|\mathbf{x}-\mu_i\|^2}{2\sigma_i^2}} \quad , \tag{6.2}$$

where $\lambda_i$ are the RBF blending coefficients or Gaussian weights, $\mu_i$ are the RBF centers, $\sigma_i^2$ are variances or Gaussian widths, and $N$ is the number of basis functions.

The spherical shaped basis function, however, has a limitation in fitting long, high-gradient shapes, for example cylindrical shapes. The radius might reach the shortest boundary of the area and require many small RBFs to fit one long shape. Figure 6.1 (a) visually displays this artifact for spherical RBFs. In the worst case, many discrete blobby structures can be shown in visualization instead of a visually accurate approximation.

In order to reduce these artifacts, ellipsoidal Gaussians – which are ellipsoidal basis functions (EBF) – can be used instead of the spherical Gaussians. There are two kinds of ellipsoidal Gaussians: axis-aligned and arbitrary directional EBFs. Figure 6.1 (b) shows the result of using the axis-aligned ellipsoidal Gaussian and Figure 6.1 (c) shows the use of arbitrary directional Gaussians. As can be seen, the narrow shape is well represented by both ellipsoidal Gaussian functions. Additionally, only a smaller number of basis functions is needed to evaluate for ren-

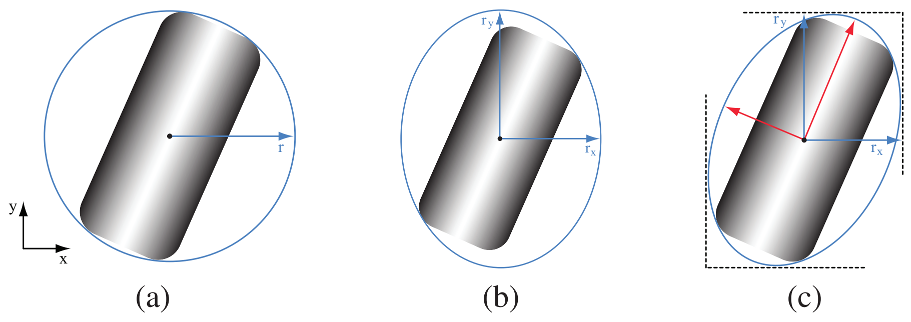


Figure 6.2: Comparison of basis functions for approximation of the grey data with three basis functions: (a) RBF; (b) axis aligned EBF; and (c) arbitrary directional EBF. The influence range of each basis function is shown as blue arrows and black curve for volume subdivison. Red arrows indicate the orientation of arbitrary directional EBF. (Images courtesy of Jang *et al.* [63], ©The Eurographics Association 2006).

dering each pixel, when using an octree data structure for spatial subdivision. Cuboids are used to generate the spatial cell distribution rather than cubes and they are aligned according to the influences along the axes. A comparison of the three basis functions is shown in Figure 6.2. This figure shows a long diagonal data distribution and the influences of the three basis functions are drawn overlaid on the data. As can be seen, the enclosed volume of the ellipsoidal representations is smaller and fit better around the structure to be encoded and thus, they are superior for reconstructing the structure in the rendering step.

The ellipsoidal Gaussian basis function in 3D space can be represented in a matrix form using the *Mahalanobis distance* instead of the *Euclidean distance*

$$\phi(\mathbf{x}) = \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\mathbf{V}^{-1}(\mathbf{x}-\mu)\right) \quad , \tag{6.3}$$

where $\mathbf{V}$ is the covariance matrix, $\mathbf{V}^{-1}$ is positive definite and is defined by a rotation matrix $\mathbf{R}$, and a scalar matrix $\mathbf{S}$, as

$$\mathbf{V}^{-1} = \begin{bmatrix} a_1 & a_4 & a_6 \\ a_4 & a_2 & a_5 \\ a_6 & a_5 & a_3 \end{bmatrix} = \mathbf{R}\cdot\mathbf{S}^{-1}\cdot\mathbf{S}^{-1}\cdot\mathbf{R}^T \quad ,$$

where $\mathbf{x}$ is a coordinate vector, $[x\ y\ z]^T$ and $\mu$ is the center vector of an ellipsoidal Gaussian, $[\mu_x\ \mu_y\ \mu_z]^T$. Moreover, for a rotation matrix, $\mathbf{R}^{-1} = \mathbf{R}^T$ and for a scaling matrix $\mathbf{S}^T = \mathbf{S}$.

Since all rotation angles are set to zero for the axis aligned ellipsoidal Gaussians, off-diagonal components in $\mathbf{V}^{-1}$ are zero and diagonal components are positive, $a_1 = \frac{1}{\sigma_x^2}$, $a_2 = \frac{1}{\sigma_y^2}$, $a_3 = \frac{1}{\sigma_z^2}$, and $a_4 = a_5 = a_6 = 0$. Therefore, the axis aligned ellipsoidal Gaussian can be represented as

$$\phi(\mathbf{x}) = \exp\left\{-\frac{(x-\mu_x)^2}{2\sigma_x^2} - \frac{(y-\mu_y)^2}{2\sigma_y^2} - \frac{(z-\mu_z)^2}{2\sigma_z^2}\right\} \quad . \tag{6.4}$$

On the other hand, in the arbitrary directional ellipsoidal Gaussian, off-diagonal components may not be zero. Therefore, the components of $\mathbf{S}$ and $\mathbf{R}$ are considered, instead of the direct form in Equation (6.4), to make sure that $\mathbf{V}^{-1}$ is positive definite.

## 6.1.2 Functional Approximation

To generate a new functional representation for a given dataset, the realization of a multi-level encoding process as shown in Figure 6.3 is required. The aim of functional approximation is to find the best parameters for the basis functions to fit the data values as closely as possible. This means that for a given set of data points $\mathbf{p}_j = (\mathbf{x}_j, \mathbf{v}_j)$ , $j = 1, \ldots, n$, the function $f(\mathbf{x})$ approximates the data domain within a certain user specific error range. The $p_j$ can be vertices in a structured or a unstructured grid, or even belong to a point based representation. In Section 6.1.1, a functional approximation using RBFs is presented. This section exemplifies the extended approximation process for the more complex EBFs. For the extension from RBFs to EBFs, the covariance $\mathbf{V}$ is used rather than the single variance in the functional approximation. The function using $\mathbf{V}$ is redefined as

$$f(\mathbf{x}) = \sum_{i=1}^{N} \lambda_i \phi\left(\mathbf{x}, \mu_i, \mathbf{V}_i\right) \quad . \tag{6.5}$$

The parameters include centers $\mu$, covariances $\mathbf{V}$, the weights $\lambda$ and $N$ is the number of basis functions, respectively.

The approach of functional approximation is based on previous RBF work [64; 161]. The number of EBF parameters is larger than the number of RBF parameters, which leads to a different approach for approximation and a slightly changed memory layout for data storage, as shown in Section 6.2.1. In particular, for the 3D scalar approximation, 2 more parameters are added for the axis aligned ellipsoidal Gaussians and 5 more parameters for the arbitrary directional ellipsoidal Gaussians, compared to spherical Gaussians. For 3D vector approximation, 6 more parameters are needed for the axis aligned ellipsoidal Gaussians and 15 more parameters for the arbitrary directional ellipsoidal Gaussians.

A nonlinear optimization algorithm is used to find the best parameters for the approximation. This algorithm quickly computes reasonable initial starting parameters using the given data. This provides better results in the nonlinear optimization and these initial parameter values are inserted into the main optimization algorithm. The algorithm determines the parameters for only a single basis

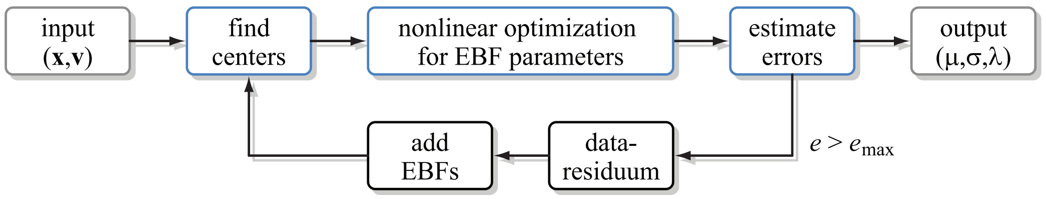


Figure 6.3: Flowchart for data encoding with ellipsoidal basis functions.

function at each iteration. Once the parameters are found that reduce the approximation error the most, the influence of the basis function is removed from the data values and the residual functional values (errors) are computed, using either L2-norm or H1-norm. These residuals are then used in the next iteration. The above approximation system is iterated until the error tolerance is satisfied. An error of 5% of the maximum data value is used as error tolerance. However, RMS errors are typically less than 1-3% in the resulting datasets. Once the best functional approximation is found, the volume is divided into several small subvolumes using an adaptive octree structure based on the basis function distribution (see Section 6.2.1). This allows us to improve rendering speed. Moreover, this algorithm can be applied to both scalar and vector datasets (see Section 6.1.3). The resulting encoding algorithm as shown in Figure 6.3 consists of the following steps:

1. Perform domain localization on the original dataset to obtain smaller domain datasets with a limited number of data points.

2. Perform a multi-level least-squared fit of the data using Equation (6.2) in each domain. In each level, the following steps are executed sequentially:

   (a) Find the Gaussian centers with filtering and clustering techniques and set the initial Gaussian weights and widths.

   (b) Perform nonlinear optimization on Gaussian widths to minimize least-squared errors in the domain, and solve the linear system for Gaussian weights.

   (c) Compute the errors at all data points in this domain. If the maximal error is less than a predefined threshold, stop encoding the current domain and go to the next domain; otherwise, encode the residual error as a new dataset starting at Step 2(a).

***Domain Localization*** Encoding large datasets is computationally expensive and the encoding system might be ill-conditioned. Domain localization can be applied to solve this problem. Domain localization was used by Nielson [107], whose method localizes the domain by multiplying by a weight function for each evenly decomposed subvolume. These spatially decomposed subvolumes, however, may generate very sparse subdomains, which do not have enough data points to be fit. Additionally, some subdomains might contain a large number of data points to be encoded.

Therefore, a k-d tree is used to decompose the volume into overlapping subdomains, each with an equal number of data points. Further, the weighting functions are also defined for the overlapping regions between subdomains. Users can predefine the maximum number of data points in each domain and the size of the overlap to which the weighting function is applied, according to their available computational power. This domain localization provides a reasonable number of data points for the encoding system. Theoretically, one subdomain is independent of others with zero error encoding. Since the encoding system is an approximation, non-zero error may exist and may be largest in the overlap areas because of the addition of subdomain encoding errors. This problem is addressed by re-encoding the residual dataset error from the first-round encoded dataset.

***Initial Parameter Approximation*** The initial Gaussian starting parameter values for nonlinear optimization are important for the optimization convergence [14]. The starting center value is set to the maximum data value point and the weight is set to the data value at this center. Since the residuals are computed after every iteration, they become the new data values in the next iteration. Therefore, the maximum error point is the maximum value data point. The position of this data point is set to the center of the basis function and the maximum value at the point is set to the weight of the basis function for either the radial or ellipsoidal basis.

There is only one variance in an RBF, which can be written as

$$\sigma_{ij}^2 = \frac{r^2}{2\ \ln\left(\left|\frac{\mathbf{v}_i}{\mathbf{v}_j}\right|\right)}\ ,\ j = 1, ..., N \quad , \tag{6.6}$$

where $\mathbf{v}_i$ is the data value at the center and $\mathbf{v}_j$ is the value at the $j^{th}$ data point assuming that $\text{sgn}(\mathbf{v}_i) = \text{sgn}(\mathbf{v}_j)$ and $|\mathbf{v}_i| > |\mathbf{v}_j|$. $r$ is defined as $|x_j - \mu_i|$ and $N$ is the number of points. Then $\sigma_{ij}^2$ with $\sigma_i^2 = \sum_j \sigma_{ij}^2/N$ is averaged to compute the approximate variance.

The variances of axis aligned EBFs are computed using both data values and gradients since there is more than one unknown variance in the axis aligned EBF. By taking derivatives of Equation (6.4), the approximation of the $x$ direction variance works as follows:

$$\sigma_{x_{ij}}^2 = -(x_j - \mu_{xi}) \cdot \frac{\mathbf{v}_j}{d\mathbf{v}_j/dx_j}\ ,\ j = 1, ..., N \quad . \tag{6.7}$$

The $y$ and $z$ direction variances can be computed similarly and the average of these variances is then used for the approximate variances.

For the arbitrary directional EBFs, the above approximated variances can be used by setting the off-diagonal components to zero, which means that all rotation angles are zero, for the initial parameter values for the nonlinear optimization.

***Nonlinear Optimization*** Once all approximate parameters are computed, they are inserted into the nonlinear optimization algorithm, i.e., the Levenberg Marquardt approach [94] is applied to minimize sum squared error. This approximation algorithm optimizes all parameters at the same time. Note that parameters of the axis aligned EBFs can be optimized in the same way as RBFs are optimized, since off-diagonal components in the covariance matrix are all zero. However, for arbitrary directional EBFs, the scaling components in the scaling matrix and the angles in the rotation matrix are optimized instead of $a_1, \ldots, a_6$ for 3D datasets, since the result of this optimization might not construct a real covariance.

***Error Measurement*** Even though EBFs are good basis functions for any structural dataset after optimization, it can still show visible artifacts. Therefore, two different cost functions are used in the optimization process, which are compared here. Commonly applied in such an optimization process is the L2-norm based cost function

$$\psi = \frac{1}{2} \sum_{j=1}^{N} \left(f(\mathbf{x_j}) - \mathbf{y}_j\right)^2 \quad , \tag{6.8}$$

which only uses the data values. For increased visual accuracy, the gradients are pre-approximated at each data point and added to the cost function, using the H1-norm based cost function [50], given as

$$\begin{aligned}\psi =& \frac{1}{2} \sum_{j=1}^{N} \Big\{ \left(f(\mathbf{x_j}) - \mathbf{y}_j\right)^2 + \left(\frac{\partial f(\mathbf{x_j})}{\partial x} - \frac{\partial \mathbf{y}_j}{\partial x}\right)^2 + \\ & \left(\frac{\partial f(\mathbf{x_j})}{\partial y} - \frac{\partial \mathbf{y}_j}{\partial y}\right)^2 + \left(\frac{\partial f(\mathbf{x_j})}{\partial z} - \frac{\partial \mathbf{y}_j}{\partial z}\right)^2 \Big\} \quad ,\end{aligned} \tag{6.9}$$

in order to remove the visual artifacts. Comparison results between these two cost functions are shown in Figure 6.13 of Section 6.3.

### 6.1.3 Scalar vs. Vector Encoding

Scalar data approximation is performed using the above approach since there is only one value at each data point. Therefore, the maximum data point is set to be a center of an ellipsoidal Gaussian and its value at the point is set to be a weight of the ellipsoidal Gaussian. Variance terms are approximated using either Equation (6.6) or (6.7) according to the basis functions. The optimization of all these parameters is performed by using the Levenberg Marquardt algorithm. For the 3D spherical Gaussian, 5 parameters (center, weight, and variance) are optimized at the same time, 7 parameters for the axis aligned ellipsoidal Gaussians, and 10 parameters for the arbitrary directional ellipsoidal Gaussians.

While scalar data approximation is rather simple, vector data needs further treatment in order to optimize all vector components at the same time. There are many approaches to encode vector datasets. A simple approach encodes each component of the vector separately, producing 3 separate systems of basis functions (one for each component). A drawback of this representation is not only the requirement of a much larger number of basis functions, but also the fact that the reconstructed vectors are less accurate since the errors of the individual components may add up to larger errors. Therefore, the chosen approach encodes the vector data as one three-valued quantity, computing the error of the encoding based on the vector error. For more accurate encoding, separate weights and variances for each vector component are calculated, but not separate center locations. Experiments have been conducted for encoding single weight and variance for each basis function, but the resulting encoding errors were much larger or the representation needed many more basis functions. Based on this experience, the memory layout for each basis function was chosen to store one center, three weights, and three variances as described in Section 6.2.1. In 3D, there are 9 parameters for the spherical Gaussians, 15 parameters for the axis aligned ellipsoidal Gaussians, and 24 parameters for the arbitrary directional ellipsoidal Gaussians.

## 6.2 Interactive Rendering

All rendering techniques that have been implemented to evaluate the strength of evaluating the encoded data are based on the capability of modern graphics adapters to perform arbitrary arithmetical operations on the GPU. The basic principle of the rendering system is shown in Figure 6.4. The high memory bandwidth and parallel processing capability of modern graphics hardware are exploited by downloading all necessary information, like the coefficients of the RBF and EBF encoding as texture maps to the graphics card. This allows the fragment unit to access all data required for reconstruction and to perform the decoding on-the-fly during rasterization.

Since the basis functions are evaluated by the GPU for each rendered fragment, the encoding of the data is hidden from the rendering and, therefore, the approach allows for a variety of visualization algorithms. For the visualization of encoded scalar fields the system supports direct volume rendering, volume-rendered isosurfaces and arbitrarily oriented cutting planes described in Section 6.2.2. Further

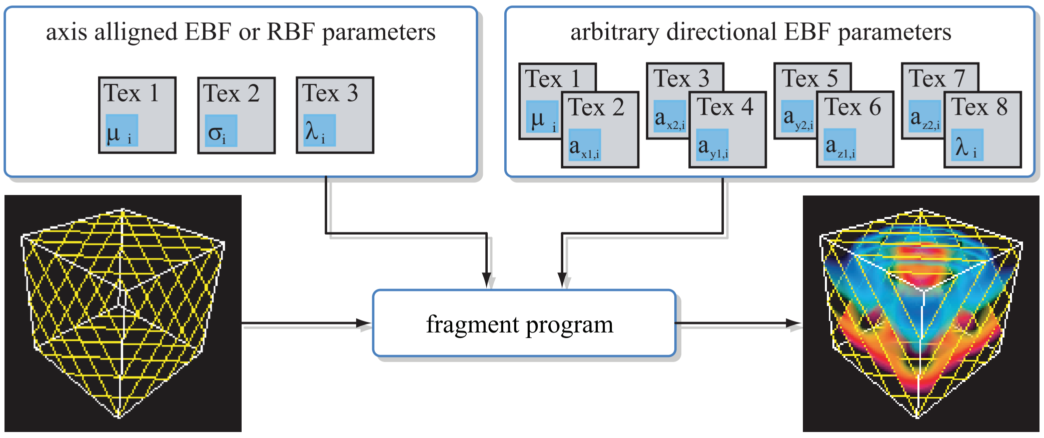


Figure 6.4: Texture layout and parameter transfer to the rendering system depends on the chosen RBF/EBF encoding. The coeffcients are accessed directly from multiple two-dimensional texture maps storing the compact representation in the local memory of the graphics card. (Images courtesy of Jang *et al.* [63], ©The Eurographics Association 2006).

possibilities include the mapping of the reconstructed data onto the surface of related geometry, e.g., color coded pressure on the body of an airplane. Reconstruction and visualization of vector data, however, requires not only more sophisticated data handling, but even more importantly, vector field specific visualization techniques as demonstrated in Sections 6.2.3 and 6.2.4. Performance results for all types of basis functions and datasets are given in Table 6.2 of Section 6.3.

### 6.2.1 Data Structure and Texture Layout

To speed up rendering, a volume subdivision algorithm [64] is used. This algorithm utilizes an adaptive octree, it obtains the distribution of basis functions using influence ranges and divides a volume into smaller subvolumes by setting a maximum number of basis functions rendered in a subvolume. Therefore, each subvolume contains less than the maximum number of basis functions, increasing rendering performance.

For RBFs, influence ranges are the same in all directions, but EBFs have different influence ranges in different directions. Figure 6.2 shows the influence ranges of RBFs and EBFs. The influence radii of RBFs are computed as

$$r_i = \sigma_i \cdot \sqrt{2 \cdot ln\left(|\lambda_i|/\epsilon\right)} \quad ,$$

where $\epsilon$ is a user defined error tolerance. Subdivided volumes are cubes. For the axis aligned EBFs, the influence range in $x$ direction can be computed similarly as

$$r_{x_i} = \sigma_{x_i} \cdot \sqrt{2 \cdot ln\,(|\lambda_i|/\epsilon)} \quad .$$

The influence ranges in $y$ and $z$ direction can be computed similarly to the above. Note that the influence range is a cuboid. The influence range of the arbitrary directional EBF in 3D can be computed in each direction as follows:

$$\begin{aligned} r_{x_i}^2 &= \frac{a_{2_i}a_{3_i} - a_{5_i}^2}{|\mathbf{V}^{-1}|} \cdot 2 \cdot ln\left(\frac{|\lambda_i|}{\epsilon}\right) \quad , \\ r_{y_i}^2 &= \frac{a_{3_i}a_{1i} - a_{6_i}^2}{|\mathbf{V}^{-1}|} \cdot 2 \cdot ln\left(\frac{|\lambda_i|}{\epsilon}\right) \quad , \\ r_{z_i}^2 &= \frac{a_{1_i}a_{2_i} - a_{4_i}^2}{|\mathbf{V}^{-1}|} \cdot 2 \cdot ln\left(\frac{|\lambda_i|}{\epsilon}\right) \quad . \end{aligned}$$

Note, this influence range is also a cuboid, but not just a parallelepiped.

The organization of the coefficients into textures has to allow efficient access to all contributing basis functions of a particular cell. With the GeForce6's dynamic branching and long fragment programs, a simple but effective greedy layout algorithm can be used. This algorithm processes all cells sorted by descending number of basis functions, starting with the cell containing the most functions and assign it to the texture line with the smallest but sufficient number of free slots. For fast computation of this slot, a free space list is utilized with one entry for each texture row. This list is sorted by ascending free slots after the placement of each cell. Texels in the texture map are filled in left-to-right order. Using this simple approach, a minimal overhead of unused texels can be achieved. The fragment program only needs to know the index of the first basis function for the current cell and can access the remaining coefficients by applying an increasing offset to the $x$-texture coordinate. Note that this may result in some data duplication, since the spatial decomposition generates several instances of the same basis function in different cells. However, as the size of a single set of coefficients is small, this overhead is acceptable.

Vector data is encoded at full precision into three floating point texture maps as indicated in Figure 6.4. All maps share the same basic layout to enable the use of the same index to address all components of the RBF parameter set. The first texture is an RGB map holding the positions $\mu_i$ of the RBF centers. A second and third map store the weights $\lambda_i$ and the widths $\sigma_i$ of the RBF functions respectively. Note that actually $(2\sigma_i^2)^{-1}$ is stored instead of the width, in order to reduce the number of required fragment operations. For efficiency, the texture format is adapted to the dimensionality of the input data using either an R, RG, RGB, or RGBA texture map. Thus, the RBF reconstruction algorithm supports vector data up to four dimensions, multi-field datasets with up to four different scalar values, or any combination with up to four data components. For vector data that has been encoded with arbitrary directional EBFs, a set of eight textures is needed to store all parameters, as illustrated in Figure 6.4. In particular, six RGB textures are needed for all covariance matrix elements $a_{1,i}, ..., a_{6,i}$. One RGB texture is used to store all centers $\mu_i$ and one RGB texture is used for the weights $\lambda_i$.

In the rendering stage, back-to-front blending is applied. Due to that it is not possible to render all cells consecutively, since this would lead to blending artifacts, because the cells are not stored as a depth sorted list. Even if they were, it would still lead to artifacts on cell boundaries. A better way is to render slice by slice. Since the order and thus the exact depth of all slices is known, only the coordinates of the intersection polygon need to be computed for each slice. For each intersection polygon, the fragment shader is called with a pointer to the first basis function in the texture and the number of functions to evaluate. Although the hierarchical structure reduces the number of evaluated functions, to improve rendering speed, it can also slow down the rendering if the hierarchical subdivision level is chosen too high. This effect is a result of the small call overhead that is needed for the render call of every intersection polygon. For too many intersection polygons, this exceeds the gain of reducing the amount of evaluated basis functions.

### 6.2.2 Slicing Planes and Volume Visualization

The first implemented class of visualization methods for multi-field data is centered on the evaluation of the original encoded data properties. Here the fragment programs can take advantage of the fact that almost all fragment instructions work on a four-component vector; thus, the number of fragment operations required for a multi-field reconstruction is essentially the same as for a single data component. A few additional instructions are introduced, since for a multi-field RBF encoding, three texture lookup operations are necessary in order to determine the coefficients for one basis function. The five coefficients for an RBF encoded scalar can always be stored in two RGBA texels.

The fragment programs can be applied to render a single slicing plane through the volume domain. The system supports an arbitrary slice plane that can be freely moved through the volume to explore flow properties. To reconstruct a scalar value from data that has been encoded e.g., by arbitrary directional basis functions, the fragment program has to calculate Equation (6.3) for all basis functions that influence the fragment. The core loop of a fragment program to reconstruct a scalar value form an arbitrary directional EBF representation is shown in Listing 6.1. Exploiting the SIMD architecture of graphics hardware, it is possible to evaluate Equation (6.3) with only eight arithmetic instructions. First, the center position is fetched into registers by a texture lookup, before the distance to the actual texel is computed. Then, the vector-matrix-vector multiplication is com-

```
1  REP numfunc;                                         // dynamic loop over all EBFs
2    TEX cDist,    txPos.xyxx, texture[1], RECT;   // fetch center position
3    SUB cDist,    fragment.texcoord[0],   cDist;  // compute distance
4
5    TEX var_X1,   txPos.xyxx, texture[2], RECT;   // fetch elem 1 - 3 and
6    TEX var_X2,   txPos.xyxx, texture[3], RECT;   // elem 4 - 6 of covar matrix
7    TEX wt.xyz,   txPos.xyxx, texture[4], RECT;   // fetch the weights
8
9    MUL tmp,      cDist,      cDist;              // start computing exponent
10   DP3 expVal.x, tmp,        var_X1;
11
12   MUL tmp,      cDist,      cDist.yzxx;         // all var_X2 terms are
13   DP3 tmp.x,    tmp,        var_X2;             // pre-multiplied by two
14   ADD expVal,   expVal,     tmp;                // now compute exponent
15   EX2 expVal.x, -expVal.x;                      // to the base of two
16
17   MAD val,      wt,         expVal,     val;    // reconstruct vector
18   ADD txPos,    txPos,      txInc;              // increment tex coords
19 ENDREP;
20
21 //projection, color mapping, ...
```

Listing 6.1: Main part of a fragment program for scalar decoding of arbitrary directional basis functions via covariance matrix. The program exploits dynamic loops. (Code courtesy of Jang *et al.* [63], ©The Eurographics Association 2006).

puted with two dot products per scalar component, multiplied with the squared distance and followed by the evaluation of the exponent (see Listing 6.1, lines 9–15). Based on the reconstructed scalar, further feature extraction or color mapping can be done.

The color mapping for the evaluated basis function on a slice is performed through a lookup in a transfer function implemented as a one-dimensional texture map. The lookup is based either on a single component of the multi-field dataset or on the magnitude of the encoded vector. Interactive switching between these behaviors can be achieved by utilizing a dynamically assigned component mask, implemented as a parameter of the reconstruction fragment program. Alternatively, a three-dimensional texture can be used as transfer function, allowing for a more sophisticated mapping based on all components of the encoded vector or multi-field dataset. During the rendering of the slice, the algorithm has to take care of the domain decomposition, since a different list of RBF centers has to be considered for each cell. Therefore, the slice is clipped at the cell boundaries and the resulting slice portions are rendered separately with indices pointing to the RBF coefficients of the corresponding cell.

Increasing the number of slicing planes and blending them back-to-front leads to direct volume renderings of the datasets, as explained in Section 2.5. The technique can also be extended to render shaded isosurfaces as demonstrated in [168]. In the latter case, the analytically reconstructed gradient is used for the lighting computation. An example for RBF based volume rendering is illustrated in Fig-

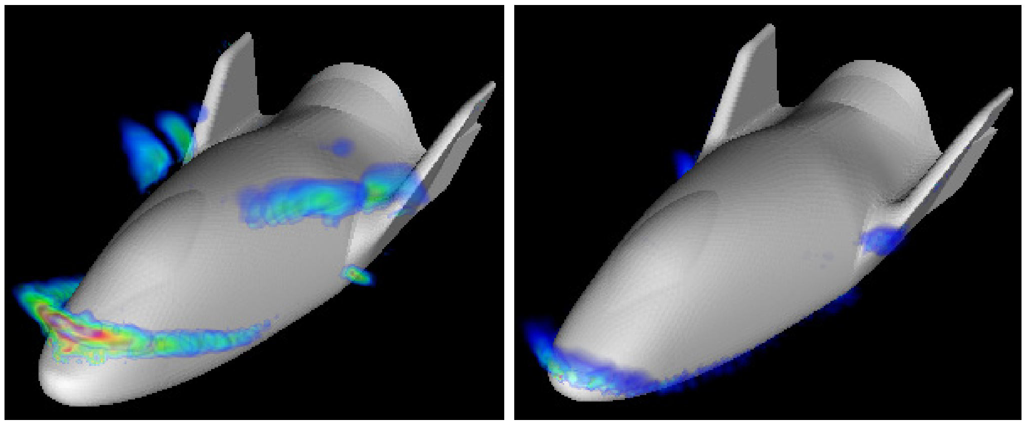


Figure 6.5: A volume rendering of the X38 compression shock (left) and expansion shock (right) using 4,883 and 6,789 RBFs, respectively. (Images courtesy of Weiler *et al.* [161], ©2005 IEEE).

ure 6.5, on the scalar compression shock of the X38 return shuttle.

Two slices of features detected on RBF encoded datasets are shown in Figure 6.6. The left image shows vorticity magnitude of the MHD dataset. The right image shows a slice of $\lambda_2$ values extracted from the tornado dataset. Computation time for these slices is about 0.04 s and almost independent of the number of basis functions reconstructed per rendering pass. Single slices are only useful to the knowledgeable fluid dynamics engineer. Volume renderings based on a stack of slices reveal more structure. Figure 6.7 (a) illustrates this with an example of the volume rendered MHD dataset with extracted vorticity magnitude. The dataset is rendered with 400 slices. The right image of Figure 6.7 shows $\lambda_2$ values computed for the tornado dataset and visualized with 256 slices. Since feature detection involves expensive gradient calculations, volume rendering of dynamically extracted features provides only limited interactivity. Rendering the tornado dataset with 32 slices and $\lambda_2$ extraction reaches an average of 1.2 fps. Nevertheless, GPU-based feature detection in radial basis space is a very promising technique that should approach interactive rates with the next generation of graphics hardware.

### 6.2.3 Feature Extraction

All the features relevant for this work are described in Section 3.3 and are based on the velocity gradient tensor $J$ of the vector field, given in Equation (3.4). Therefore, if features of the encoded vector datasets are to be extracted, a fragment program is needed that is capable of analytically calculating the nine-component Jacobian matrix. The GeForce6 chip series not only allows the reconstruction of a dynamic number of basis functions, but also the computation of all nine elements

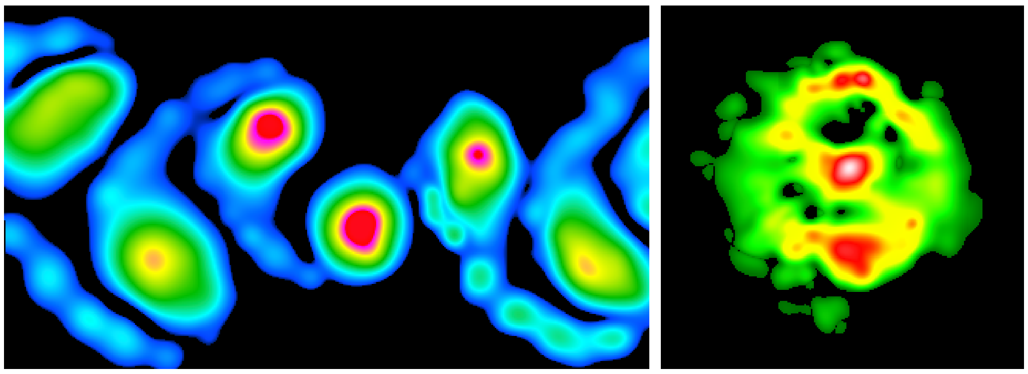


Figure 6.6: Slices of features extracted directly from the RBF encoding. Image (a) shows vorticity magnitude of the MHD dataset encoded with 2,145 RBFs. In image (b) the $\lambda_2$ values have been extracted from the tornado dataset. Note that the same transfer functions as in Figure 6.7 have been applied, in order to allow direct comparison with the corresponding volume renderings. (Images courtesy of Weiler *et al.* [161], ©2005 IEEE).

of the Jacobian matrix and the final evaluation in one single fragment program. Therefore, only one single rendering pass is required resulting in a very fast performance. Sample code that shows the reconstruction of a basis function with gradient computation in a dynamic loop is given in Listing 6.2.

Based on the reconstructed velocity gradient tensor, the system supports feature calculations for vorticity, helicity and $\lambda_2$ vortex detection, to demonstrate the flexibility and use of this approach for feature detection. To gain an understanding of the local flow, a good first approach is to calculate the vorticity of a vector (cf. Equation (3.5)). Adding the required computation to the fragment program presented in Listing 6.2 is straight forward. Since vorticity is a vector quantity, the implementation allows the user to interactively define a bitmask for masking out single components of $\omega$ and visualizing the magnitude of the resulting vector.

Vorticity already gives a good impression of where vortices can be found in the vector field. However, advanced vortex detection algorithms, like the $\lambda_2$ criterion (cf. Equation (3.7)), give even better results and are also suitable for GPU-based implementations due to their local nature. As before, a fragment program like the one sketched in Listing 6.2 is used to retrieve the partial derivatives. After this reconstruction the Jacobian is decomposed into a symmetric and an asymmetric part. To determine the eigenvalues of the matrix $S^2 + \Omega^2$, and to find the relevant eigenvalue $\lambda_2$, the approach proposed in [143] is adopted. The basic idea is to use a modified version of *Cardan's Solution* to analytically determine the root of the characteristic polynomial. By pre-computing coefficients and storing them into texture maps, no trigonometric functions need to be evaluated. As a result, the

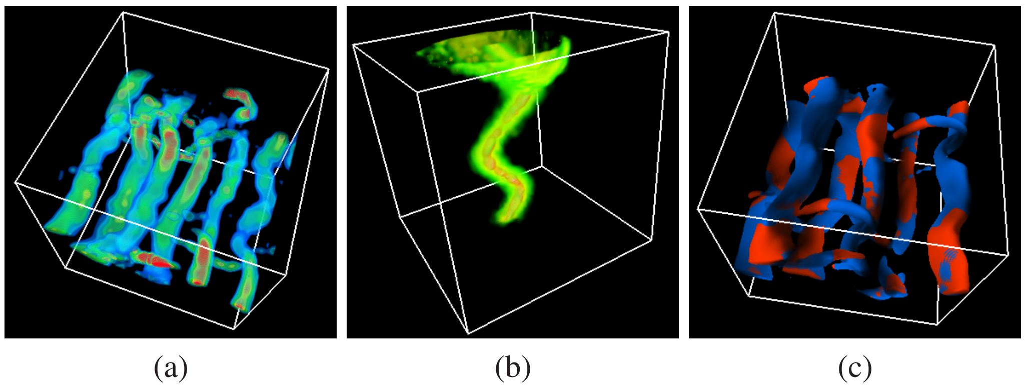


Figure 6.7: Volume rendered features extracted from the RBF encoded vortex and tornado datasets. Image (a) shows velocity magnitude of the plasma flow (MHD), image (b) illustrates the computed $\lambda_2$ values of the synthetic tornado. (Images courtesy of Weiler *et al.* [161], ©2005 IEEE).

computation is very efficient despite its higher complexity.

By computing and visualizing the helicity of a velocity field, the analyst can examine the potential for helical flow, or flow that appears to move in a corkscrew pattern. Helicity is computed using Equation (3.6), and physically represents the curl in the direction of the velocity field. If the fluid moves in a dominant streamwise direction, then helicity looks similar to vorticity. However, if the flow is not dominated by a single direction, then the helicity will provide interesting and different results than those obtained by computing and analyzing either curl or vorticity.

Examples for feature based rendering are given for slice planes in Figure 6.6 and for volume rendering in Figure 6.7. The two slices of features detected on RBF encoded datasets show vorticity magnitude of the MHD dataset in Figure 6.6 (a) and $\lambda_2$ values extracted from the tornado dataset in Figure 6.6 (b). Figure 6.7 (a) illustrates this with an example of the volume rendered MHD dataset with extracted vorticity magnitude. Image (b) of Figure 6.7 shows the tornado dataset visualized with 256 slices. Figure 6.7 (c) shows isosurface rendering of vorticity magnitude, with positive helicity mapped to red colors and negative helicity to blue colors.

### 6.2.4 Particle Advection

Particle tracking is another well-known technique for understanding flows, as presented in Section 2.4.1. The encoded vector fields are particularly well-suited for this technique since the vector field has to be reconstructed only at a small number of positions. Although the positions may be distributed across the 3D volume, the compact representation with basis functions can be used to accomplish this

```
REP numfunc;                                          // dynamic loop over all RBFs
  TEX cDist.xyz, txPos.xyxx, texture[0], RECT; // fetch RBF's center position
  TEX vari.xyz,  txPos.xyxx, texture[1], RECT; // fetch RBF's variances
  TEX lmbda.xyz, txPos.xyxx, texture[2], RECT; // fetch RBF's weights

  SUB cDist.xyz,  fragment.texcoord[0], cDist; // compute distance
  DP3 expval.xyz, cDist.xyzx,  cDist.xyzx;      // expval = ((-|x - mu_i|^2)
  MUL expval.xyz, expval.xyzx, -variances.xyzx;// / (2*sigma_i^2))

  EX2 expres.x,   expVal.x;                     // compute exponents
  EX2 expres.y,   expVal.y;                     // to the base of two
  EX2 expres.z,   expVal.z;

  MUL fctrs,      lmbda,    vari;               // factor = ((x-mu_i)*lambda_i
  MUL fctrX,      cDist,    fctrs.x;            // / (2*sigma_i^2))
  MUL fctrY,      cDist,    fctrs.y;            // see correction below
  MUL fctrZ,      cDist,    fctrs.z;

  MAD derivX,     fctrX,    expRes.x,  derivX; // compute partial derivative
  MAD derivY,     fctrY,    expRes.y,  derivY; // for every direction
  MAD derivZ,     fctrZ,    expRes.z,  derivZ;

  MAD val,        expRes,   lmbda,     val;    // reconstructed the vector
  ADD txPos,      txPos,    txInc;             // increment tex coords
ENDREP;

// correction term since variances store (1 / (2*sigma_i^2))
MUL derivX, derivX, {2}.x;
MUL derivY, derivY, {2}.x;
MUL derivZ, derivZ, {2}.x;

// feature calculation and shading
```

Listing 6.2: Fragment program for the combined calculation of the vector field and the velocity gradient tensor. The program exploits the possibility of dynamic loops. (Images courtesy of Weiler *et al.* [161], ©2005 IEEE).

task. The visualization system supports a particle advection routine that is capable of tracking a large number of particles simultaneously, exploiting the parallel rendering pipelines of current graphics cards.

The initial positions of the particles have to be defined by the user as a set of 3D coordinates. The particle coordinates are then stored in a 2D floating point texture with as many texels as are required for storing the positions. In the next step, a quadrilateral of the size of the texture is rendered to a offscreen buffer. For each generated fragment, the velocity vector is then reconstructed as described in Listing 6.2, using the appropriate particle coordinate that is stored in the texture, for evaluating the sum of basis functions. The particle position is updated using an Euler integration step as given in Equation (2.5), based on the reconstructed velocity. These steps are repeated using the new particle positions as the input texture until a user-defined number of iterations has been reached.

After each step, the updated particle positions are stored in a 2D floating point

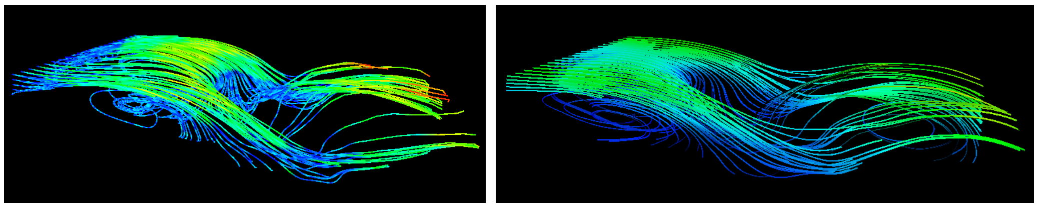


Figure 6.8: Traces of 110 particles tracked over 400 time steps in the Channel dataset. Image (a): RBF encoded dataset and GPU-based computation. Image (b): Original Cartesian grid and software implementation. (Images courtesy of Weiler *et al.* [161], ©2005 IEEE).

texture. Ideally, this texture should be used directly for rendering particle traces as OpenGL vertex arrays without the need to read back the particle positions from the graphics card. However, at the time this algorithms was developed, this functionality was only available for ATI cards by means of the so-called *uberbuffer* extension. Thus, rendering the particle traces on the GeForce6, requires a costly `glReadPixels` for each time step to read the particle positions to main memory and a transfer back to the GPU for the rendering.

An example of GPU based particle tracing is illustrated in Figure 6.8 (a). The image shows 110 particles traced over 400 time steps, the particle traces were calculated on the GPU for the RBF encoded dataset, using Euler integration with fixed step size. Image (b) shows the traces computed by the commercial flow visualization package *PowerVIZ* [39] using the original Cartesian grid and fourth order Runge-Kutta integration with adaptive stepsize. As expected, the Euler integration employed for the GPU implementation produces less accurate but nevertheless comparable results.

### 6.2.5 Texture-based Flow Visualization

Particle traces serve good for the visualization of local flow behavior, represented by the single traces. For a global visualization of flow fields, texture advection is a well suited and common technique that is capable of representing the underlying flow with a low density up to a high density of particles. The transport is based on a semi-Lagrangian scheme, which is explained in Section 3.4.1.

As described before, the particles that are injected into the flow are represented on a regular grid, namely a texture, which is denoted as the property field $\varrho(\mathbf{x})$. From the Eulerian point of view, the position of a particle is implicitly given by the location of the corresponding texel in the property field. Particles are transported along streamlines for steady, or along pathlines for unsteady vector fields. Bilinear interpolation is applied to reconstruct the property field at arbitrary locations.

In the visualization framework an arbitrary slice plane is implemented, on

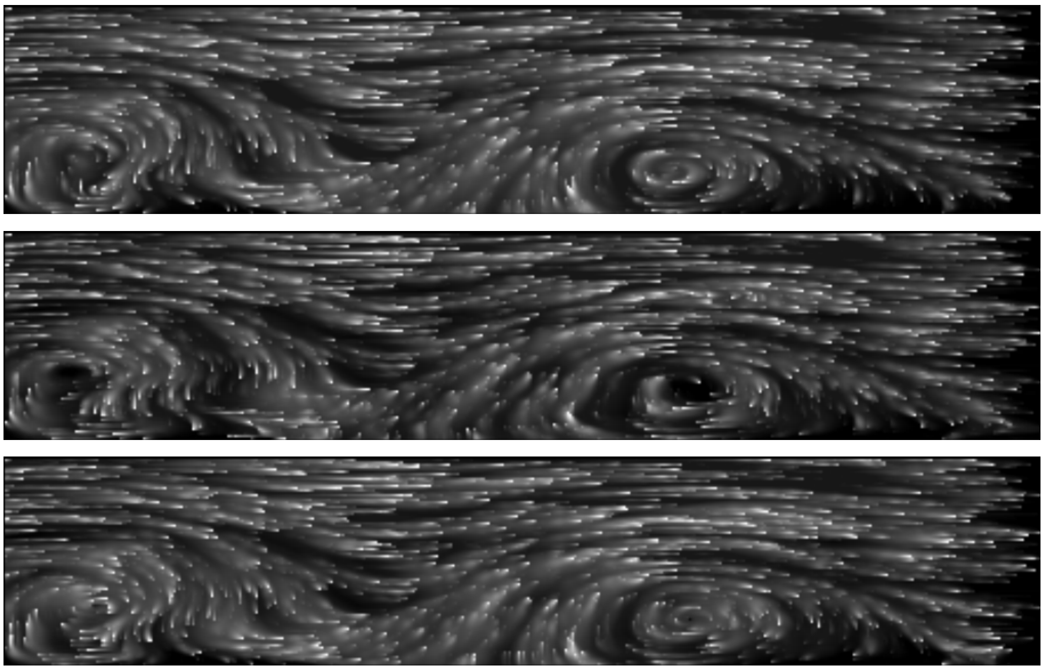


Figure 6.9: Comparison of water channel data with RBF (top), axis aligned EBF (middle), and aribtrary directional EBF (bottom). (Images courtesy of Jang *et al.* [63], ©The Eurographics Association 2006).

which the texture advection can be computed. The plane can be freely moved through the volume to explore flow properties, as illustrated in Figures 6.9 and 6.10. To reconstruct a scalar value from data that has been encoded by arbitrary directional basis functions, a fragment program is needed that is capable of analytically calculating Equation (6.3) for all basis functions that influence the fragment. Exploiting the SIMD architecture of graphics hardware, it is possible to evaluate Equation (6.3) with only eight arithmetic instructions, as shown in the body of the loop in Listing 6.2. The vector reconstruction needed for particle advection requires fourteen arithmetic instructions, which is basically the same procedure as scalar decoding, but applied for every vector component. The particle advection can be computed by first order Euler integration due to Equation (2.5). To evaluate the property field $\varrho(\mathbf{x})$ at time step $t - \Delta t$, two textures are used as render targets, as detailed in Section 3.4.1, and ping-pong rendering is applied.

Texture based flow visualization was applied for all three encoding techniques, to one slice of the water channel dataset, shown in Figure 6.9. Although all three different techniques show very similar results in this case, the axis aligned encoding needs fewer basis functions and thus gives superior real-time rendering results. Further, the advection technique in combination with the arbitrary slice

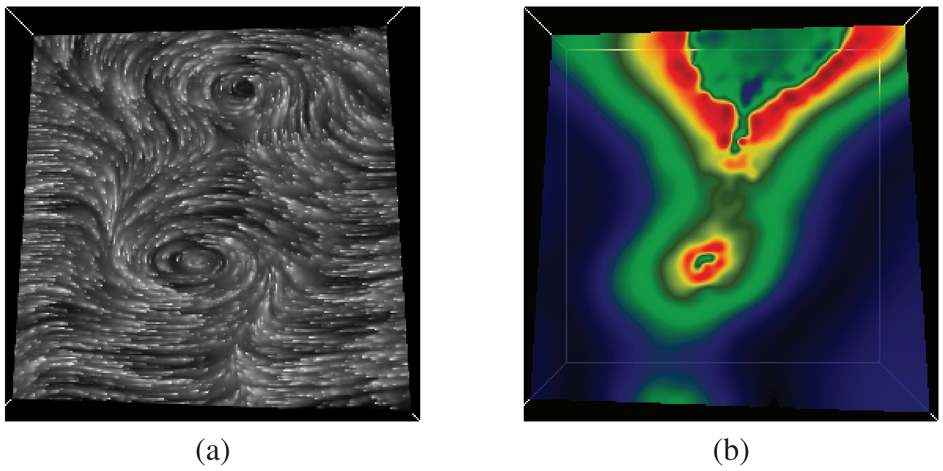


Figure 6.10: EBF encoded tornado dataset with texture based flow visualization (a), and vorticity magnitude (b). The arbitrary sliceplane cuts the noosed vortex core two times. (Images courtesy of Jang *et al.* [63], ©The Eurographics Association 2006).

plane enables the user to interactively explore flow features as shown for the tornado dataset in Figure 6.10.

## 6.3 Application Cases

The proposed techniques for rendering the encoded multi-field data have been implemented using C++, OpenGL and Assembler code for the fragment programs. The rendering framework supports both, MS Windows and Linux systems and has been tested on a Pentium 4 3.4GHz processor with an NVIDIA GeForce 7800 GTX graphics board for a variety of datasets. If not stated different, all timings presented in Table 6.2 have been conducted on this system with a viewport of $512^2$ and 128 slices for volume rendering.

Under the multiplicity of used datasets reside different application cases such as the simulation of fluid flow, convection scenarios and shock wakes or real-world measurements of density and fluid velocity of participating media. The specification and origin of the datasets should be described first, whereby all data has been encoded with the schemes proposed in Section 6.1.2 for evaluation.

The oil reservoir data used in Figure 6.1 was computed by the Center for Subsurface Modeling at The University of Texas at Austin. This 156,642 tetrahedra dataset is a simulation of a black-oil reservoir model used to predict placement of water injection wells to maximize oil from production wells.

The X38 dataset shown in Figure 6.5 is based on a tetrahedral finite element viscous calculation computed on geometry configured to emulate the X38 Crew Return Vehicle. The geometry and the simulation were computed at the Engineering Research Center at Mississippi State University by the Simulation and Design

| Dataset | ML | Convection | Bluntfin | Oil Reservoir | Water Channel |
|---|---|---|---|---|---|
| I | 2,092 | 237 | 891 | 59 | 895 |
| II | 208 | 101 | 264 | 13 | 293 |
| III | 112 | 90 | 282 | 13 | 299 |

Table 6.1: Statistical functional approximation results. Each number indicates the number of basis functions for RBFs (I); axis aligned EBFs (II); and arbitrary directional EBFs (III). (Courtesy of Jang *et al.* [63], ©The Eurographics Association 2006).

Center. This dataset represents a single time step in the reentry process into the atmosphere. The simulation was computed on an unstructured grid containing 1,943,483 tetrahedra at a 30 degree angle of attack.

The Magnetohydrodynamics dataset (MHD) shown in Figure 6.7 (a,c) is a simulation of plasma flow in the outer heliosphere of the sun computed by D. Aaron Roberts at NASA Goddard Space Flight Center. This dataset has been encoded with 2,145 RBFs. The norm of the curl of the velocity field is used as a measure of vorticity, showing the alternating vortices in the plasma flow.

The water channel measurement used for the particle tracing in Figure 6.8 and the texture advection in Figure 6.9, is a time-dependent dataset obtained in an experiment studying laminar-turbulent boundary layer transitions in a water channel.

The natural convection dataset used for the visualization in Figure 6.11, simulates a non-Newtonian fluid in a box, heated from below, cooled from above, with a fixed linear temperature profile imposed on the side walls. The simulation was developed by the Computational Fluid Dynamics Laboratory at The University of Texas at Austin and was run for 6000 time steps on a mesh consisting of 48000 tetrahedral elements.

The Marschner-Lobb data shown in Figure 6.12 was obtained using an equation developed by Marschner and Lobb [98], and the same parameters that they used have been applied in this simulation, except for spatial range. For encoding the dataset, it was sampled with 50,000 points randomly in $-0.5 < x, y, z < 0.5$.

The Bluntfin dataset illustrated in Figure 6.13 for comparison of the different encodings, was developed by C.M. Hung and P.G. Buning and it is a 40x32x32 single-zone, curvilinear, structured block dataset in plot3D format.

The last dataset used for the evaluation is the synthetic tornado dataset shown in Figures 6.6, 6.7 and 6.10. The 32x32x32 dataset was provided by Roger Crawfis of The Ohio State University.

To compare the encoding results according to the different basis functions a compilation of the encoding results is given in Table 6.1. As shown in the ta-

| Dataset | ML | Convection | Bluntfin | Oil Reservoir | Water Channel |
|---|---|---|---|---|---|
| I | 0.1 | 0.7 | 1.3 | 2.7 | 44.8 |
| II | 0.7 | 1.3 | 1.8 | 10.6 | 128.4 |
| III | 0.9 | 1.1 | 1.6 | 8.0 | 87.2 |

Table 6.2: Performance measurements for different scenarios in frames per second (fps). All volumes have been rendered with 128 slices and a viewport of $512^2$. The datasets are tested for the three different encoding techniques, RBFs (I); axis aligned EBFs (II); and arbitrary directional EBFs (III). For the Water channel dataset, the timings have been evaluated for the texture advection technique on one sliceplane. (Courtesy of Jang *et al.* [63], ©The Eurographics Association 2006).

ble, EBFs generate better statistical results than RBFs. Since all datasets have non-spherically shaped volumes, EBFs are more flexible and appropriate to approximate the given volumes. For the comparison between axis aligned EBFs and arbitrary directional EBFs, the results depend on the datasets. If one dataset has more diagonal shapes than another, the arbitrary directional EBF is a more appropriate basis function. Otherwise, the axis-aligned EBF is more appropriate as approximation using arbitrary directional EBFs requires more computation than approximation using axis-aligned EBFs and there are more parameters required for the rendering system.

To give a fair visual comparison of the encoding techniques, the data has been encoded by all methods, with approximately the same cost, according to the error criteria. Therefore, an impression of how the number of basis functions needed by each technique impacts the quality of the reconstructed structures. Figure 6.11 shows a comparison using two time steps of the natural convection data. Since it has mostly axis aligned symmetric shapes, axis aligned EBFs and arbitrary directional EBFs show similar statistical and visual results. However, the result using RBFs shows artifacts in the high gradient area in the upper part of Figure 6.11 (a) and (e). We highlight areas with white boxes to more easily compare our results visually. For the rendering, we use the flow illustration technique proposed by Svakhine et al. [148].

Figure 6.12 shows the reconstructions of the Marschner Lobb dataset from six diverse encodings. The left column shows three rendering results using the L2-norm based cost function. Since the Marschner Lobb data has very high frequency data values, the RBF result shows the worst approximation and EBFs show better approximation. The lower row shows the rendering results using the H1-norm based cost function. The H1-norm based cost function gives more accurate results compared to the L2-norm based cost function.

Figure 6.13 compares six rendering results for the bluntfin dataset. Similar to

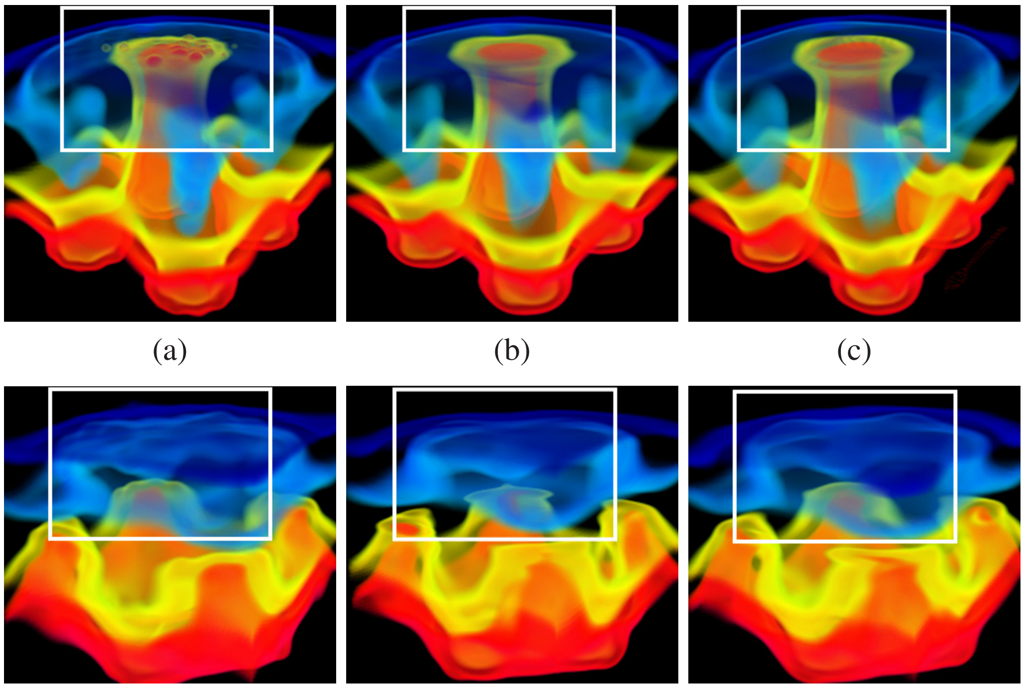


Figure 6.11: Comparison of two time steps (70th : a,b,c, 150th : d,e,f) of the natural convection datasets encoded with: RBFs (a) and (d); axis aligned EBFs (b) and (e); and aribtray directional EBFs (c) and (f). Focus areas are highlighted with white boxes to compare the quality of THE encoding results visually. (Images courtesy of Jang *et al.* [63], ©The Eurographics Association 2006).

the Marschner Lobb result, EBFs shows better results than RBFs and the H1-norm based cost function generates more visually accurate results than the L2-norm based cost function. However, even if the rendered images look like diagonal shapes of data distribution, the data distribution is mainly along the $z$ direction. Therefore, the number of axis aligned EBFs needed for the encoding is similar to the number of arbitrary directional EBFs. Additionally, the rendering results of both EBFs show similar visual results. As shown in Figure 6.11 and 6.13, the arbitrary directional EBF encoding provides the best reconstruction with fewest basis functions. Even though this fragment shader needs the most instructions to evaluate Equation 6.3, the minimum number of basis functions leads to the best performance.

Concluding it can be said that the use of multi-field data encoding with radial and ellipsoidal basis functions for large datasets can overcome the approximation and visualization problems common for non-spherical structures, as well as the bottleneck of limited memory on graphics hardware. The application cases

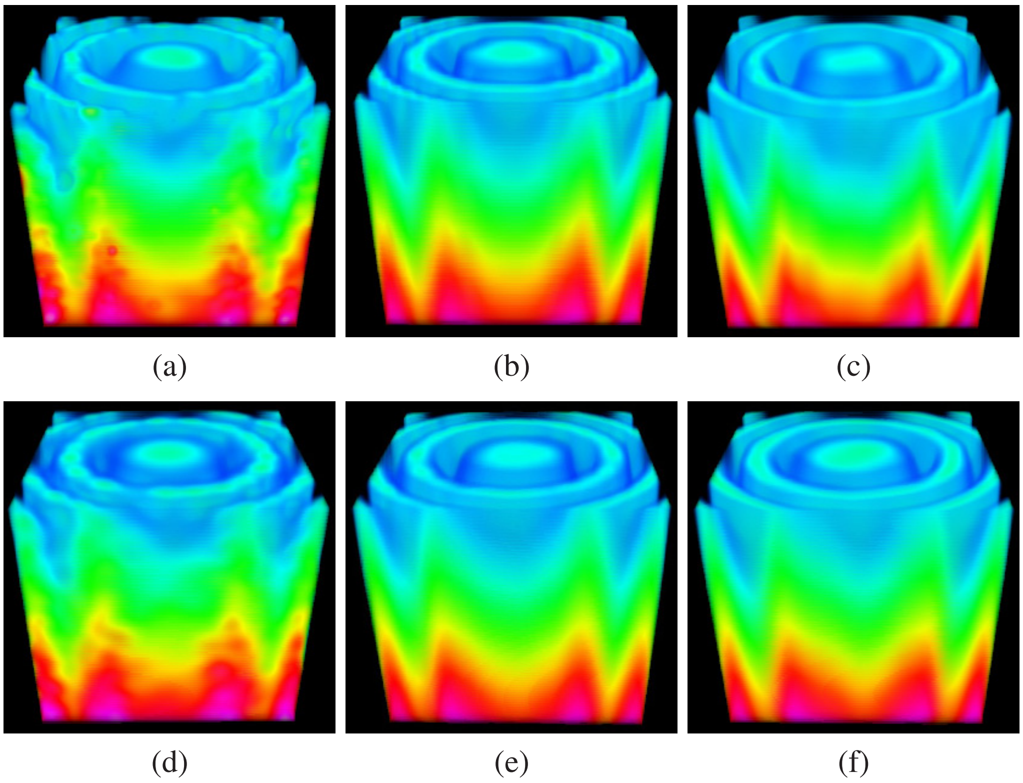


Figure 6.12: Comparison of the Marschner-Lobb dataset encoded with: RBFs (a); axis aligned EBFs (b); arbitrary directional EBFs (c); RBFs with gradient (d); axis aligned EBFs with gradient (e); and arbitrary directional EBFs with gradient (f). (Images courtesy of Jang *et al.* [63], ©The Eurographics Association 2006).

showed that both statistical and visual results for comparison are more flexible and appropriate for any data distribution, and specifically EBFs give greater compression and more accurate visual representation of datasets. Although the extension to EBFs gives better approximation and visual accuracy, visual artifact can still be visible for some datasets. In order to reduce these artifacts, the L2-norm based and the H1-norm based cost function were applied in the encoding process to examine their influence to optimization and visualization, whereby the results showed that the L2-norm based cost function provides better visual representations.

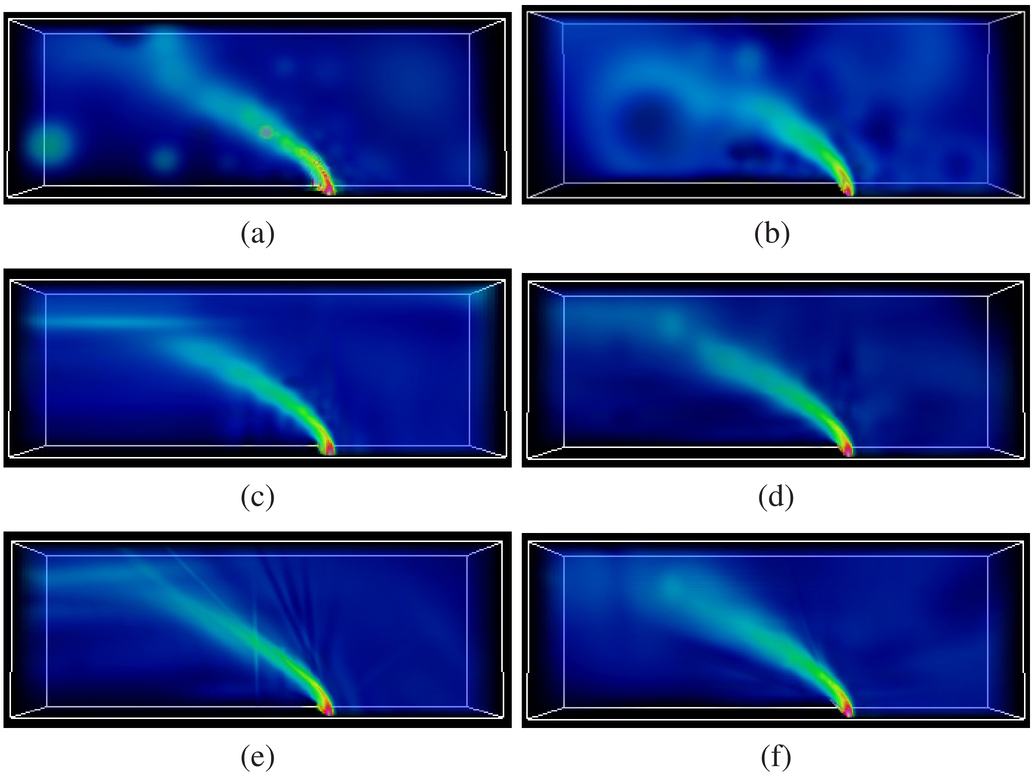


Figure 6.13: Comparison of the Bluntfin dataset encoded with: RBFs (a); RBFs with gradient (b); axis aligned EBFs (c); axis aligned EBFs with gradient (d); arbitrary directional EBFs (e); and arbitrary directional EBFs with gradient (f). (Images courtesy of Jang *et al.* [63], ©The Eurographics Association 2006).

# Chapter 7 Multi-field Techniques in Visualization

This thesis addressed the challenges of illustrating a combination of multi-field data, primarily of scalar and vector nature, and presented new visualization approaches for a multiplicity of application areas. A diagram of the proposed rendering techniques for different application areas is illustrated in Figure 7.1. This work brings all presented methods in a context, and tries to extract guidelines for the implementation of multi-field visualization techniques on GPUs. As the diversity of applications and required solutions for visualization is almost unlimited, not all problems and combinations could be addressed, and thus the given diagram is by no means complete. However, the contribution of this work to the visualization community should inspire other researchers and give them a paradigm or guideline to solve similar multi-field visualization problems.

All processed multi-field data in this thesis consisted of a collection of scalar and vector data. This data was acquired by different modalities and was applied to different application areas for visualization, i.e., (i) flow visualization; (ii) volume visualization; (iii) video visualization; and (iv) encoded data visualization. Therefore, the developed techniques that work one or more of the input fields are shown in Figure 7.1 and include: (1) isosurface rendering; (2) volume rendering; (3) particle tracing; (4) texture advection; and (5) additional color mapping of the extend of an attribute mapped to the result of one of previous techniques. In particular, the achievements and contributions on multi-field visualization in each individual area of research can be summarized as follows:

## Visualization of Multi-field Flow Data

A generic texture-based strategy to visualize an additional variate, such as uncertainty, in time-dependent flow was presented in Chapter 3. As specific examples for this strategy, five novel techniques were proposed: (i) *cross advection*; (ii) *error diffusion*; (iii) *multi-frequency noise*; (iv) *color mapping*; and *uncertainty*

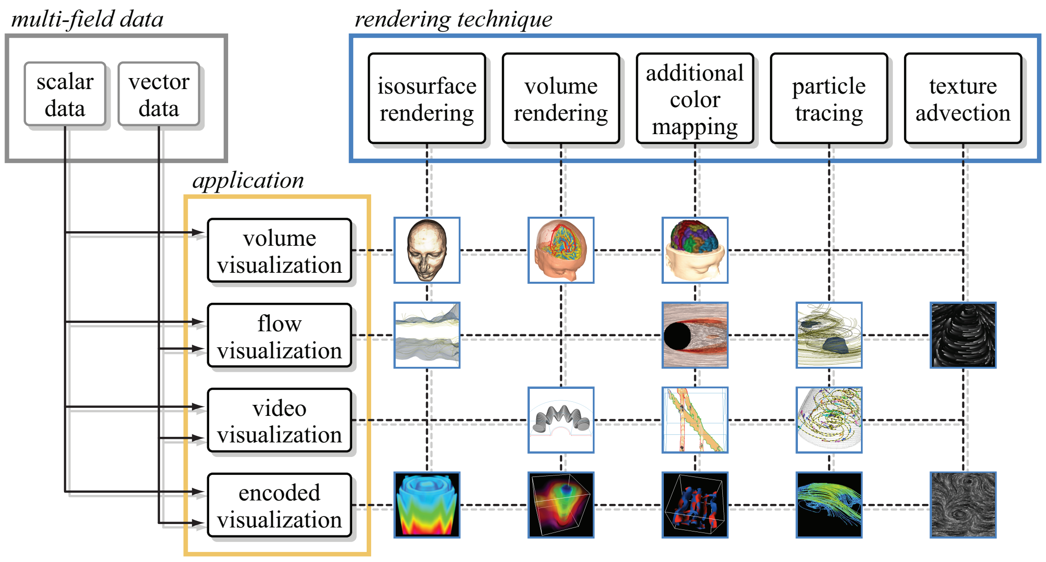


Figure 7.1: Diagram of all developed multi-field rendering techniques applied to several application areas.

*edge detection*. According to underlying uncertainty of the data, these techniques either change the spatial frequency perpendicular to the flow direction, or apply a color mapping to or around the streaklines of injected particles. The main advantage of these techniques is their flexibility and generality, as they can be directly combined with semi-Lagrangian advection. Therefore, they can be applied to any density of texture representation ranging from dense noise-based up to sparse dye-based methods. Moreover, all approaches can be directly mapped to GPUs in order to achieve real-time visualization. In this way, the user can interactively explore the flow field.

In the following of this chapter, a system for the interactive analysis of flow fields, based on a flexible combination of user defined feature criteria was presented. Starting form the extracted feature sets an investigation of the surrounding flow behavior has been employed by placing particles on the characteristic feature hull and tracing streamlines on the field. It was shown that this technique is a useful tool to cope with multivariate fields by representing them on a higher level of abstraction and to give the user the possibility to interactively steer the system during analysis.

## Flexible Multi-volume Visualization

In Chapter 4, a generic framework for combined multi-volume rendering of different modalities was introduced. It was shown that such a generic system is a powerful method for illustrating complex structures in the data. However, programming shader techniques is a very complicated and time consuming task that requires expert knowledge. With the help of the dynamic shader generation technique, hardware shading has advanced to a more modular and flexible procedure. By combining various specified shader nodes, a user without experience in shader programming can easily mix these techniques to achieve the desired visual result. The power of this approach was exemplified on specific medical application datasets. The challenges for implementing three different multi-volume slicing approaches were discussed, as well as the difficulties that occur for multi-volume raycasting. Both algorithms directly utilize the dynamic generation of GPU shaders by applying optimized shaders to the previously segmented scene, which only consider the currently traversed volumes.

**Multi-field Video Visualization**

Chapter 5 described a system designed specifically for real-time video volume visualization. This system is capable of handling multi-field datasets and rendering combined volume and flow visualization, whereby a bricking approach has been found to play a critical role in delivering this technology. Not only does it enable large multi-field datasets to be accommodated in memory-restricted graphics hardware, but it also provides a practical mechanism for visualizing real-time video streams.

An extensive user study and an expert survey were conducted that provided an extensive set of useful data about human factors in video visualization. Through this work some first-hand evaluation as to the effectiveness of different video processing techniques and visualization techniques could be gathered. In particular, the first set of evidence was obtained showing that human observers can learn to recognize types of motion from their visual signatures. Considering that most observers had little knowledge about visualization technology in general, over $80\%$ of them gained $50\%$ or above success rate within a 45 minute learning process. The reduction of response time within a session is significant, while the improvement of accuracy may possibly gain through experiencing video visualization regularly over a period. Some of the findings obtained in this user study indicate the possibility that perspective projection in a video may not necessarily be a major barrier, since human observers can recognize size changes at ease.

The findings of the user study and the expert survey led to the design of an extended system for action-based video visualization. This system combines computer vision techniques with a volumetric visualization of space-time video streams. The computer vision methods include the extraction of persons from background, the application of a motion descriptor for action recognition, and an object-object relationship filter. The visualization component of the system is based on direct volume rendering, extending it to multi-field visualization of action type, relations between objects, and level of uncertainty. The multi-field visualization relies on a special design of transfer functions, additional glyph representations, and the display of snapshots of the video stream.

An evaluation of the system showed that the main strength of the approach is that it combines the best of automatic computer-based video analysis and human-centered visualization. Statistical and computer vision techniques are most suited for low-level analysis such as background extraction and classification of actions. In contrast, human users are highly capable of building semantic information out of the low-level input, for example, complete activities out of low-level actions. In particular, users are able to resolve ambiguous, uncertain classifications from computer vision methods.

**Visualization of Encoded Multi-field Data**

Chapter 6 presented a functional approximation for structured or unstructured scalar, vector, and multi-field data using RBFs and EBFs, enabling them to be efficiently stored and reconstructed on commodity graphics hardware. It was shown that by an extension to EBFs, the approximation and visualization problems common for non-spherical structures can be overcome. The statistical and visual comparison of the results indicated that EBFs are more flexible and appropriate for a variety of different data grids, and give greater compression and more accurate visual representation of datasets. Although EBFs give better approximation and visual accuracy, visual artifact can still be visible for some datasets. In order to reduce these artifacts, the optimization algorithm utilized the L2-norm based and the H1-norm based cost functions that were compared in terms of their influence to optimization and visualization.

The flexibility in visualizing these RBF and EBF encoded datasets was demonstrated on a variety of rendering techniques, namely: texture-based volume rendering, cutting planes, isosurfaces, texture advection, and particle traces. Performing these computations on the GPU also allows for pixel-accurate feature detection and provides a flexible framework for interactive feature exploration, where the feature parameters can be interactively adjusted.

## 7.1 Conclusion

In retrospective, the development of the presented multi-field visualization techniques in this thesis was driven by the rapid progress and accessibility of high-end graphics hardware on the consumer market. Thus, the design of each individual algorithm was specially fitted to the latest available functionality provided by actual GPUs at this very time. Even though a redesign of these algorithms to the latest hardware would lead to some adaption of the code and to an increase of rendering speed by at least a factor of two in most cases, the lessons that have been learnt during development and the essential findings are mostly decoupled from the steadily increasing processing power of graphics hardware and can be described on a more abstract conceptual level.

Nevertheless, exploiting the extreme rasterization power of graphics processing units is always an important aspect for the design of multi-field visualization techniques, as GPU implementations are often superior in rendering performance to comparable CPU implementations. The suggested generic guidelines for the development of new multi-field visualization techniques are based on the four major questions described in Section 1.1. These questions where considered to extract some basic rules for the design of multi-field algorithms and lead to some important findings that are described in the following:

1. *How can multi-field (large-scale) data that has its origin in a broad range of application fields and that can be acquired from various data sources be processed effectively in uncompressed or compressed form?*

An important issue for the design of GPU-based multi-field visualization algorithms is the aspect of large datasets that need to be downloaded to the graphics hardware memory for processing. As the data bandwidth and the memory size on the hardware will always be limited – no matter how powerful the actual hardware is, because the data size and processing requirements are steadily increasing too – this can become a major bottleneck. Throughout the thesis, different approaches to this problem have been demonstrated: (i) the first approach used logical operators to combine $n$ scalar fields to one characteristic set and thus reduced the data to be downloaded and processed on the GPU by the factor of $n - 1$; (ii) the next concept was based on a volume bricking mechanism that was extended to frame-by-frame bricking, to allow a continuous streaming of single video frames of a bricked video volume to the graphics hardware. With this bricking mechanism the continuous rendering of a large video stream as video volume could be realized, without the requirement for the whole data to reside in graphics memory; and (iii) the last approach presented for an effective data layout exploited functional approximation of multi-field data, to reduce the data size and enhance the evaluation by utilizing the SIMD fragment shader operations. It was shown that multi-field datasets can be approximated by spherical and ellipsoidal basis functions and stored with less memory requirement, so that the whole data fits in graphics memory for processing. This is advantageous for the analysis of global features or phenomena, where all data is needed for evaluation at once. Summarizing, the processing of large-scale multi-field data can be significantly improved by taking into account three aspects: data reduction, data partitioning and efficient data layout. Especially the former two aspects have impact on the next question.

2. *What has to be considered to develop novel algorithms to extract important features or signatures from the raw multi-field data and visualize a combination of them most clearly at a time?*

For some cases it can be an advantage to compute certain features of the data on-the-fly in the visualization stage. This is however only possible, if the whole dataset fits into hardware memory, or only a subset of the data needs to be analyzed. If the analyst is interested in different features of separate fields that do not fit into memory, a different strategy needs to be considered, which is tightly coupled with the solutions to question 1. Further, this issue can be addressed by thinning out the data, applying feature extraction algorithms on each individual field and pass only the extracted signatures of one or more features to the visualization system. The visualization system is then assigned to illustrate these features in best way possible. This can be realized by using different metaphors or graphical representations for each feature, such as isosurface, streamlines, glyphs, or a translucent volumetric illustration of the whole domain, whereby individual variates need to be clearly distinguishable by the user. To further enhance perception, several diverse mapping techniques can de utilized to map the extent of various features to different visual cues, to facilitate a clear separation. The aspect of perception also pours into the solution to the following point and is taken on there.

3. *In which way can errors and uncertainties inhering the data from faulty measurements, algorithmic instabilities or reliability of the algorithms be included into visualization?*

Not only the existence and the extraction of features is important to understand certain phenomena in multi-field data. It is also of great relevance to include uncertainties inhering the data from faulty measurements, algorithmic instabilities or the reliability of an algorithm into the visualization. This information is significant for a reliable analysis process and should be included as auxiliary variate. As shown throughout the chapters, this additional information can be a measured value, a derived value or a statistical value and can be displayed in correspondence with the original visualization, or manipulate this visualization. The critical aspect for the illustration of this additional variate is its clear perception, and thus, a visual separation of all other variates. This can be achieved by mapping this value to a different visual channel, like a different color value or unequal hue. Otherwise, the unreliable feature or signature can be surrounded by a separate silhouette that expresses the extent of uncertainty. In all cases, it is important to discretely illustrate all variates and to not visually occlude each other and cause a loss of information.

4. *How can we creatively map new visualization algorithms to graphics hardware, exploiting their computation power to effectively utilize interactive visualization?*

Finally, one of the most important aspects in visualization is that interactivity is often a key aspect for understanding complex features or phenomena in large multi-field data and thus, interactivity should be one of the substantial goals for designing new algorithms. A major backup to fulfill this goal arises from the current advances in hardware development. Besides the central processing unit, modern desktop PCs are commonly equipped with high-performance graphics processing units, whose computation power is a valuable aid for the acceleration of visualization algorithms. This work demonstrated that all proposed techniques can be mapped to graphics hardware architectures and benefit from these implementations in terms of rendering performance. Nevertheless, it is not profitable to source out the whole computation to the GPU and completely release the CPU. A sophisticated load balancing between CPU and GPU is essential for an adequate solution. Therefore, preprocessing or data storage of intermediate results should be taken into account whenever possible.

Concluding it can be said that although this work investigated in several different application areas that produce a plethora of unequal multi-field data, the requirements for GPU-based algorithmic design overlap for the most parts. Following the suggestions given in this section can lead to a successful development of multi-field visualization algorithms and give engineers and scientists a well-founded scaffolding for the design of new multi-field visualization solutions.

## 7.2 Future Challenges

Based on the experience gained during the compilation of this thesis and the extracted generic guidelines for the algorithmic design, some challenges that will arise for further development can be prognosticated. First of all, increasing the availability of multi-field visualization methods is a first step towards an improvement of the acceptance of such techniques in scientific and economic environments. To achieve a broader distribution, further steps are required. Second, the user handling of multi-field visualization techniques must be improved in terms of user interaction and learnability of such systems. The operation of such an application must be further decoupled from the detailed knowledge of the underlying algorithms, as addressed in Chapter 4, and these systems need to allow for an exact reproducibility of previously obtained solutions. Especially in economic systems, reproducibility is extremely important for the verification and comparison of examination results.

Beyond that, a deeper investigation in automatic feature detection algorithms is necessary to disburden the user from having expert technical knowledge and intensify his responsibility in problem identification and decision making. Therefore, we need a tighter coupling between simulation algorithms, extraction algorithms and visualization algorithms, to allow computational steering on a higher abstraction level and achieve more interactivity. Further speedup can be produced by exploiting multicore GPUs, which automatically leads to new algorithmic design requirements and novel solutions in terms of a distributed visualization.

# Bibliography

[1] B. Aubert-Broche, M. Griffin, G. B. Pike, A. C. Evans, and D. L. Collins. Twenty new digital brain phantoms for creation of validation image data bases. *IEEE Transactions on Medical Imaging*, 25(5):1410–1416, 2006.

[2] D. C. Banks and B. A. Singer. Vortex tubes in turbulent flows: Identification representation, reconstruction. In *Proceedings of IEEE Visualization*, pages 132–139, 1994.

[3] D. C. Banks and B. A. Singer. A predictor-corrector technique for visualizing unsteady flow. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):151–163, 1995.

[4] M. F. Barnsley, A. Jacquin, F. Malassenet, L. Reuter, and A. D. Sloan. Harnessing chaos for image synthesis. 22(4):131–140, 1988.

[5] J. L. Barron, D. J. Fleet, and S. S. Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1):43–77, 1994.

[6] E. P. Bennett and L. McMillan. Proscenium: a framework for spatiotemporal video editing. In *Proceedings of ACM Multimedia*, pages 177–184, 2003.

[7] I. Boada, I. Navazo, and R. Scopigno. Multiresolution volume visualization with a texture-based octree. *The Visual Computer*, 17(3):185–197, 2001.

[8] A. F. Bobick and J. W. Davis. The recognition of human movement using temporal templates. *IEEE Transactions on Pattern Analysis Machine Intelligence*, 23(3):257–267, 2001.

[9] R. P. Botchen, S. Bachthaler, F. Schick, M. Chen, G. Mori, D. Weiskopf, and T. Ertl. Action-based multifield video visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):885–899, 2008.

[10] R. P. Botchen, M. Chen, D. Weiskopf, and T. Ertl. GPU-assisted multifield video volume visualization. In *Proceedings of IEEE / VGTC Volume Graphics*, pages 47–54, 2006.

[11] R. P. Botchen, A. Lauser, D. Weiskopf, and T. Ertl. Flow feature visualization using logical operators on multivariate fields. In *Electronic Proceedings of International Symposium on Flow Visualization*, 2008.

[12] R. P. Botchen, D. Weiskopf, and T. Ertl. Texture-based visualization of uncertainty in flow fields. In *Proceedings of IEEE Visualization*, pages 647–654, 2005.

[13] R. P. Botchen, D. Weiskopf, and T. Ertl. Interactive visualization of uncertainty in flow fields using texture-based techniques. In *Electronic Proceedings of International Symposium on Flow Visualization*, 2006.

[14] M. A. Branch, T. F. Coleman, and Y. A. Li. A subspace, interior, and conjugate gradient method for large-scale bound-constrained minimization problems. *SIAM Journal on Scientific Computing*, 21(1):1–23, 1999.

[15] R. Brown. Animated visual vibrations as an uncertainty visualisation technique. In *Proceedings of GRAPHITE*, pages 84–89, 2004.

[16] S. Bruckner and M. E. Gröller. VolumeShop: An interactive system for direct volume illustration. In *Proceedings of IEEE Visualization*, pages 671–678, 2005.

[17] B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings of the Symposium on Volume Visualization*, pages 91–98, 1994.

[18] B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In *Proceedings of ACM SIGGRAPH*, pages 263–270, 1993.

[19] W. Cai and G. Sakas. Data intermixing and multi-volume rendering. *Computer Graphics Forum*, 18(3):359–368, 1999.

[20] G. Calafiore. Approximation of n-dimensional data using spherical and ellipsoidal primitives. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 32(2):1083–4427, 2002.

[21] B. P. Carneiro, C. T. Silva, and A. E. Kaufman. Tetra-cubes: an algorithm to generate 3D isosurfaces based upon tetrahedra. In *Proceedings of the IX SIBGRAPI International Conference*, pages 205–210, 1996.

[22] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and Representation of 3D Objects With Radial Basis Functions. In *Proceedings of ACM SIGGRAPH*, pages 67–76, 2001.

[23] M. Chen. Combining point clouds and volume objects in volume scene graphs. In *Proceedings of IEEE / VGTC Volume Graphics*, pages 127–135, 2005.

[24] M. Chen, R. P. Botchen, R. R. Hashim, D. Weiskopf, T. Ertl, and I. M. Thornton. Visual signatures in video visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1093–1100, 2006.

[25] M. Chen and J. V. Tucker. Constructive volume geometry. *Computer Graphics Forum*, 19(4):281–293, 2000.

[26] C. S. Co, B. Heckel, H. Hagen, B. Hamann, and K. I. Joy. Hierarchical clustering for unstructured volumetric scalar fields. In *Proceedings of IEEE Visualization 2003*, 2003.

[27] R. T. Collins, A. J. Lipton, and T. Kanade. Special section on video surveillance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):745–746, 2000.

[28] C. D. Correa, D. Silver, and M. Chen. Feature aligned volume manipulation for illustration and visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1069–1076, 2006.

[29] R. Cutler, C. Shekhar, B. Burns, R. Chellappa, R. Bolles, and L. Davis. Monitoring human and vehicle activities using airborne video. In *Proceedings of 28th Applied Imagery Pattern Recognition Workshop (AIPR)*, pages 146–153, 1999.

[30] G. W. Daniel and M. Chen. Video visualization. In *Proceedings of IEEE Visualization*, pages 409–416, 2003.

[31] W. C. de Leeuw and J. J. van Wijk. Enhanced spot noise for vector field visualization. In *Proceedings of IEEE Visualization*, pages 233–239, 1995.

[32] H. Doleisch, M. Gasser, and H. Hauser. Interactive feature specification for focus+context visualization of complex simulation data. In *Proceedings of VISSYM*, pages 239–248, 2003.

[33] D. J. Dorney. Reynolds-Averaged Navier-Stokes studies of low Reynolds number effects on the losses in a low pressure turbine. Final Contractor Report G-NAG3-1668, Lewis Research Center, 1996.

[34] D. Dovey. Vector plots for irregular grids. In *Proceedings of IEEE Visualization*, pages 248–253, 1995.

[35] A. A. Efros, A. C. Berg, G. Mori, and J. Malik. Recognizing action at a distance. In *Proceedings of IEEE International Conference on Computer Vision*, pages 726–733, 2003.

[36] K. Engel, M. Hadwiger, J. M. Kniss, C. Rezk-Salama, and D. Weiskopf. *Real-Time Volume Graphics*. AK Peters, 2006.

[37] K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of EUROGRAPHICS / ACM SIGGRAPH Workshop on Graphics Hardware*, pages 9–16, 2001.

[38] G. Erlebacher, B. Jobard, and D. Weiskopf. Flow textures: High-resolution flow visualization. In C. D. Hansen and C. R. Johnson, editors, *The Visualization Handbook*, pages 279–293. Elsevier, Amsterdam, 2005.

[39] Exa Corporation. PowerVIZ specifications, 2001. http://www.exa.com/pdf/PowerVIZscreen.pdf.

[40] S. Fels, E. Lee, and K. Mase. Techniques for interactive video cubism. In *Proceedings of 8th ACM International Conference on Multimedia (Posters)*, pages 368–370, 2000.

[41] R. B. Fisher. The PETS04 surveillance ground-truth data sets. In *Proceedings of 6th IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*, pages 1–5, 2004.

[42] L. K. Forssell and S. D. Cohen. Using line integral convolution for flow visualization: Curvilinear grids, variable-speed animation, and unsteady flows. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):133–141, 1995.

[43] O. Frederich, E. Wassen, and F. Thiele. Flow simulation around a finite cylinder on massively parallel computer architecture. In *Proceedings of the International Conference on Parallel Computational Fluid Dynamics 2005*, 2005.

[44] W. Freeman, J. O. K. Tanaka, and K. Kyuma. Computer vision for computer games. In *Proceedings of IEEE 2nd Intl. Conf. on Automatic Face and Gesture Recognition*, pages 100–105, 1996.

[45] D. M. Gavrila. The visual analysis of human movement: A survey. *Computer Vision and Image Understanding: CVIU*, 73(1):82–98, 1999.

[46] A. A. Goshtasby. Grouping and parameterizing irregularly spaced points for curve fitting. *ACM Transactions on Graphics (TOG)*, 19(3):185–203, 2000.

[47] N. Grammalidis and M. G. Strintzis. Head detection and tracking by 2-D and 3-D ellipsoid fitting. In *Proceedings of IEEE Computer Graphics International Conference*, pages 221–226, 2000.

[48] H. Griethe and H. Schumann. The visualization of uncertain data: Methods and problems. In *Proceedings of SimVis*, pages 143–156, 2006.

[49] S. Grimm, S. Bruckner, A. Kanitsar, and E. Gröller. Flexible direct multi-volume rendering in interactive scenes. In *Proceedings of Vision, Modeling, and Visualization*, pages 379–386, 2004.

[50] R. Grosso and T. Ertl. Mesh optimization and multilevel finite element approximations. In *In Proceedings of Visualization and Mathematics*, page 1930, 1997.

[51] S. Guthe, S. Gumhold, and W. Straßer. Interactive visualization of volumetric vector fields using texture based particles. In *Proceedings of WSCG 2002 Conference*, pages 33–41, 2002.

[52] R. B. Haber and D. A. McNabb. Visualization idioms: A conceptual model for scientific visualization systems. In *Visualization in Scientific Computing, pages =*.

[53] M. Hadwiger, C. Berger, and H. Hauser. High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. In *Proceedings of IEEE Visualization*, pages 301–308, 2003.

[54] R. Haimes and D. Kenwright. On the velocity gradient tensor and fluid feature extraction. Technical Report 99-3288, AIAA, Norfolk, Virginia, 1999.

[55] R. Hardy. Theory and applications of the multiquadric-biharmonic method. *Computers and Mathematics with Applications*, 19:163–2082, 1990.

[56] C. G. Healey. Choosing effective colours for data visualization. In *Proceedings of IEEE Visualization*, pages 263–270, 1996.

[57] A. Hertzmann and K. Perlin. Painterly rendering for video and interaction. In *Proceedings of 1st International Symposium on Non-Photorealistic Animation and Rendering*, pages 7–12, 2000.

[58] L. Hesselink and J. Helman. Evaluation of flow topology from numerical data. *AIAA*, (87-1811), 1987.

[59] K. D. Hinsch. Particle image velocimetry. In R. S. Sirohi, editor, *Speckle Metrology*, pages 235–324. Marcel Dekker, New York, 1993.

[60] B. K. P. Horn and B. G. Schunk. Determining optical flow. *Artificial Intelligence*, 17:185–201, 1981.

[61] A. Huang, G. E. Farin, D. P. Baluch, and D. G. Capco. Thin structure segmentation and visualization in three-dimensional biomedical images: A shape-based approach. *IEEE Transactions on Visualization and Computer Graphics*, 12(1):93–102, 2006.

[62] V. Interrante and C. Grosch. Visualizing 3D flow. *IEEE Computer Graphics & Applications*, 18(4):49–53, 1998.

[63] Y. Jang, R. P. Botchen, A. Lauser, D. S. Ebert, K. P. Gaither, and T. Ertl. Enhancing the interactive visualization of procedurally encoded multifield data with ellipsoidal basis functions. *Computer Graphics Forum*, 25(3):587–596, 2006.

[64] Y. Jang, M. Weiler, M. Hopf, J. Huang, D. S. Ebert, K. P. Gaither, and T. Ertl. Interactively visualizing procedurally encoded scalar fields. In *Proceedings of EUROGRAPHICS / IEEE TCVG Symposium on Visualization*, 2004.

[65] J. Jeong and F. Hussain. On the identification of a vortex. *Journal of Fluid Mechanics*, (285):69–94, 1995.

[66] B. Jobard, G. Erlebacher, and M. Y. Hussaini. Lagrangian-Eulerian advection of noise and dye textures for unsteady flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):211–222, 2002.

[67] B. Jobard and W. Lefer. The motion map: Efficient computation of steady flow animations. In *Proceedings of IEEE Visualization*, pages 323–328, 1997.

[68] B. Jobard and W. Lefer. Unsteady flow visualization by animating evenly-spaced streamlines. In *Proceedings of EUROGRAPHICS*, pages 31–40, 2000.

[69] C. R. Johnson. Top scientific research problems. *IEEE Computer Graphics & Applications*, 24(4):13–17, 2004.

[70] C. R. Johnson and A. R. Sanderson. A next step: Visualizing errors and uncertainty. *IEEE Computer Graphics & Applications*, 23(5):6–10, 2003.

[71] A. Kaufman and N. Stolte. Discrete implicit surface models using interval arithmetics. In *Proceedings of 2nd CGC Workshop on Computational Geometry*, 1997.

[72] D. N. Kenwright. Automatic detection of open and closed separation and attachment lines. In *Proceedings of IEEE Visualization*, pages 151–158, 1998.

[73] G. Kindlmann, E. Reinhard, and S. Creem. Face-based luminance matching for perceptual colormap generation. In *Proceedings of IEEE Visualization*, pages 299–306, 2002.

[74] M. Kiu and D. C. Banks. Multi-frequency noise for LIC. In *Proceedings of IEEE Visualization*, pages 121–126, 1996.

[75] R. V. Klassen and S. J. Harrington. Shadowed hedgehogs: A technique for visualizing 2D slices of 3D vector fields. In *Proceedings of IEEE Visualization*, pages 148–153, 1991.

[76] A. Klein, P. Sloan, A. Colburn, A. Finkelstein, and M. Cohen. Video cubism. Technical report, Microsoft Research Technical Report MSR-TR-2001-45, 2001.

[77] J. M. Kniss, R. V. Uitert, A. Stephens, G.-S. Li, T. Tasdizen, and C. Hansen. Statistically quantitative volume visualization. In *Proceedings of IEEE Visualization*, pages 287–294, 2005.

[78] A. N. Kolmogorov. The local structure of turbulence in incompressible viscous fluid for very large Reynolds numbers. *C. R. (Doklady) Acad. Sci. URSS (N.S.)*, 30:301–305, 1941.

[79] O. Konrad-Verse, A. Littmann, and B. Preim. Virtual resection with a deformable cutting plane. In *Proceedings of SimVis*, pages 203–214, 2004.

[80] L. H. Koopmans. *The Spectral Analysis of Time Series*. Academic Press, 1995.

[81] K. Kreeger and A. Kaufman. Mixing translucent polygons with volumes. In *Proceedings of IEEE Visualization*, pages 191–198, 1999.

[82] W. Krueger. The application of transport theory to visualization of 3D scalar data fields. In *Proceedings of IEEE Visualization*, pages 273–280, 1990.

[83] J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *Proceedings of IEEE Visualization*, pages 287–292, 2003.

[84] Y. Kurzion and R. Yagel. Interactive space deformation with hardware-assisted rendering. *IEEE Computer Graphics & Applications*, 17(5):66–77, 1997.

[85] R. S. Laramee, H. Hauser, H. Doleisch, B. Vrolijk, F. H. Post, and D. Weiskopf. The state of the art in flow visualization: Dense and texture-based techniques. *Computer Graphics Forum*, 23(2):143–161, 2004.

[86] V. Lavrenko, S. L. Feng, and R. Manmatha. Statistical models for automatic video annotation and retrieval. In *Proceedings of the IEEE ICASSP International Conference on Acoustics, Speech and Signal Processing*, pages 17–21, 2004.

[87] Y. Levy, D. Degani, and A. Seginer. Graphical visualization of vortical flows by means of helicity. *AIAA Journal*, 28:1347–1352, 1990.

[88] Z. P. Liu and R. J. Moorhead. AUFLIC: An accelerated algorithm for unsteady flow line integral convolution. In *Proceedings of EROGRAPHICS / IEEE TCVG Symposium on Visualization*, pages 43–52, 2002.

[89] S. K. Lodha, A. Pang, R. E. Sheehan, and C. M. Wittenbrink. UFLOW: Visualizing uncertainty in fluid flow. In *Proceedings of IEEE Visualization*, pages 249–254, 1996.

[90] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics Conference Series*, 21(4):163–169, 1987.

[91] D. Lovely and R. Haimes. Shock detection from computational fluid dynamics results, 1999.

[92] A. Lu, C. J. Morris, D. S. Ebert, P. Rheingans, and C. D. Hansen. Non-photorealistic volume rendering using stippling techniques. In *Proceedings of IEEE Visualization*, pages 211–218, 2002.

[93] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of DARPA Image Understanding Workshop*, pages 674–679, 1981.

[94] K. Madsen, H. B. Nielsen, and O. Tingleff. Methods for non-linear least squares problems, 1999.

[95] K. Mahrous, J. Bennett, G. Scheuermann, B. Hamann, and K. Joy. Topological segmentation in three dimensional vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 2(10):198–205, 2004.

[96] X. Mao, M. Kikukawa, N. Fujita, and A. Imamiya. Line integral convolution for 3D surfaces. In *Proceedings of EUROGRAPHICS Workshop on Visualization*, pages 57–70, 1997.

[97] D. Marcum and K. Gaither. Solution adaptive unstructured grid generation using pseudo pattern recognition techniques. *AIAA*, (97-1860), 1997.

[98] S. R. Marschner and R. J. Lobb. An evaluation of reconstruction filters for volume rendering. In *Proceedings of IEEE Visualization*, pages 100–107, 1994.

[99] O. Marxen, M. Lang, U. Rist, and S. Wagner. A combined experimental/numerical study of unsteady phenomena in a laminar separation bubble. *Flow, Turbulence and Combustion*, 71(1-4):133–146, 2003.

[100] J. Mattson and M. Simon. *The Pioneers of NMR and Magnetic Resonance in Medicine: The Story of MRI.* Bar-Ilan University Press, 1996.

[101] N. Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995.

[102] N. Max and B. Becker. Flow visualization using moving textures. In *Proceedings of ICASW / LaRC Symposium on Visualizing Time-Varying Data*, pages 77–87, 1995.

[103] D. Meyer. *Discrete numerische Simulation nichtlinearer Transitionsmechanismen in der Strömungsgrenzschicht einer ebenen Platte*. Phd thesis, Luft- und Raumfahrttechnik, Universität Stuttgart, 2003.

[104] T. B. Moeslund and E. Granum. A survey of computer vision-based human motion capture. *Computer Vision and Image Understanding: CVIU*, 81(3):231–268, 2001.

[105] D. R. Nadeau. Volume scene graphs. In *Proceedings of IEEE Symposium on Volume Visualization*, pages 49–56, 2000.

[106] K. G. Nguyen and D. Saupe. Rapid high quality compression of volume data for visualization. *Computer Graphics Forum*, 20(3):49–56, 2001.

[107] G. M. Nielson. Scattered data modeling. *IEEE Computer Graphics & Applications*, 13(1):60–70, 1993.

[108] G. M. Nielson and B. Hamann. The asymptotic decider: resolving the ambiguity in marching cubes. In *Proceedings of IEEE Visualization*, pages 83–91, 1991.

[109] S. Ogawa, T. M. Lee, A. R. Kay, and D. W. Tank. Brain magnetic resonance imaging with contrast dependent on blood oxygenation. *Proceedings of National Academy of Sciencs USA*, 87(24):9868–9872, 1990.

[110] R. Oi, M. Magnor, and K. Aizawa. A solid-state, simultaneous wide angle-detailed view video surveillance camera. In *Proceedings of Vision, Modeling and Visualization*, pages 329–336, 2003.

[111] K. Okada, D. Comaniciu, and A. Krishnan. Robust anisotropic gaussian fitting for volumetric characterization of pulmonary nodules in multislice ct. *IEEE Transactions on Medical Imaging*, 24(3):409–423, 2005.

[112] A. T. Pang, C. M. Wittenbrink, and S. K. Lodha. Approaches to uncertainty visualization. *The Visual Computer*, 13(8):370–390, 1997.

[113] N. V. Patel and I. K. Sethi. Video shot detection and characterization for video databases. *Pattern Recognition, Special Issue on Multimedia*, 30(4):583–592, 1997.

[114] R. Peikert and M. Roth. The parallel vectors operator - A vector field visualization primitive. In *Proceedings of IEEE Visualization*, pages 263–270, 1999.

[115] T. Porter and T. Duff. Compositing digital images. *ACM SIGGRAPH Computer Graphics*, 183(3):253–259, 1984.

[116] F. H. Post, B. Vrolijk, H. Hauser, R. S. Laramee, and H. Doleisch. The state of the art in flow visualization: Feature extraction and tracking. *Computer Graphics Forum*, 22(4):775–792, 2003.

[117] A. Prasad. Particle image velocimetry. *Current Science*, 79(1):101–110, 2000.

[118] L. Q, W. D., P. R., V. J., G. G., and W. J. Implicit fitting using radial basis functions with ellipsoid constraint. *Computer Graphics Forum*, 23(1):55–69, 2004.

[119] C. Rao, A. Yilmaz, and M. Shah. View-invariant representation and recognition of actions. *International Journal of Computer Vision*, 50(2):203–226, 2002.

[120] C. Rezk-Salama, M. Scheuering, G. Soza, and G. Greiner. Fast volumetric deformation on general purpose hardware. In *Proceedings of EUROGRAPHICS / ACM SIGGRAPH Workshop on Graphics Hardware*, pages 17–24, 2001.

[121] P. J. Rhodes, R. S. Laramee, R. D. Bergeron, and T. M. Sparr. Uncertainty visualization methods in isosurface rendering. In *Proceedings of EUROGRAPHICS Short Papers*, pages 83–88, 2003.

[122] F. Rößler, R. P. Botchen, and T. Ertl. Dynamic shader generation for flexible multi-volume visualization. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 17–24, 2008.

[123] F. Rößler, R. P. Botchen, and T. Ertl. Dynamic shader generation for GPU-based multi-volume raycasting. *IEEE Computer Graphics & Applications*, 28(5):66–77, 2008.

[124] R. J. Rost and J. M. Kessenich. *OpenGL Shading Language*. Addison-Wesley, 2003.

[125] S. Röttger, S. Guthe, D. Weiskopf, and T. Ertl. Smart Hardware-Accelerated Volume Rendering. In *Proceedings of EUROGRAPHICS / IEEE VGTC Symposium on Visualization*, pages 231–238, 2003.

[126] T. Salzbrunn and G. Scheuermann. Streamline predicates. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1601–1612, 2006.

[127] W. Sander and B. Weigand. Direct numerical simulation and analysis of instability enhancing parameters in liquid sheets at moderate reynolds numbers. *Geophysical Research Letters*, (to appear), 2008.

[128] N. Sauber, H. Theisel, and H.-P. Seidel. Multifield-graph: An approach to visualizing correlations in multifield scalar data. In *Proceedings of IEEE Visualization*, pages 917–924, 2006.

[129] V. V. Savchenko, A. A. Pasko, O. G. Okunev, and T. L. Kunii. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum*, 14(4):181–188, 1995.

[130] R. Scardovelli and . S. Zaleski. Direct numerical simulation of free-surface and interfacial flow. *Annual Review of Fluid Mechanics*, 31:567–603, 1999.

[131] T. Schafhitzel, J. Vollrath, J. Gois, D. Weiskopf, A. Castelo, and T. Ertl. Topology-preserving lambda2-based vortex core line detection for flow visualization. *Computer Graphics Forum*, 27(3):1023–1030, 2008.

[132] T. Schafhitzel, D. Weiskopf, and T. Ertl. Interactive exploration of unsteady 3D flow with linked 2D/3D texture advection. In *Proceedings of the 3rd International Conference on Coordinated and Multiple Views in Exploratory Visualization*, pages 96–105, 2005.

[133] O. Schall, A. Belyaev, and H.-P. Seidel. Robust filtering of noisy scattered point data. In *Proceedings of EUROGRAPHICS Symposium on Point-Based Graphics*, pages 71–77, 2005.

[134] I. O. Sebe, J. Hu, S. You, and U. Neumann. 3D video surveillance with augmented virtual environments. In *Proceedings of IWVS '03: First ACM SIGMM international workshop on Video surveillance*, pages 107–112, 2003.

[135] E. Shechtman and M. Irani. Space-time behavior based correlation. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 405–412, 2005.

[136] H.-W. Shen, C. R. Johnson, and K.-L. Ma. Visualizing vector fields using line integral convolution and dye advection. In *Proceedings of IEEE Symposium on Volume Visualization*, pages 63–70, 1996.

[137] H. W. Shen and D. L. Kao. UFLIC: A line integral convolution algorithm for visualizing unsteady flows. In *Proceedings of IEEE Visualization*, pages 317–323, 1997.

[138] J. Smagorinsky. General circulation experiments with the primitive equations. *Monthly Weather Review*, 91(3):99–164, 1963.

[139] C. G. M. Snoek and M. Worring. Multimodal video indexing: A review of the state-of-the-art. *Multimedia Tools and Applications*, 25(1):5–35, 2005.

[140] Y. Song, L. Goncalves, and P. Perona. Unsupervised learning of human motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(7):814–827, 2003.

[141] P. R. Spalart, W. H. Jou, M. Strelets, and S. R. Allmaras. Comments on the feasibility of LES for wings, and on a hybrid RANS/LES approach. In *Proceedings of 1st AFOSR International Conference on DNS/LES*, pages 137–147, 1997.

[142] J. Stam. Stable fluids. In *Proceedings of ACM SIGGRAPH*, pages 121–128, 1999.

[143] S. Stegmaier and T. Ertl. A graphics hardware-based vortex detection and visualization system. In *Proceedings of IEEE Visualization*, pages 195–202, 2004.

[144] S. Stegmaier, M. Strengert, T. Klein, and T. Ertl. A simple and flexible volume rendering framework for graphics-hardware–based raycasting. In *Proceedings of EUROGRAPHICS / IEEE VGTC Workshop on Volume Graphics*, pages 187–195, 2005.

[145] D. Sujudi and R. Haimes. Identification of swirling flow in 3D vector fields. Technical Report 95-1715, Deptartment of Aeronautics and Astronautics, MIT, Cambridge, 1995.

[146] Z. Sun and W. Jiang. Diagnostic value of multislice computed tomography angiography in coronary artery disease: A meta-analysis. *European Journal of Radiology*, 60(2):279–286, 2006.

[147] A. Sundquist. Dynamic line integral convolution for visualizing streamline evolution. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):273–283, 2003.

[148] N. Svakhine, Y. Jang, D. S. Ebert, and K. P. Gaither. Illustration and photography inspired visualization of flows and volumes. In *Proceedings of IEEE Visualization*, pages 687–694, 2005.

[149] E. Tejada, J. P. Gois, L. G. Nonato, A. Castelo, and T. Ertl. Hardware-accelerated extraction and rendering of point set surfaces. In *Proceedings of EUROGRAPHICS / IEEE VGTC Symposium on Visualization*, pages 21–28, 2006.

[150] E. Tejada, T. Schafhitzel, and T. Ertl. Hardware-accelerated point-based rendering of surfaces and volumes. In *Proceedings of International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, pages 41–48, 2007.

[151] A. M. K. Thomas, A. K. Banerjee, and U. Busch. *Classic papers in modern diagnostic radiology*. Springer, 2005.

[152] J. J. Thomas and K. A. Cook. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. IEEE Press, 2005.

[153] G. M. Treece, R. W. Prager, and A. H. Gee. Regularised marching tetrahedra: improved iso-surface extraction. *Computers & Graphics*, 23(4):583–598, 1999.

[154] G. Turk and J. F. O'Brien. Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics (TOG)*, 21(4):855–873, 2002.

[155] J. J. van Wijk. Image based flow visualization. *ACM Transactions on Graphics (TOG)*, 21(3):745–754, 2002.

[156] J. E. Vollrath, D. Weiskopf, and T. Ertl. A generic software framework for the GPU volume rendering pipeline. In *Proceedings of Vision, Modeling, and Visualization*, pages 391–398, 2005.

[157] Y. Wang, D. M. Krum, E. M. Coelho, and D. A. Bowman. Contextualized videos: Combining videos with environment models to support situational understanding. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1568–1575, 2007.

[158] C. Ware. Color sequences for univariate maps. *IEEE Computer Graphics & Applications*, 8(5):41–49, 1988.

[159] C. Ware. *Information Visualization: Perception for Design*. Elsevier, Amsterdam, second edition, 2004.

[160] R. Wegenkittl, E. Gröller, and W. Purgathofer. Animating flow fields: Rendering of oriented line integral convolution. In *Proceedings of Computer Animation*, pages 15–21, 1997.

[161] M. Weiler, R. P. Botchen, S. Stegmeier, T. Ertl, J. Huang, Y. Jang, D. S. Ebert, and K. P. Gaither. Hardware-assisted feature analysis of procedurally encoded multifield volumetric data. *IEEE Computer Graphics & Applications*, 25(5):72–81, 2005.

[162] D. Weiskopf. Dye advection without the blur: A level-set approach for texture-based visualization of unsteady flow. *Computer Graphics Forum (Proceedings of EUROGRAPHICS)*, 23(3):479–488, 2004.

[163] D. Weiskopf, R. P. Botchen, and T. Ertl. Interactive visualization of divergence in unsteady flow by level-set dye advection. In *Proceedings of SimVis*, pages 221–232, 2005.

[164] D. Weiskopf, K. Engel, and T. Ertl. Interactive clipping techniques for texture-based volume visualization and volume shading. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):298–312, 2003.

[165] D. Weiskopf and G. Erlebacher. Overview of flow visualization. In C. D. Hansen and C. R. Johnson, editors, *The Visualization Handbook*, pages 261–278. 2005.

[166] D. Weiskopf, M. Hopf, and T. Ertl. Hardware-accelerated visualization of time-varying 2D and 3D vector fields by texture advection via programmable per-pixel operations. In *Proceedings of Vision, Modeling, and Visualization*, pages 439–446, 2001.

[167] D. Weiskopf, T. Schafhitzel, and T. Ertl. Texture-based visualization of 3D unsteady flow by real-time advection and volumetric illumination. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):569–582, 2007.

[168] R. Westermann and T. Ertl. Efficiently using graphics hardware in volume rendering applications. *ACM SIGGRAPH Computer Graphics Conference Series*, 32(4):169–179, 1998.

[169] B. Wilson, E. B. Lum, and K.-L. Ma. Interactive multi-volume visualization. In *Proceedings of Computational Science-Part II*, pages 102–110, 2002.

[170] C. M. Wittenbrink, A. T. Pang, and S. K. Lodha. Glyphs for visualizing uncertainty in vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):266–279, 1996.

[171] G. Wyvill, C. McPheeters, and B. Wyvill. Data structure for soft objects. *The Visual Computer*, 2(4):227–234, 1986.

[172] L. Zelnik-Manor and M. Irani. Event-based video analysis. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition.*, 2(2):123–130, 2001.

[173] Y. Zhang, R. Rohling, and D. K. Pai. Direct surface extraction from 3D freehand ultrasound images. In *Proceedings of IEEE Visualization*, pages 45–52, 2002.